au3Irr2 brings together the ease of **AutoIt** with the power of the 3D engine **Irrlicht**. It covers a whole bunch of features to create serious 3D applications as well as nice games, useable both by beginners and more experienced to learn, prototype, and realise ideas without a big overhead.

Also au3Irr2 is still work in progress, it already provides well over 260 commands that cover features like Bitmaps, 3D models, Animation, Collision, Scene management, Maps, Terrains, Cameras, Lights, special effects, GUI and more. Additionally, there are around 100 well commented examples showing the different features and their usage.

Technically, au3Irr2 is a 'wrapper for a wrapper', based upon Frank Dodd's great **FreeBasic Irrlicht Wrapper**. Standing on this mature project au3Irr2 has really good potential to grow and prosper. Especially if you use it and be active in the [forum](http://example.com)!
How to start

Pretty easy. Of course you need installed AutoIt, also full version of SciTE is highly recommend.

- Extract the au3Irr2.zip to whereever you like.
- Run the 'example launcher' to run & enjoy the examples. Open them via the launcher with SciTE to see how it is done.
- Use the helpfile - it lists all working functions, so it can be used as reference and overview.
- Use the setup feature inside the example launcher to add context help and calltips into SciTE.

Remark

Because helpfile does (yet) not provide detailed informations about parameters and usage of all working functions, documentation of the original Freebasic Wrapper is also included as appendix. au3Irr2 follows (mostly) its syntax and usage, so it's a good place to look into (keep in mind: not all FB Wrapper functions are also implemented in au3Irr2).
Where to continue

- jRowe's au3Irr2 topic on autoitscript.com. Don't be shy, be part of it.
- au3Irr2 project page with latest releases and sources.
- Frank Dodd's Irrlicht Wrapper Online Portal.
- Frank Dodd's Irrlicht Wrapper in the FreeBasic Forum.
- Irrlicht engine - the 'mother' of it all.
Software License

au3Irr2

WWW: http://code.google.com/p/au3irrlicht2/
Contact: via au3Ir2 topic on autoitscript.com
Authors: J.Rowe and Andreas Templin
END-USER LICENSE AGREEMENT FOR THIS SOFTWARE

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
ADDENDUM

au3Irr2 is based upon the open source projects

- 'IrrlichtWrapper for FreeBasic' from Frank Dodd and the IrmlchtWrapper for FB team (http://www.freebasic.net/forum/viewtopic.php?t=3584), and
- 'Irrlicht Engine' from Nikolaus Gebhardt and the Irrlicht team (http://irrlicht.sourceforge.net/).
  The Irrlicht Engine itself is based in part on the work of the Independent JPEG Group and the zlib.

Please refer to documentations and license agreements of this two projects for further information.

[END OF LICENSE]
This file was created automatically as a subset. See include\_au3Irr2\_changelog.txt for complete list of changes.
Release 2.04 (2010-10-09)

Script breaking changes:

- _IrrAddSplineAnimator: Parameters changed to simplify usage (see examples)
- _IrrGetNodePosition, _IrrGetNodeRotation, _IrrGetNodeAbsolutePosition, _IrrGetCameraTarget: Removed required byRef array parameter (see help file/adjusted examples for syntax)

UDFs:

- Fixed/Added: Some more working functions (mainly around added examples)
- Added: Some missing 2D functions and documentation (contributions from smashly)
- Fixed: _IrrSetFog (linear and exponential fog were interchanged)
- Fixed: _IrrCreateMesh ($s_MeshName not being passed in the DLL call)
- Fixed: _IrrGetScreenCoordinatesFrom3DPosition (did not return anything useable because of wrong dllCall)

Examples:

- Added: 015 (CustomMesh), 022 (Indices+Vertices), 024 (Mesh to file), 062 (6DOF_Camera)
- Added: (contribution from smashly): 039 (Texture blending), 070 (Texture_and_Images), 103 (Billboard Groups), 104 (LOD)
- Added: 007/025/027/066/067/068/069/074/075 (ParticleSystem)
- Reworked: 011 (Animators) shows usage of changed _IrrAddSplineAnimator
- Fixed: 010 (TerrainAndFog) / 029/054 (Skydomes)

Help file:

- Added: Completed documentation including example code for some more functions. Current status: Topics completed along examples 1-16 + several more. At least naked reference w/o detailed
informations for all functions inside other examples.

- Added: 'Copy to clipboard'-button for included examples (using VBS code from GEOSoft)

**Internal tools:**

- Fixed: Examples could not be opened when running launcher from path with spaces
- Added: Setup feature to example launcher to merge au3Irr2 help into local au3 help and add/update calltips for SciTe
- Excluded `\internal_tools` because not too interesting for 99.9% (for the 0.1%: available via the sources from project page)

**Other:**

- Changed: global $result used in UDF files switched to local variables (WIP, finished for 2D, Scene, Camera, Animation, Node)
Release 2.03 (2010-09-05)

UDF:

- Changed: Splitted UDF per category into \include. Main UDF (and the only one which is needed to be included) is still au3Irrlicht2.au3
- Changed: All used dll files are moved to \bin to clean up the root dir. _IrrStart is modified to find them anyway.
- Fixed: _IrrSetNodeVisibility

Examples:

- Added: 089 (Orthogonal Camera), 106 (Advanced start)
- Changed: 029 (Skydome)
- Fixed: 049 (Loaded_Scene_Collision), 051 (Clouds)

Help file:

- Added: First version as reference of all working + proven functions. Current status: Topics completed including example code along examples 1 to 6. Naked reference w/o detailed informations for other examples.
- Changed: Moved original freeBasic Wrapper docs from \FB_documentation to \internal_tools\buildHelp\html_static. It's now included into help file.
- Added: Merged help to use the au3Irr2 help inside the au3 help.

Internal tools:

- Added: helper scripts and files for building help file and au3.user.calltips.api (see \internal tools\help_building readme.txt)

Other:

- Fixed: \media\fonthaettenschweiler.bmp (bogQ)
- Added: msvcr71.dll - possibly missing on some machines (jl)
First release in one package (updated UDF + all needed .dll's, more examples, ExampleLauncher)
Release 2.01 (2010-07-20)

Updated UDF with fixes, additions and separate example package
Release 2.00 (2010-05-03)

First release of jRowe
Tutorials

Where are the tutorials?

There are no tutorials nor bigger demos yet. But why not be the first to write one?
Get yourself a place on the credits page and help au3Irr2 to become a mature project!

In the meantime ...

... you are not alone. There are a lot of well commented examples in the example directory. They should give you all you need to work yourself into au3Irr2.

Have fun!
_IrrSetTextureCreationFlag

Sets texture creation flags controlling how textures are handled when they are created.

#include <au3Irrlicht2.au3>
_IrrSetTextureCreationFlag($i_Flag, $i_Value)

Parameters

<table>
<thead>
<tr>
<th>$i_Value</th>
<th>The following flags can be set;</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ETCF_ALWAYS_16_BIT</td>
<td>Forces the driver to always create 16 bit textures, independently of which format the file on disk has.</td>
</tr>
<tr>
<td>$ETCF_ALWAYS_32_BIT</td>
<td>Forces the driver to always create 32 bit textures, independently of which format the file on disk has.</td>
</tr>
<tr>
<td>$ETCF_OPTIMIZED_FOR_QUALITY</td>
<td>Lets the driver decide in which format the textures are created and tries to make the textures look as good as possible.</td>
</tr>
<tr>
<td>$ETCF_OPTIMIZED_FOR_SPEED</td>
<td>Lets the driver decide in which format the textures are created and tries to create them maximizing render speed.</td>
</tr>
<tr>
<td>$ETCF_CREATE_MIP_MAPS</td>
<td>Automatically creates mip map levels for the textures.</td>
</tr>
<tr>
<td>$ETCF_NO_ALPHA_CHANNEL</td>
<td>Discard any alpha layer and use non-alpha color format.</td>
</tr>
</tbody>
</table>

| $i_Flag | Turn Creation Flag Off or On ($IRR_OFF or $IRR_ON) |

Return Value

Success: True
Failure: False
Remarks
None.

Related
None.

Example

```au3
#include <au3Irrlicht2.au3>
Global $hTexture, $aInfo

_IrrStart()

; Set the Texture creation flag to load textures in 16 bit without alpha (R5G6B5 format)
_IrrSetTextureCreationFlag( BitOR($ETCF_ALWAYS_16_BIT, $ETCF_NO_ALPHA_CHANNEL), $IRR_ON )

; Load a texture
$hTexture = _IrrGetTexture("./media/cross.bmp")

; query some info about the loaded texture, index 3 of the returned array is Color Reference
$aInfo = _IrrGetTextureInformation($hTexture)

; show the color format in a string to see the Texture creation flag was set
MsgBox(64, "Texture color format", _TextureFormatString($aInfo[3]))

_IrrStop()

Func _TextureFormatString($iValue)
```
Local $sMsg

Switch $iValue
  Case $ECF_R5G6B5
    $sMsg &= "R5G6B5 - 16 bit without alpha channel"
  Case $ECF_A1R5G5B5
    $sMsg &= "A1R5G5B5 - 16 bit with alpha channel"
  Case $ECF_R8G8B8
    $sMsg &= "R8G8B8 - 24 bit without alpha channel"
  Case $ECF_A8R8G8B8
    $sMsg &= "A8R8G8B8 - 32 bit with alpha channel"
  Case Else
    $sMsg &= "Unknown"
EndSwitch
Return $sMsg
EndFunc  ;==>_TextureFormatString
__IrrGetTexture

Loads 2D texture from bitmap file into video memory that can then be used to texture a model or to draw onto the screen.

```
#include <au3Irrlicht2.au3>
__IrrGetTexture($s_ImageFile)
```

**Parameters**

$s_ImageFile Full path to the bitmap file.

**Return Value**

Success: Handle of the device dependend irrlicht texture object
Failure: False

**Remarks**

Irrlicht engine supports currently this image file formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Support</th>
<th>Format</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG File Interchange Format (.jpg, r/w)</td>
<td></td>
<td>Portable Network Graphics (.png, r/w)</td>
<td></td>
</tr>
<tr>
<td>Truevision Targa (.tga, r/w)</td>
<td></td>
<td>Windows Bitmap (.bmp, r/w)</td>
<td></td>
</tr>
<tr>
<td>Zsoft Paintbrush (.pcx, r/w)</td>
<td></td>
<td>Portable Pixmaps (.ppm, r/w)</td>
<td></td>
</tr>
<tr>
<td>Adobe Photoshop (.psd, r)</td>
<td></td>
<td>Quake 2 textures (.wal, r)</td>
<td></td>
</tr>
</tbody>
</table>

**Related**

__IrrRemoveTexture__, __IrrGetImage__
```plaintext
#include "au3Irrlicht2.au3"

_IrrStart()

local $camera = _IrrAddCamera(2,2,2,0,0,0)

local $mesh = _IrrGetMesh( "./media/capsuleX.obj" )
local $sceneNode = _IrrAddMeshToScene( $mesh )

local $texture = _IrrGetTexture("./media/default_texture.png")
_IrrSetNodeMaterialTexture( $sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag( $sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

WHILE _IrrRunning()
    _IrrBeginScene(50, 50, 50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
```
_IrrGetImage

Loads 2D texture from bitmap file into main memory for CPU based operations.

```
#Include <au3Irrlicht2.au3>
_IrrGetImage($s_ImageFile)
```

**Parameters**

$s_ImageFile | Full path to the bitmap file.

**Return Value**

Success: Handle of the irrlicht memory texture object
Failure: False

**Remarks**

This images can not be used to texture 3D objects! Instead, they can be used to supply a heightmap to a terrain or other similar CPU based operations.

Irrlicht engine supports currently this image file formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG File Interchange Format (.jpg, r/w)</td>
<td>Portable Network Graphics (.png, r/w)</td>
</tr>
<tr>
<td>Truevision Targa (.tga, r/w)</td>
<td>Windows Bitmap (.bmp, r/w)</td>
</tr>
<tr>
<td>Zsoft Paintbrush (.pcx, r/w)</td>
<td>Portable Pixmaps (.ppm, r/w)</td>
</tr>
<tr>
<td>Adobe Photoshop (.psd, r)</td>
<td>Quake 2 textures (.wal, r)</td>
</tr>
</tbody>
</table>
Related

_IrrRemovelImage, _IrrGetTexture
_IrrCreateTexture

Creates a blank texture.

```cpp
#include <au3Irrlicht2.au3>
_IrrCreateTexture($s_TextureName, $i_XSize, $i_YSize, $i_ColorFormat)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_TextureName</td>
<td>Texture name as string.</td>
</tr>
<tr>
<td>$i_XSize</td>
<td>Width of the texture.</td>
</tr>
<tr>
<td>$i_YSize</td>
<td>Height of the texture.</td>
</tr>
<tr>
<td>$i_ColorFormat</td>
<td>The format of the texture can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>$ECF_A1R5G5B5 - 16 bit color format used by the software driver, and thus</td>
</tr>
<tr>
<td></td>
<td>preferred by all other irrlicht engine video drivers.</td>
</tr>
<tr>
<td></td>
<td>$ECF_R5G6B5 - Standard 16 bit color format.</td>
</tr>
<tr>
<td></td>
<td>$ECF_R8G8B8 - 24 bit color, no alpha channel, but 8 bit for red, green and</td>
</tr>
<tr>
<td></td>
<td>blue.</td>
</tr>
<tr>
<td></td>
<td>$ECF_A8R8G8B8 - Default 32 bit color format. 8 bits are used for every</td>
</tr>
<tr>
<td></td>
<td>component: red, green, blue and alpha.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Handle of the newly created irrlicht texture object
Failure: False and @error 1

**Remarks**

None.
Related
_\text{IrrDraw2DImage}, \text{IrrGetTextureInformation}, \text{IrrLockTexture}, \text{IrrUnlockTexture}

Example

```c
#include "au3Irrlicht2.au3"

Global $hTexture

_IrrStart()

$hTexture = _\text{IrrCreateTexture}("MyTexture", 128, 128, $\text{ECF\_A8R8G8B8})

\text{While} \_\text{IrrRunning}()

\quad _\text{IrrBeginScene}(255, 255, 0)
\quad _\text{IrrDraw2DImage}($hTexture, 0, 0)
\quad _\text{IrrEndScene}()

\text{WEnd}

_IrrStop()
```

Copy to Clipboard
Function Reference

_IrrCreateImage

Creates a blank image that does not use video memory.

#include <au3Irrlicht2.au3>
_IrrCreateImage($i_XSize, $i_YSize, $i_ColorFormat)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_XSize</td>
<td>Width of the texture.</td>
</tr>
<tr>
<td>$i_YSize</td>
<td>Width of the texture.</td>
</tr>
<tr>
<td>$i_ColorFormat</td>
<td>The format of the texture can be one of the following: $ECF_A1R5G5B5 - 16 bit color format used by the software driver, and thus preferred by all other irrlicht engine video drivers. $ECF_R5G6B5 - Standard 16 bit color format. $ECF_R8G8B8 - 24 bit color, no alpha channel, but 8 bit for red, green and blue. $ECF_A8R8G8B8 - Default 32 bit color format. 8 bits are used for every component: red, green, blue and alpha.</td>
</tr>
</tbody>
</table>

Return Value

Success: Handle of the newly created device dependend irrlicht image object
Failure: False and @error 1

Remarks

This images can not be used to texture 3D objects!
Instead, they can be used to supply a heightmap to a terrain or other similar CPU based operations.
Related

_IrrLockImage, _IrrUnlockImage
# _IrrRemoveTexture

Removes the texture from memory freeing up the space it occupied.

```plaintext
#include <au3Irrlicht2.au3>
_IrrRemoveTexture($h_Texture)
```

## Parameters

| $h_Texture | Handle of an device dependent irrlicht texture object |

## Return Value

- Success: True
- Failure: False

## Remarks

You should ensure that the texture is not in use by materials assigned to nodes.

## Related

- _IrrGetTexture
- _IrrGetImage

## Example

```plaintext
#include "au3Irrlicht2.au3"

_IrrStart()
```
local $camera = _IrrAddCamera(2,2,2, 0,0,0 )

local $mesh = _IrrGetMesh( "\media\capsuleX.obj" )
local $sceneNode = _IrrAddMeshToScene( $mesh )

local $texture = _IrrGetTexture("\media\default_texture.png")
_IrrSetNodeMaterialTexture( $sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag( $sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

local $time = TimerInit()
WHILE _IrrRunning()
   _IrrBeginScene(50, 50, 50)
   _IrrDrawScene()
   _IrrEndScene()

   if $sceneNode <> 0 AND TimerDiff($time) > 3000 then
      _IrrRemoveNode($sceneNode)
      _IrrRemoveTexture($texture) ; no longer needed
      $sceneNode = 0
   EndIf

WEND

_IrrStop()
**_IrrRemoveImage**

Removes the image from memory freeing up the space it occupied.

```
#include <au3Irrlicht2.au3>
_IrrRemoveImage($h_Image)
```

**Parameters**

$\textit{$h\_Image}$ | Handle of the irrlicht memory image object.

**Return Value**

Success: True
Failure: False

**Remarks**

You should ensure that the image is not in use by other functions.

**Related**

_\textit{IrrGetImage}, _\textit{IrrGetTexture}
Function Reference

_IrrLockTexture

Locks the texture and returns a pointer to the pixels.

```
#include <au3Irrlicht2.au3>
_IrrLockTexture($h_Texture)
```

Parameters

$h_Texture  Handle to an irrlicht texture object

Return Value

Success: Pointer to the pixels.
Failure: False and @error 1

Remarks

None.

Related

_IrrUnlockTexture

Example

```
#include "au3Irrlicht2.au3"

Global $hTexture
Global $iWidthHeight = 128
```
Global $iPixelsAmount = $iWidthHeight * $iWidthHeight
Global $pPixels, $tPixels
Global $iColor = 0xFFFF0000 ; Red

_IrrStart()

$hTexture = _IrrCreateTexture("Red", $iWidthHeight, $iWidthHeight, $ECF_A8R8G8B8)
$pPixels = _IrrLockTexture($hTexture)
$tPixels = DllStructCreate("uint[" & $iPixelsAmount & "]", $pPixels)
For $i = 1 To $iPixelsAmount
   DllStructSetData($tPixels, 1, $iColor, $i)
   $pPixels += 1
Next
_IrrUnlockTexture($hTexture)
$tPixels = 0

While _IrrRunning() And Sleep(10)
   _IrrBeginScene(255, 255, 0)
   _IrrDraw2DImage($hTexture, 0, 0)
   _IrrEndScene()
WEnd

_IrrStop()

Copy to Clipboard
_IrrUnlockTexture

Unlock the texture, presumably after it has been modified and recreate the mipmap levels.

```c
#include <au3Irrlicht2.au3>
_IrrUnlockTexture($h_Texture)
```

**Parameters**

| $h_Texture | Handle to an irrlicht texture object that has been Locked by prior call to _IrrLockOpenGLTexture or _IrrLockTexture. |

**Return Value**

Success: True
Failure: False

**Remarks**

None

**Related**

_`IrrLockOpenGLTexture`, _`IrrLockTexture`

**Example**

```c
#include "au3Irrlicht2.au3"
```
Global $hTexture
Global $iWidthHeight = 128
Global $iPixelsAmount = $iWidthHeight * $iWidthHeight
Global $pPixels, $tPixels
Global $iColor = 0xFFFF0000 ; Red

_IrrStart()

$hTexture = _IrrCreateTexture("Red", $iWidthHeight, $iWidthHeight, $ECF_A8R8G8B8)
$pPixels = _IrrLockTexture($hTexture)
$tPixels = DllStructCreate("uint[" & $iPixelsAmount & "]", $pPixels)
For $i = 1 To $iPixelsAmount
    DllStructSetData($tPixels, 1, $iColor, $i)
    $pPixels += 1
Next
_IrrUnlockTexture($hTexture)
$tPixels = 0

While _IrrRunning() And Sleep(10)
    _IrrBeginScene(255, 255, 0)
    _IrrDraw2DImage($hTexture, 0, 0)
    _IrrEndScene()
WEnd

_IrrStop()

Copy to Clipboard
_IrrLockImage

Locks an image object and returns a pointer to the pixels.

```cpp
#include <au3Irrlicht2.au3>
_ IrrLockImage($h_ Image)
```

### Parameters

- **$h_ Image** | Handle to an irrlicht image object

### Return Value

- **Success**: Pointer to the image pixels
- **Failure**: False and @error 1

### Remarks

None.

### Related

_-IrrCreatelImage, _IrrGetImage, _IrrUnlockImage_

### Example

```cpp
#include "au3Irrlicht2.au3"

Global $hSrcImage, $hDesTexture
Global $pSrcPixels, $pDesPixels
```
Global $tSrcPixels, $tDesPixels
Global $iWidthHeight = 256
Global $iPixelsAmount = $iWidthHeight * $iWidthHeight

_IrrStart()

$hSrcImage = _IrrGetImage("./media/splatter.tga")
$hDesTexture = _IrrCreateTexture("Desination", $iWidthHeight, $iWidthHeight, $ECF_A8R8G8B8)

$pSrcPixels = _IrrLockImage($hSrcImage)
$pDesPixels = _IrrLockTexture($hDesTexture)

$tSrcPixels = DllStructCreate("uint[" & $iPixelsAmount & "]", $pSrcPixels)
$tDesPixels = DllStructCreate("uint[" & $iPixelsAmount & "]", $pDesPixels)
For $i = 1 To $iPixelsAmount
    DllStructSetData($tDesPixels, 1, DllStructGetData($tSrcPixels, 1, $i), $i)
    $pDesPixels += 1
    $pSrcPixels += 1
Next

_IrrUnlockImage($hSrcImage)
_IrrUnlockTexture($hDesTexture)

$tSrcPixels = 0
$tDesPixels = 0

While _IrrRunning() And Sleep(10)
    _IrrBeginScene(255, 255, 0)
    _IrrDraw2DImage($hDesTexture, 0, 0)
    _IrrEndScene()
WEnd

_IrrStop()
_IrrUnlockImage

[todo]

#include <au3Irrlicht2.au3>
_IrrUnlockImage($h_Image)

Parameters

$h_Image  Handle to an irrlicht image object that has been Locked by prior call to _IrrLockImage

Return Value

Success: True
Failure: False and @error > 0

Remarks

None.

Related

_IrrLockImage

Example

#include "au3Irrlicht2.au3"

Global $hSrcImage, $hDesTexture
Global $pSrcPixels, $pDesPixels
Global $tSrcPixels, $tDesPixels
Global $iWidthHeight = 256
Global $iPixelsAmount = $iWidthHeight * $iWidthHeight

_IrrStart()

$hSrcImage = _IrrGetImage("./media/splatter.tga")
$hDesTexture = _IrrCreateTexture("Desination", $iWidthHeight, $iWidthHeight, $ECF_A8R8G8B8)

$pSrcPixels = _IrrLockImage($hSrcImage)
$pDesPixels = _IrrLockTexture($hDesTexture)

$tSrcPixels = DllStructCreate("uint" & $iPixelsAmount & "]", $pSrcPixels)
$tDesPixels = DllStructCreate("uint" & $iPixelsAmount & "]", $pDesPixels)
For $i = 1 To $iPixelsAmount
   DllStructSetData($tDesPixels, 1, DllStructGetData($tSrcPixels, 1, $i), $i)
   $pDesPixels += 1
   $pSrcPixels += 1
Next

_IrrUnlockImage($hSrcImage)
_IrrUnlockTexture($hDesTexture)

$tSrcPixels = 0
$tDesPixels = 0

While _IrrRunning() And Sleep(10)
   _IrrBeginScene(255, 255, 0)
   _IrrDraw2DImage($hDesTexture, 0, 0)
   _IrrEndScene()
WEnd

_IrrStop()
au3Irr2 Function Reference

_IrrCreateRenderTargetTexture

Create a texture that is suitable for the scene manager to use as a surface to which it can render its 3d object.

`#Include <au3Irrlicht2.au3>
_IrrCreateRenderTargetTexture($i_XSize, $i_YSize)`

Parameters

<table>
<thead>
<tr>
<th>$i_XSize</th>
<th>Width of the texture</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_YSize</td>
<td>Height of the texture</td>
</tr>
</tbody>
</table>

Return Value

Success: Handle to an irrlicht texture object
Failure: False and @error 1

Remarks

Each of the dimensions must be of a power of two for example 128x128 or 256x256.
This function is very important when producing texture maps for special effects for example a rendering of a model for a 2D image displayed in the HUD,
the rendering of a model for display on a 3D surface for example a video display of virtual camera, the rendering of the texture for the reflection of a mirror,
the rendering of the environment for use in a water or chrome shader.
Most cards, even old cards, will support this very important function.
Related

[todo: functionName, functionName]
_IrrMakeNormalMapTexture

Create a normal map from a gray-scale height map texture.

```plaintext
#include <au3Irrlicht2.au3>
_IrrMakeNormalMapTexture($h_Texture, $f_Amplitude)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Texture</td>
<td>Handle of an device dependend irrlicht texture object</td>
</tr>
<tr>
<td>$f_Amplitude</td>
<td></td>
</tr>
</tbody>
</table>

**Return Value**

Success: True  
Failure: False and @error 1

**Remarks**

Normal maps are used to add a high level of surface lighting detail to what are normally low resolution models. They can have a massive effect on the realism of an object, the model you create will have to be created in "tangent" space to support this.

**Related**

[todo: functionName, functionName]
_IrrBlendTextures

Blend the source texture into the destination texture to create a single texture.

#include <au3Irrlicht2.au3>
_IrrBlendTextures($h_TextureDest, $h_TextureSrc, $i_Xoffset, $i_Yoffset, $i_Operation)

Parameters

$h_TextureDest Handle to the Destination irrlicht texture object.
$h_TextureSrc Handle to the Source irrlicht texture object.
$i_Xoffset X position where the Source texture will be drawn into the Destination texture.
$i_Yoffset Y position where the Source texture will be drawn into the Destination texture.
$i_Operation Can be one of the following:
   $BLEND_SCREEN ; 0
   $BLEND_ADD ; 1
   $BLEND_SUBTRACT ; 2
   $BLEND_MULTIPLY ; 3
   $BLEND_DIVIDE ; 4

Return Value

Success: True
Failure: False and set @error, check @extended to see what the error is.
   @extended 0 then the @error is autoit failed the DllCall
   @extended 1 Incompatible texture types
   @extended 2 Unsupported texture format, must be 32bit
Remarks
Textures must be 32 bit format.

Related
_IrrGetTexture, _IrrCreateTexture, _IrrDraw2DImage

Example

```
#include "au3Irrlicht2.au3"

Global $hTextureDest, $hTextureSrc

_IrrStart()

$hTextureDest = _IrrGetTexture("./media/Diagonal.bmp")
$hTextureSrc = _IrrGetTexture("./media/cross.bmp")

_IrrBlendTextures($hTextureDest, $hTextureSrc, 0, 0, $BLEND_MULTIPLY)

While _IrrRunning() And Sleep(10)

  _IrrBeginScene(50, 50, 50)
  _IrrDraw2DImage($hTextureDest, 0, 0)
  _IrrEndScene()

WEnd

_IrrStop()
```
Function Reference

_IrrColorKeyTexture

Copies any parts of the texture that are the same as the specified color into the textures alpha channel.

```
#Include <au3Irrlicht2.au3>
_IrrColorKeyTexture($h_Texture, $i_Red, $i_Green, $i_Blue)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Texture</td>
<td>Handle of the texture object</td>
</tr>
<tr>
<td>$i_Red</td>
<td>Red value from 0 to 255</td>
</tr>
<tr>
<td>$i_Green</td>
<td>Green value from 0 to 255</td>
</tr>
<tr>
<td>$i_Blue</td>
<td>Blue value from 0 to 255</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True
Failure: False

**Remarks**

This can be used for special effects or to make these regions transparent.

**Related**

None.

**Example**
```c
#include "au3Irrlicht2.au3"

_IrrStart()

local $logo = _IrrGetTexture("./media/cross.bmp")
*IrrColorKeyTexture($logo, 255, 255, 255)

WHILE _IrrRunning()
    _IrrBeginScene(50, 0, 0)
    _IrrDraw2DImageElement( $logo, 0, 0 , 0, 0, 128, 64, $IRR_IGNORE_ALPHA)
    _IrrDraw2DImageElement( $logo, 0, 64, 0, 64, 128, 128, $IRR_USE_ALPHA)
    _IrrEndScene()
WEND

_IrrStop()
```
_IrrDraw2DImage

Draws the texture to the display at the supplied coordinates.

```
#Include <au3Irrlicht2.au3>
_IrrDraw2DImage($h_Image, $i_XPos, $i_YPos)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Image</td>
<td>Handle to an irrlicht image object</td>
</tr>
<tr>
<td>$i_XPos</td>
<td>X position on display from where drawing starts</td>
</tr>
<tr>
<td>$i_YPos</td>
<td>Y position on display from where drawing starts</td>
</tr>
</tbody>
</table>

### Return Value

- Success: True
- Failure: False

### Remarks

None.

### Related

_IrrGetTexture, _IrrDraw2DImageElement

### Example

```
#include "au3Irrlicht2.au3"
```
_IrrStart()

local $logo = _IrrGetTexture("./media/cross.bmp")

WHILE _IrrRunning()
    _IrrBeginScene(50, 0, 0)
    _IrrDraw2DImage( $logo, 0, 0 )
    _IrrEndScene()
WEND

_IrrStop()
#Include <au3Irrlicht2.au3>
_IrrDraw2DImageElement($h_Texture, $i_XPos, $i_YPos, $i_SourceTopX, $i_SourceTopY, $i_SourceBottomX, $i_SourceBottomY, $i_UseAlpha)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_\text{Image}$</td>
<td>Handle to an irrlicht image object</td>
</tr>
<tr>
<td>$i_\text{XPos}$</td>
<td>X position on display from where drawing starts</td>
</tr>
<tr>
<td>$i_\text{YPos}$</td>
<td>Y position on display from where drawing starts</td>
</tr>
<tr>
<td>$i_\text{SourceTopX}$</td>
<td>X top position of rectangle in the source texture</td>
</tr>
<tr>
<td>$i_\text{SourceTopY}$</td>
<td>Y top position of rectangle in the source texture</td>
</tr>
<tr>
<td>$i_\text{SourceBottomX}$</td>
<td>X bottom position of rectangle in the source texture</td>
</tr>
<tr>
<td>$i_\text{SourceBottomY}$</td>
<td>Y bottom position of rectangle in the source texture</td>
</tr>
<tr>
<td>$i_\text{UseAlpha}$</td>
<td>Whether or not to use the alpha channel should be one of the following values: $\text{IRR_IGNORE_ALPHA}$, $\text{IRR_USE_ALPHA}$</td>
</tr>
</tbody>
</table>

**Return Value**

- Success: True
- Failure: False

**Remarks**

Draws the texture to the display at the supplied co-ordinates, the image is
copied from the specified rectangle in the source texture, this enables you to put many images onto a single texture. This function also supports the alpha channel when drawing the image to the display and can draw the image transparently.

Related
_IrrGetTexture, _IrrColorKeyTexture, _IrrDraw2DImage, _IrrDraw2DImageElementStretch

Example

```plaintext
#include "au3Irrlicht2.au3"

_IrrStart()

local $logo = _IrrGetTexture("./media/cross.bmp")
_IrrColorKeyTexture($logo, 255, 255, 255)

WHILE _IrrRunning()
  _IrrBeginScene(50, 0, 0)
  _IrrDraw2DImageElement($logo, 0, 0, 0, 0, 0, 128, 64, $IRR_IGNORE_ALPHA)
  _IrrDraw2DImageElement($logo, 0, 64, 0, 64, 64, 128, 128, $IRR_USE_ALPHA)
  _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
_IrrDraw2DImageElementStretch

Draws specified rectangle from Source texture sizing it to fit the specified Destination rectangle.

#include <au3Irrlicht2.au3>
_IrrDraw2DImageElementStretch($h_Texture, $i_DestTopX, $i_DestTopY, $i_DestBottomX, $i_DestBottomY, $i_SourceTopX, $i_SourceTopY, $i_SourceBottomX, $i_SourceBottomY, $i_UseAlpha)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Texture</td>
<td>Handle to an irrlicht image object</td>
</tr>
<tr>
<td>$i_DestTopX</td>
<td>Top X Destination where the drawing will start.</td>
</tr>
<tr>
<td>$i_DestTopY</td>
<td>Top Y Destination where the drawing will start.</td>
</tr>
<tr>
<td>$i_DestBottomX</td>
<td>Bottom X Destination where the drawing will end.</td>
</tr>
<tr>
<td>$i_DestBottomY</td>
<td>Bottom Y Destination where the drawing will end.</td>
</tr>
<tr>
<td>$i_SourceTopX</td>
<td>X top position of rectangle in the source texture</td>
</tr>
<tr>
<td>$i_SourceTopY</td>
<td>Y top position of rectangle in the source texture</td>
</tr>
<tr>
<td>$i_SourceBottomX</td>
<td>X bottom position of rectangle in the source texture</td>
</tr>
<tr>
<td>$i_SourceBottomY</td>
<td>Y bottom position of rectangle in the source texture</td>
</tr>
<tr>
<td>$i_UseAlpha</td>
<td>Whether or not to use the alpha channel should be one of the following values:</td>
</tr>
<tr>
<td></td>
<td>$IRR_IGNORE_ALPHA</td>
</tr>
<tr>
<td></td>
<td>$IRR_USE_ALPHA</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False
Remarks
The image is copied from the specified rectangle in the source texture, this enables you to put many images onto a single texture. If the rectangles are different sizes this function will scale the images appropriately. This function also supports the alpha channel when drawing the image to the display and can draw the image transparently.

Related
_IrrGetTexture, _IrrColorKeyTexture, _IrrDraw2DImageElement

Example

```au3
#include <au3Irrlicht2.au3>

_IrrStart()

Local $hLogo = _IrrGetTexture("./media/Cross.bmp")

; Use White as Alpha color
_IrrColorKeyTexture($hLogo, 255, 255, 255)

While _IrrRunning()

    _IrrBeginScene(255, 255, 0)

    ; Draw Original texture just to see what the unchanged texture looks like.
    _IrrDraw2DImage($hLogo, 0, 0)

    ; Draw texture smaller using Alpha beside the original.
    _IrrDraw2DImageElementStretch($hLogo, 128, 0, 192, 64, 0, 0, 128, 128, $IRR_USE_ALPHA)
```
; Draw texture Larger using Alpha beside the previous smaller texture.
_IrrDraw2DImageElementStretch($hLogo, 192, 0, 448, 256, 0, 0, 128, 128, $IRR_USE_ALPHA)

; Draw beside Larger texture quarter of source texture to new larger destination not using Alpha.
_IrrDraw2DImageElementStretch($hLogo, 448, 0, 800, 352, 0, 0, 64, 64, $IRR_IGNORE_ALPHA)

_IrrEndScene()
WEnd

_IrrStop()
au3Irr2 Function Reference

_IrrGetFont

Loads a bitmap containing a bitmap font.

```c
#include <au3Irrlicht2.au3>
_IrrGetFont($s_Font)
```

Parameters

| $s_Font       | Filename of the bitmap font file |

Return Value

Success: Handle of the irrlicht font texture object
Failure: False

Remarks

None

Related

_Irr2DFontDraw

Example

```c
#include "au3Irrlicht2.au3"
_IrrStart()
local $bitmapFont = _IrrGetFont ( "/media/fonthaettenschweiler.bmp" )
```
WHILE _IrrRunning()
  _IrrBeginScene( 0,0,0 )
  _Irr2DFontDraw ( $BitmapFont, "@! Example Text with 'German Umlauts': ÄäÖöÜü !@", 120, 80, 250, 96 )
  _IrrEndScene()
WEND

_IrrStop()
_Irr2DFontDraw

Draws the text into the supplied rectangular area using the supplied font object.

```plaintext
#Include <au3Irrlicht2.au3>
_Irr2DFontDraw($h_Font, $s_Text, $i_XPos, $i_YPos, $i_BottomX, $i_BottomY)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Font</td>
<td>Handle of an irrlicht font texture object</td>
</tr>
<tr>
<td>$s_Text</td>
<td>Text string to display</td>
</tr>
<tr>
<td>$i_XPos</td>
<td>X top position of rectangle for the text</td>
</tr>
<tr>
<td>$i_YPos</td>
<td>Y top position of rectangle for the text</td>
</tr>
<tr>
<td>$i_BottomX</td>
<td>X bottom position of rectangle for the text</td>
</tr>
<tr>
<td>$i_BottomY</td>
<td>Y bottom position of rectangle for the text</td>
</tr>
</tbody>
</table>

### Return Value

Success: True
Failure: False

### Remarks

None

### Related

_IrrGetFont
Example

```au3
#include "au3Irrlicht2.au3"
_IrrStart()
local $bitmapFont = _IrrGetFont( "./media/fonthaettenschweiler.bmp" )

WHILE _IrrRunning()
    _IrrBeginScene( 0,0,0 )
    _Irr2DFontDraw ( $BitmapFont, "@! Example Text with 'German Umlauts': ÄäÖöÜü !@", 120, 80, 250, 96 )
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
_IrrSaveScreenShot

Save a screenshot out to a file.

```
#include <au3Irrlicht2.au3>
_IrrSaveScreenShot($s_Filename)
```

**Parameters**

$s_Filename file name to save the screenshot as.

**Return Value**

Success: True
Failure: False

**Remarks**

The image format is defined by the extension applied to the filename.

Irrlicht currently supports: bmp, png, tga, ppm and jpg

**Related**

_IrrGetScreenShot

**Example**

```
#include "au3Irrlicht2.au3"
```
Global $sScreenShot = @MyDocumentsDir & "\IrrScreenShot.jpg"
Global $camera, $mesh, $sceneNode, $texture

_IrrStart();

; Adding some props to the scene, so we get a screenshot of something.
$camera = _IrrAddCamera(2,2,2, 0,0,0 )
.mesh = _IrrGetMesh( "\media\capsuleX.obj" )
$sceneNode = _IrrAddMeshToScene( $mesh )
.texture = _IrrGetTexture("\media\default_texture.png")
_IrrSetNodeMaterialTexture( $sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag( $sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

; We want the props to be drawn before we take a screenshot.
_IrrBeginScene(50, 50, 50)
_IrrDrawScene()
_IrrEndScene()

; Taking a screenshot now the props have been drawn.
_IrrSaveScreenShot($sScreenShot)

; If the screenshot was successful then we open it with the users default viewer
If FileExists($sScreenShot) Then ShellExecute($sScreenShot)

_IrrStop()
_IrrGetScreenShot

Return a pointer to a texture containing a rectangular portion of a screenshot.

```
#include <au3Irrlicht2.au3>
_IrrGetScreenShot($i_XPos, $i_YPos, $i_Width, $i_Height)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_XPos</td>
<td>X position for the screenshot</td>
</tr>
<tr>
<td>$i_YPos</td>
<td>Y position for the screenshot</td>
</tr>
<tr>
<td>$i_Width</td>
<td>Width of the screenshot</td>
</tr>
<tr>
<td>$i_Height</td>
<td>Height of the screenshot</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Pointer to a texture containing a rectangular portion of a screenshot.
Failure: False and @error 1

**Remarks**

None

**Related**

_[IrrSaveScreenShot], [IrrDraw2DImage], [IrrDraw2DImageElement], [IrrDraw2DImageElementStretch]_
Example

```plaintext
#include "au3Irrlicht2.au3"

Global $hTexture, $camera, $mesh, $sceneNode, $texture

_IrrStart()

; Just adding some props to the scene, so we can get a screenshot of something
$camera = _IrrAddCamera(2,2,2,0,0,0)
$mesh = _IrrGetMesh("\media\capsuleX.obj")
$sceneNode = _IrrAddMeshToScene($mesh)
$texture = _IrrGetTexture("\media\default_texture.png")
_IrrSetNodeMaterialTexture($sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag($sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)

WHILE _IrrRunning()

    _IrrBeginScene(50,50,50)

    ; We want the props to be drawn before we take a screenshot.
    _IrrDrawScene()

    ; This will draw our screenshot scaled down to the top left of the window
    If $hTexture Then _IrrDraw2DImageElementStretch($hTexture, 0, 0, 200, 150, 0, 0, 800, 600, $IRR_IGNORE_ALPHA)

    _IrrEndScene()

    ; Taking a screenshot just once now the props have been drawn.
    If Not $hTexture Then $hTexture = _IrrGetScreenShot(0, 0, 800, 600)

WEND
```
_IrrStop()
au3Irr2 Function Reference

_IrrGetTextureInformation

Get information of a texture. The width, height, pitch and color format is returned in an array.

#include <auIrrlicht2.au3>
_IrrGetTextureInformation($h_Texture)

Parameters

$h_Texture | Handle to an irrlicht texture object

Return Value

Success: 1D Array with the information
$Array[0] = Width of the texture
$Array[1] = Height of the texture
$Array[2] = Pitch of the texture
$Array[3] = Color Reference of the texture (e.g.: $ECF_A1R5G5B5, $ECF_R5G6B5, $ECF_R8G8B8, $ECF_A8R8G8B8)
Failure: Empty Array and Sets @error to 1

Remarks

This function cannot be used for image objects. For this, use _IrrGetImageInformation instead.

Related

_IrrGetTexture, _IrrGetImageInformation
Example

```au3
#include <au3Irrlicht2.au3>

_IrrStart()

Local $hLogo = _IrrGetTexture("./media/Cross.bmp")

Local $aInfo = _IrrGetTextureInformation($hLogo)
If Not @error Then MsgBox(64, "Texture Information", "Width: " & $aInfo[0] & @LF & _ "Height: " & $aInfo[1] & @LF & _ "Pitch: " & $aInfo[2] & @LF & _ "Color Format: " & $aInfo[3])

_IrrStop()
```

Copy to Clipboard
_IrrGetImageInformation

Get information of an image. The width, height, pitch and color format is returned in an array.

```au3
#include <au3Irrlicht2.au3>
_IrrGetImageInformation($h_Image)
```

**Parameters**

$h_Texture Handle to an irrlicht image object

**Return Value**

Success: 1D Array with the information
- $Array[0] = Width of the image
- $Array[1] = Height of the image
- $Array[2] = Pitch of the image
- $Array[3] = Color Reference of the image (e.g.: $ECF_A1R5G5B5, $ECF_R5G6B5, $ECF_R8G8B8, $ECF_A8R8G8B8)
Failure: Empty Array and Sets @error to 1

**Remarks**

This function cannot be used for texture objects. For this, use _IrrGetTextureInformation instead.

**Related**

_IrrGetImage, _IrrGetTextureInformation
Example

```c
#include <au3Irrlicht2.au3>

_IrrStart()

Local $hLogo = _IrrGetImage("./media/Cross.bmp")

Local $aInfo = _IrrGetImageInformation($hLogo)
If Not @error Then MsgBox(64, "Image Information", "Width: " & $aInfo[0] & @LF & _ "Height: " & $aInfo[1] & @LF & _ "Pitch: " & $aInfo[2] & @LF & _ "Color Format: " & $aInfo[3])

_IrrStop()
```

Copy to Clipboard
_IrrSetNodeAnimationRange

Sets the range of animation that is to be played in the node.

```
#include <au3Irrlicht2.au3>
_IrrSetNodeAnimationRange($h_Node, $i_Start, $i_End)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle to a character scene node.</td>
</tr>
<tr>
<td>$i_Start</td>
<td>Start frame</td>
</tr>
<tr>
<td>$i_End</td>
<td>End frame</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False

Remarks

An animation sequence might run from 0 to 200 frames and a sequence where your character is running might only occupy a portion of this.

Related

_IrrGetMesh, _IrrAddMeshToScene, _IrrGetMeshFrameCount, _IrrGetNodeAnimationFrame, _IrrPlayNodeMD2Animation, _IrrSetNodeAnimationSpeed

Example
#include "au3Irrlicht2.au3"

Global $hMD2Mesh, $hMeshTexture, $hSceneNode

_IrrStart()

$hMD2Mesh = _IrrGetMesh("./media/zumlin.md2")
$hMeshTexture = _IrrGetTexture("./media/zumlin.pcx")
$hSceneNode = _IrrAddMeshToScene($hMD2Mesh)
_IrrSetNodeMaterialTexture($hSceneNode, $hMeshTexture, 0)
_IrrSetNodeMaterialFlag($hSceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)

_IrrSetNodeAnimationRange($hSceneNode, 200, 230)

_IrrAddCamera(50, 0, 0, 0, 0, 0)

While _IrrRunning() And Sleep(10)
   _IrrBeginScene(240, 255, 255)
   _IrrDrawScene()
   _IrrEndScene()
WEnd

_IrrStop()
_IrrPlayNodeMD2Animation

Selects the animation sequence of MD2 to be played.

```
#include <au3Irrlicht2.au3>
_IrrPlayNodeMD2Animation($h_Node, $i_Animation)
```

Parameters

<table>
<thead>
<tr>
<th>$h_Node</th>
<th>Handle to a character scene node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Animation</td>
<td>sequence should be one of the following values: $IRR_EMAT_STAND $IRR_EMAT_RUN $IRR_EMAT_ATTACK $IRR_EMAT_PAIN_A $IRR_EMAT_PAIN_B $IRR_EMAT_PAIN_C $IRR_EMAT_JUMP $IRR_EMAT_FLIP $IRR_EMAT_SALUTE $IRR_EMAT_FALLBACK $IRR_EMAT_WAVE $IRR_EMAT_POINT $IRR_EMAT_CROUCH_STAND $IRR_EMAT_CROUCH_WALK $IRR_EMAT_CROUCH_ATTACK $IRR_EMAT_CROUCH_PAIN $IRR_EMAT_CROUCH_DEATH $IRR_EMAT_DEATH_FALLBACK $IRR_EMAT_DEATH_FALLFORWARD $IRR_EMAT_DEATH_FALLBACKSLOW $IRR_EMAT_BOOM</td>
</tr>
</tbody>
</table>
Return Value
Success: True
Failure: False

Remarks
MD2 format models have specific animation sequences contained within them that can be played back with a simple call.

Related
_IrrGetMesh, _IrrAddMeshToScene, _IrrGetMeshFrameCount, _IrrGetNodeAnimationFrame, _IrrSetNodeAnimationRange, _IrrSetNodeAnimationSpeed

Example

```c
#include "au3Irrlicht2.au3"

_IrrStart()

local $MD2Mesh = _IrrGetMesh( "./media/zumlin.md2" )
local $MeshTexture = _IrrGetTexture( "./media/zumlin.pcx" )
local $SceneNode = _IrrAddMeshToScene( $MD2Mesh )
_IrrSetNodeMaterialTexture( $SceneNode, $MeshTexture, 0 )
_IrrSetNodeMaterialFlag( $SceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrPlayNodeMD2Animation( $SceneNode, $IRR_EMAT_SALUTE )

local $camera = _IrrAddCamera( 50,0,0, 0,0,0 )

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 0)
```
**au3Irr2 Function Reference**

**_IrrSetNodeAnimationSpeed**

Change the speed at which an animation is played for a node

```
#include <au3Irrlicht2.au3>
_IrrSetNodeAnimationSpeed($h_Node, $f_Speed)
```

**Parameters**

<table>
<thead>
<tr>
<th>$h_Node</th>
<th>Handle to a character scene node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_Speed</td>
<td>How many frames per second.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True
Failure: False

**Remarks**

You could use this to make a character run slowly or quickly and still keep its feet on the ground.

**Related**

_Invalid_

**Example**

```
#include "au3Irrlicht2.au3"
```
Global $hMD2Mesh, $hMeshTexture, $hSceneNode

_IrrStart()

$hMD2Mesh = _IrrGetMesh("./media/zumlin.md2")
$hMeshTexture = _IrrGetTexture("./media/zumlin.pcx")
$hSceneNode = _IrrAddMeshToScene($hMD2Mesh)
_IrrSetNodeMaterialTexture($hSceneNode, $hMeshTexture, 0)
_IrrSetNodeMaterialFlag($hSceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)

_IrrSetNodeAnimationSpeed($hSceneNode, 75)

_IrrAddCamera(50, 0, 0, 0, 0)

While _IrrRunning() And Sleep(10)
  _IrrBeginScene(240, 255, 255)
  _IrrDrawScene()
  _IrrEndScene()
WEnd

_IrrStop()
_IrrGetNodeAnimationFrame

Get the frame number that is currently being played by the node.

```
#include <au3Irrlicht2.au3>
_IrrGetNodeAnimationFrame($h_Node)
```

**Parameters**

$h\_Node$ | Handle to a character scene node.

**Return Value**

Success: Current frame number being played.
Failure: False and @error 1

**Remarks**

None.

**Related**

_IrrGetMesh, _IrrAddMeshToScene, _IrrGetMeshFrameCount, _IrrSetNodeAnimationSpeed, _IrrSetNodeAnimationRange

**Example**

```
#include "au3Irrlicht2.au3"

Global $hMD2Mesh, $hMeshTexture, $hSceneNode
```
_IrrStart()

$hMD2Mesh = _IrrGetMesh("./media/zumlin.md2")
$hMeshTexture = _IrrGetTexture("./media/zumlin.pcx")
$hSceneNode = _IrrAddMeshToScene($hMD2Mesh)
_IrrSetNodeMaterialTexture($hSceneNode, $hMeshTexture, 0)
_IrrSetNodeMaterialFlag($hSceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)

_IrrAddCamera(50, 0, 0, 0, 0)

While _IrrRunning() And Sleep(10)
    _IrrBeginScene(240, 255, 255)
    _IrrSetWindowCaption("_IrrGetNodeAnimationFrame - Playing Frame: ", & _IrrGetNodeAnimationFrame($hSceneNode))
    _IrrDrawScene()
    _IrrEndScene()
WEnd

_IrrStop()

Copy to Clipboard
_IrrSetNodeAnimationFrame

Set the current frame number being played in the animation.

```c
#include <au3Irrlicht2.au3>
_IrrSetNodeAnimationFrame($h_Node, $f_Frame)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle to a character scene node.</td>
</tr>
<tr>
<td>$f_Frame</td>
<td>Frame number to play</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True  
Failure: False

**Remarks**

None.

**Related**

_`IrrGetMesh`, `IrrAddMeshToScene`, `IrrGetMeshFrameCount`_

**Example**

```c
#include "au3Irrlicht2.au3"

Global $hMD2Mesh, $hMeshTexture, $hSceneNode
```
```
_IrrStart()

$hMD2Mesh = _IrrGetMesh("./media/zumlin.md2")
$hMeshTexture = _IrrGetTexture("./media/zumlin.pcx")
$hSceneNode = _IrrAddMeshToScene($hMD2Mesh)
_IrrSetNodeMaterialTexture($hSceneNode, $hMeshTexture, 0)
_IrrSetNodeMaterialFlag($hSceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)

_IrrSetNodeAnimationFrame($hSceneNode, 600)

_IrrAddCamera(50, 0, 0, 0, 0)

While _IrrRunning() And Sleep(10)
    _IrrBeginScene(240, 255, 255)
    _IrrDrawScene()
    _IrrEndScene()
WEnd

_IrrStop()
```
_IrrSetTransitionTime

Sets the transition time across which two poses of an animated mesh are blended.

```lua
#Include <au3Irrlicht2.au3>
_IrrSetTransitionTime($h_Node, $f_Speed)
```

**Parameters**

<table>
<thead>
<tr>
<th>$h_Node</th>
<th>Handle to a character scene node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_Speed</td>
<td>Speed of the transition</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True
Failure: False

**Remarks**

For example a character in a sitting pose can be switched into a lying down pose by blending the two frames, this will provide a more convincing smooth transition instead of a snap change in position.
_IrrAnimateJoints must be called before IrrDrawScene if blending is used.

**Related**

_ _IrrAnimateJoints, _ _IrrSetJointMode_
_IrrAnimateJoints

Animates the mesh based on the position of the joints.

```c
#include <au3Irrlicht2.au3>
_IrrAnimateJoints($h_Node)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle to a character scene node.</td>
</tr>
</tbody>
</table>

### Return Value

- **Success:** True
- **Failure:** False

### Remarks

This should be used at the end of any manual joint operations including blending and joints animated using IRR_JOINT_MODE_CONTROL and _IrrSetNodeRotation on a bone node.

### Related

- _IrrSetNodeRotation
- _IrrSetJointMode
_IrrSetJointMode

Sets the animation mode of joints in a node.

#include <au3Irrlicht2.au3>
_IrrSetJointMode($h_Node, $i_Mode)

Parameters

<table>
<thead>
<tr>
<th>$h_Node</th>
<th>Handle to a character scene node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Mode</td>
<td>can be one of the following;</td>
</tr>
<tr>
<td></td>
<td>$IRR_JOINT_MODE_NONE - no animation of the model</td>
</tr>
<tr>
<td></td>
<td>based on bones.</td>
</tr>
<tr>
<td></td>
<td>$IRR_JOINT_MODE_READ - automatic animation based</td>
</tr>
<tr>
<td></td>
<td>upon the animation defined with calls like</td>
</tr>
<tr>
<td></td>
<td>_IrrSetNodeAnimationRange.</td>
</tr>
<tr>
<td></td>
<td>IRR_JOINT_MODE_CONTROL - allow the position of the</td>
</tr>
<tr>
<td></td>
<td>bones to be set through code.</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False

Remarks

When using the control mode _IrrAnimateJoints must be called before IrrDrawScene.

Related

_IrrSetNodeAnimationRange, _IrrAnimateJoints, _IrrSetTransitionTime
_IrrAddCollisionAnimator

Animator applying collision detection and gravity to its parent node.

```
#include <au3Irrlicht2.au3>
void _IrrAddCollisionAnimator($h_IrrSelector, $h_Node, $f_RadiusX, $f_RadiusY, $f_RadiusZ, $f_GravityX, $f_GravityY, $f_GravityZ, $f_OffsetX, $f_OffsetY, $f_OffsetZ)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_IrrSelector</td>
<td>Handle of a selector object as created with _IrrGetCollision[...]. The selector represents a selection of triangles in the scene, this is usually all of the triangles in a map for instance.</td>
</tr>
<tr>
<td>$h_Node</td>
<td>Handle of a scene node to be collided against the selector.</td>
</tr>
<tr>
<td>$f_RadiusX, $f_RadiusY, $f_RadiusZ</td>
<td>Define an ellipsoid that defines the area of collision. This elliptical shape allows the collision detection to slide the object up steps and even ladders. If you make it too big you might be too large to get through a doorway but if you make it too small you may not be able to climb steps. You should play with these values and find the best ones for your scene.</td>
</tr>
<tr>
<td>$f_GravityX, $f_GravityY, $f_GravityZ</td>
<td>Specify the force that is applied to the node for each axis. For example 0.0,-9.8,0.0 defines a typical downward force. Other values could be used to simulate e.g. wind effects.</td>
</tr>
<tr>
<td>$f_OffsetX, $f_OffsetY, $f_OffsetZ</td>
<td>Offset the node by a specific distance from the center of the collision. As the center of the object and the size of your collision ellipsoid vary you can use this to adjust the position of</td>
</tr>
</tbody>
</table>
the node and to bring it into contact with the ground.

Return Value
Success: Handle of the created animator.
Failure: False

Remarks
The collision detection will stop the object penetrating through a surface in the objects it is colliding against and will also press it against the surface using gravity.

Related
_IrrRemoveAnimator, _IrrGetCollisionGroupFromMesh,
_IrrGetCollisionGroupFromComplexMesh,
_IrrGetCollisionGroupFromBox, _IrrGetCollisionGroupFromTerrain,
_IrrCreateCombinedCollisionGroup

Example

#include "au3Irrlicht2.au3"

_IrrStart()

_IrrAddZipFile( "/media/map-20kdm2.pk3", $IRR_IGNORE_CASE, $IRR_IGNORE_PATHS )
$meshBSP = _IrrGetMesh( "20kdm2.bsp" )
local $nodeBSP = _IrrAddMeshToSceneAsOcttree( $meshBSP )

local $nodeCamera = _IrrAddFPSCamera()
_IrrSetNodePosition( $nodeCamera, 1750, 149, 1369 )
local $selectorMap = _IrrGetCollisionGroupFromComplexMesh( $meshBSP, $nodeBSP )
local $animator = _IrrAddCollisionAnimator($selectorMap, $nodeCamera, _
  30.0,30.0,30.0, 0.0,-9.8,0.0, 0.0,50.0,0.0 )

WHILE _IrrRunning() 
  _IrrBeginScene(50, 50, 50)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
_IrrAddDeleteAnimator

Animator deleting its parent node after specified time (ms).

#Include <au3Irrlicht2.au3>
_İrrAddDeleteAnimator($h_Node, $i_Time)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene Node.</td>
</tr>
<tr>
<td>$i_Time</td>
<td>Living time of the animator in milliseconds.</td>
</tr>
</tbody>
</table>

Return Value

Success: Handle of the created animator.
Failure: False

Remarks

You could use this animator to delete a falling rock for example, all you would need to do is attach the delete animator, a movement animator and then forget about it.

Related

_İrrRemoveAnimator

Example

#include "au3Irrlicht2.au3"
local $nodeCamera = _IrrAddCamera(10,10,10,0,0,0)
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
_IrrGetTexture("\.\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrAddDeleteAnimator($nodeTest, 3000)

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 0)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
_IrrAddFlyCircleAnimator

Animator moving its parent node in a circular path.

```c
#include <au3Irrlicht2.au3>
_IrrAddFlyCircleAnimator($h_Node, $f_CenterX, $f_CenterY, $f_CenterZ, $f_Radius, $f_Speed)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene Node.</td>
</tr>
<tr>
<td>$f_CenterX, $f_CenterY, $f_CenterZ</td>
<td>Define center of the circular path.</td>
</tr>
<tr>
<td>$f_Radius</td>
<td>Defines the radius of the circular path.</td>
</tr>
<tr>
<td>$f_Speed</td>
<td>Defines how far the node is moved each frame.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Handle of the created animator.
Failure: False

**Remarks**

None

**Related**

_IIrrRemoveAnimator
Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeCamera = _IrrAddCamera(10,10,10, 0,0,0)
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrAddFlyCircleAnimator($nodeTest, 0, 0, 5, 0.005)

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 0)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
_IrrAddFlyStraightAnimator

Animator moving its parent node in a straight line from a start to an end point.

```c
#include <au3Irrlicht2.au3>
_IrrAddFlyStraightAnimator($h_Node, $f_StartX, $f_StartY, $f_StartZ, 
$f_EndX, $f_EndY, $f_EndZ, $i_Time, $i_DoLoop)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene Node.</td>
</tr>
<tr>
<td>$f_StartX,</td>
<td>Specify the start point of the path.</td>
</tr>
<tr>
<td>$f_StartY,</td>
<td></td>
</tr>
<tr>
<td>$f_StartZ</td>
<td></td>
</tr>
<tr>
<td>$f_EndX,</td>
<td>Specify the end point of the path.</td>
</tr>
<tr>
<td>$f_EndY,</td>
<td></td>
</tr>
<tr>
<td>$f_EndZ</td>
<td></td>
</tr>
<tr>
<td>$i_Time</td>
<td>Number of milliseconds the animator will take to move the node from start to end point.</td>
</tr>
<tr>
<td>$i_DoLoop</td>
<td>Determines single or continuously movement type:</td>
</tr>
<tr>
<td></td>
<td>$IRR_ONE_SHOT - For a single animation and then stop.</td>
</tr>
<tr>
<td></td>
<td>$IRR_LOOP - To continuously repeat the animation.</td>
</tr>
</tbody>
</table>

### Return Value

Success: Handle of the created animator.
Failure: False

### Remarks

None
#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeCamera = _IrrAddCamera(10,10,10, 0,0,0)
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
  _IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrAddFlyStraightAnimator($nodeTest, -25, -10, -10, 25, 10, 2, 5000, $IRR_LOOP)

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 0)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
_IrrAddRotationAnimator

Animator spinning its parent node around each axis.

```au3
#include <au3Irrlicht2.au3>
_IrrAddRotationAnimator($h_Node, $f_RotX, $f_RotY, $f_RotZ)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene Node.</td>
</tr>
<tr>
<td>$f_RotX, $f_RotY, $f_RotZ</td>
<td>Specify number of radians the object is spun around each axis.</td>
</tr>
</tbody>
</table>

### Return Value

- **Success:** Handle of the created animator.
- **Failure:** False

### Remarks

None

### Related

_`IrrRemoveAnimator`

### Example

```au3
#include "au3Irrlicht2.au3"
```
local $nodeCamera = _IrrAddCamera(10,10,10, 0,0,0 )
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrAddRotationAnimator($nodeTest, 0.1, 0.2, 0.3)

WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 0)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
_IrrAddSplineAnimator

Animator moving its parent node along a spline curve.

```
#include <au3Irrlicht2.au3>
_IrrAddSplineAnimator($h_Node, $tVectorStruct, $i_Start, $f_Speed, $f_Tightness)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene Node.</td>
</tr>
<tr>
<td>$tVectorStruct</td>
<td>Struct array with 3D-points defining the spline curve. Use __CreateVectStruct and __SetVectStruct to build the required struct.</td>
</tr>
<tr>
<td>$i_Start</td>
<td>Time in milliseconds that must pass before the animation starts.</td>
</tr>
<tr>
<td>$f_Speed</td>
<td>Defines the rate the node moves along the spline curve.</td>
</tr>
<tr>
<td>$f_Tightness</td>
<td>Specifies how tightly the curve is tied to the points. Value between 0 (angular) and 1 (very loose).</td>
</tr>
</tbody>
</table>

### Return Value

Success: Handle of the created animator.
Failure: False and sets @error:
1 - error from .dll call
2 - $tVectorStruct is not a dllstruct

### Remarks

This is one of the more difficult to set up of the animators but is very natural looking and powerful.
A spline is a curved line that passes through or close to a list of co-
ordinates, creating a smooth flight. This animator needs a list of coordinates stored in a struct array for the X, Y and Z locations of all the points. A good way to get coordinates for this struct is to load in the camera position example and move your camera to a point and write down its coordinates.

Related
__CreateVectStruct, __SetVectStruct, _IrrRemoveAnimator

Example

```
#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeCamera = _IrrAddCamera(150,50,0,0,75,0)
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

local $tVectors = __CreateVectStruct(4)
__SetVectStruct($tVectors, 0, -100, 50, 0)
__SetVectStruct($tVectors, 1, 0, 100, -100)
__SetVectStruct($tVectors, 2, 100, 50, 0)
__SetVectStruct($tVectors, 3, 0, 100, 100)

_IrrAddSplineAnimator($nodeTest, $tVectors, 0, 0.5, 1)

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 0)
    _IrrDrawScene()
    _IrrEndScene()
```
WEND

_IrrStop()
_IrrAddFadeAnimator

This animator deletes the node it is attached to after the specified number of milliseconds.

```
#include <au3Irrlicht2.au3>
_IrrAddFadeAnimator($h_Node, $i_DeleteAfterMilliseconds, $f_Scale = 1.0)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene Node.</td>
</tr>
<tr>
<td>$i_DeleteAfterMilliseconds</td>
<td>Number of milliseconds before deleting the node.</td>
</tr>
<tr>
<td>$f_Scale</td>
<td>Scaled amount while fading.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Handle of the created animator.
Failure: False

**Remarks**

During the time while it is waiting to delete it the node is slowly faded to invisibility and is also scaled by the specified amount. You could use this animator to fade and delete an object from a scene that was no longer required like a used medical pack, all you would need to do is attach the fade animator and forget about it.

**Related**
Example

```au3
#include "au3Irrlicht2.au3"

Global $hMD2Mesh, $hMeshTexture, $hSceneNode

_IrrStart()

$hMD2Mesh = _IrrGetMesh("./media/zumlin.md2")
$hMeshTexture = _IrrGetTexture("./media/zumlin.pcx")
$hSceneNode = _IrrAddMeshToScene($hMD2Mesh)
_IrrSetNodeMaterialTexture($hSceneNode, $hMeshTexture, 0)
_IrrSetNodeMaterialFlag($hSceneNode, $IRR_EMF_LIGHTING, $IRR_ON)
_IrrSetAmbientLight( 1,1,1 )

_IrrAddFadeAnimator($hSceneNode, 3000, -1)

_IrrAddCamera(50, 0, 0, 0, 0, 0)

While _IrrRunning() And Sleep(10)
  _IrrBeginScene(240, 255, 255)
  _IrrDrawScene()
  _IrrEndScene()
WEnd

_IrrStop()
```

Copy to Clipboard
_IrrRemoveAnimator

Removes an applied animator from a node.

```
#include <au3Irrlicht2.au3>
_IrrRemoveAnimator($h_Node, $h_Animator)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene Node.</td>
</tr>
<tr>
<td>$h_Animator</td>
<td>Handle of the animator to be removed.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True  
Failure: False

**Remarks**

After the function call a played animation is stopped or the animator cleaned up so you can apply a new one.

**Related**

_IrrAddCollisionAnimator,  _IrrAddDeleteAnimator,  
_IrrAddFlyCircleAnimator,  _IrrAddFlyStraightAnimator,  
_IrrAddRotationAnimator,  _IrrAddSplineAnimator,  _IrrAddFadeAnimator

**Example**

```au3```
```c
#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeCamera = _IrrAddCamera(10,10,10, 0,0,0 )
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
_IrrGetTexture("./media/au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

local $animator = _IrrAddFlyCircleAnimator($nodeTest, 0, 0, 5, 0.005)

local $timer = TimerInit()

WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 0)
   _IrrDrawScene()
   _IrrEndScene()

   if TimerDiff($timer) > 3000 then _IrrRemoveAnimator($nodeTest, $animator)
WEND

_IrrStop()
```

_IrrAddFPSCamera

Adds a ‘first person shooter’ style camera with mouse and keyboard control into the scene.

```au3
#include <au3Irrlicht2.au3>
_IrrAddFPSCamera($h_ParentNode = 0, $f_RotateSpeed = 100.0, $f_MoveSpeed = 0.5, $i_ID = -1, $h_KeyMapArray = 0, $i_KeyMapSize = 0, $i_NoVerticalMovement = 0, $f_JumpSpeed = 0.0)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_RotateSpeed</td>
<td>[optional] Speed in degrees with which the camera is rotated. This can be done only with the mouse.</td>
</tr>
<tr>
<td>$f_MoveSpeed</td>
<td>[optional] Speed in units per millisecond with which the camera is moved. Movement is done with the cursor keys.</td>
</tr>
<tr>
<td>$i_ID</td>
<td>[optional] id of the camera. This id can be used to identify the camera.</td>
</tr>
<tr>
<td>$h_KeyMapArray</td>
<td>[optional] Adress of a key map as created with <code>__CreatePtrKeyMapArray</code>, specifying what keys should be used to move the camera. If this is null, the default keymap is used. You can define actions more then one time in the array, to bind multiple keys to the same action.</td>
</tr>
<tr>
<td>$i_KeyMapSize</td>
<td>[optional] Amount of items in the keymap array.</td>
</tr>
</tbody>
</table>
### $b_{\text{NoVerticalMovement}}$ [optional]

Setting this to true makes the camera only move within a horizontal plane, and disables vertical movement as known from most ego shooters. Default is 'false', with which it is possible to fly around in space, if no gravity is there.

### $f_{\text{JumpSpeed}}$ [optional]

Speed with which the camera is moved when jumping.

#### Return Value

Success: Handle of the camera object
Failure: False

#### Remarks

Adds a camera scene node with an animator which provides mouse and keyboard control appropriate for first person shooters (FPS). If however you capture events when starting irrlicht this will become a normal camera that can only be moved by code.

This FPS camera is intended to provide a demonstration of a camera that behaves like a typical First Person Shooter.

It is useful for simple demos and prototyping but is not intended to provide a full solution for a production quality game.

It binds the camera scene node rotation to the look-at target.

#### Related

__CreatePtrKeyMapArray__, __IrrAddCamera__, __IrrAddMayaCamera__

#### Example

```
#include "au3Irrlicht2.au3"
```
_IrrStart()

local $camera = _IrrAddFPSCamera()

local $mesh = _IrrGetMesh( "\media\capsuleX.obj" )
local $sceneNode = _IrrAddMeshToScene( $mesh )
_IrrSetNodePosition($sceneNode, 0, 0, 5)

local $texture = _IrrGetTexture("\media\default_texture.png")
_IrrSetNodeMaterialTexture( $sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag( $sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrSetWindowCaption("Move with mouse + cursor keys - quit with ALT-F4!")
WHILE _IrrRunning() DO
    _IrrBeginScene(50, 50, 50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
_IrrAddCamera

Adds a camera into the scene.

```
#include <au3Irrlicht2.au3>
_IrrAddCamera($f_CamX, $f_CamY, $f_CamZ, $f_TargetX, $f_TargetY, $f_TargetZ)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_CamX</td>
<td>X value for view point of the camera.</td>
</tr>
<tr>
<td>$f_CamY</td>
<td>Y value for view point of the camera.</td>
</tr>
<tr>
<td>$f_CamZ</td>
<td>Z value for view point of the camera.</td>
</tr>
<tr>
<td>$f_TargetX</td>
<td>X value for target of camera.</td>
</tr>
<tr>
<td>$f_TargetY</td>
<td>Y value for target of camera.</td>
</tr>
<tr>
<td>$f_TargetZ</td>
<td>Z value for target of camera.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Handle of the camera object
Failure: False

**Remarks**

The camera will be used to define the view point and target point and other attributes of the view into the 3D scene. Animators and other node functions can be applied to this node.

**Related**

_ IrrAddFPSCamera, _IrrAddMayaCamera
Example

```c
#include "au3Irrlicht2.au3"

_IrrStart()

local $camera = _IrrAddCamera(2,2,2,0,0,0)

local $mesh = _IrrGetMesh("./media\capsuleX.obj")
local $sceneNode = _IrrAddMeshToScene($mesh)

local $texture = _IrrGetTexture("./media\default_texture.png")
_IrrSetNodeMaterialTexture($sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag($sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)

WHILE _IrrRunning()
    _IrrBeginScene(50,50,50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
**au3Irr2 Function Reference**

**_IrrAddMayaCamera**

Adds a Maya style camera to into the scene.

```
#include <au3Irrlicht2.au3>
_IrrAddMayaCamera($h_Node = $IRR_NO_OBJECT, $f_Rotate = 100.0, $f_Zoom = 100.0, $f_Move = 100.0)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle parent irr node, if no parent is needed then use $IRR_NO_OBJECT.</td>
</tr>
<tr>
<td>$f_Rotate</td>
<td>Speed at which the camera revolves.</td>
</tr>
<tr>
<td>$f_Zoom</td>
<td>Speed at which the camera zooms in and out.</td>
</tr>
<tr>
<td>$f_Move</td>
<td>Speed at which the camera moves.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Handle of the camera object
Failure: False

**Remarks**

The user can click with the left, middle and right mouse buttons to move, zoom and rotate the camera.

**Related**

_**_IrrAddCamera, _IrrAddFPSCamera
Example

#include "au3Irrlicht2.au3"

Global $hMD2Mesh, $hMeshTexture, $hSceneNode, $hMayaCamera

_IrrStart()

$hMD2Mesh = _IrrGetMesh("./media/zumlin.md2")
$hMeshTexture = _IrrGetTexture("./media/zumlin.pcx")
$hSceneNode = _IrrAddMeshToScene($hMD2Mesh)
_IrrSetNodeMaterialTexture($hSceneNode, $hMeshTexture, 0)
_IrrSetNodeMaterialFlag($hSceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)

$hMayaCamera = _IrrAddMayaCamera($hSceneNode)
_IrrSetCameraTarget($hMayaCamera, 0, 0, 0)

While _IrrRunning() And Sleep(10)
    _IrrBeginScene(240, 255, 255)
    _IrrDrawScene()
    _IrrEndScene()
WEnd

_IrrStop()
_IrrSetCameraTarget

Sets the point in space that the camera is looking at.

```c
#include <au3Irrlicht2.au3>
_IrrSetCameraTarget($h_Camera, $f_CamX, $f_CamY, $f_CamZ)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Camera</td>
<td>Handle of a camera object</td>
</tr>
<tr>
<td>$f_CamX, $f_CamY, $f_CamZ</td>
<td>Position in the scene to target with the camera.</td>
</tr>
</tbody>
</table>

### Return Value

Success: True  
Failure: False

### Remarks

The camera view point can be moved by simply using the `_IrrSetNodePosition` function but this operation will change the point that the camera is pointing at.

### Related

- `_IrrGetCameraTarget`  
- `_IrrSetCameraUpAtRightAngle`  
- `_IrrSetCameraUpDirection`  
- `_IrrGetCameraOrientation`  
- `_IrrGetCameraUpDirection`
#include "au3Irrlicht2.au3"

_IrrStart()

; set up a sphere into the scene:
local $nodeSphere = _IrrAddSphereSceneNode(5)
_IrrSetNodeMaterialTexture( $nodeSphere, _IrrGetTexture(".\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeSphere, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodePosition($nodeSphere, 10, 5, 10)

local $nodeCamera = _IrrAddFPSCamera()

; target the sphere with the camera:
_IrrSetCameraTarget($nodeCamera, 10, 5, 10)

WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 25)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
_IrrGetCameraTarget

Returns array with coordinates of point in space that the camera is looking at.

```
#include <au3Irrlicht2.au3>
_IrrGetCameraTarget($h_Camera)
```

Parameters

$h_Camera | Handle of a camera object

Return Value

success: Array with three elements for X, Y, Z.
failure: False

Remarks

None.

Related

_IrrSetCameraTarget, _IrrSetCameraUpAtRightAngle, _IrrSetCameraUpDirection, _IrrGetCameraOrientation, _IrrGetCameraUpDirection

Example

```
#include "au3Irrlicht2.au3"
```
_IrrStart()

_IrrAddTestSceneNode()
local $nodeCamera = _IrrAddFPSCamera()
_IrrSetNodePosition($nodeCamera, 0, 0, -50)

local $aTarget

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 50)

    $aTarget = _IrrGetCameraTarget($nodeCamera)
    _IrrSetWindowCaption('Targeting: X: ' & int($aTarget[0]) & ' / Y: ' & int($aTarget[1]) & ' / Z: ' & int($aTarget[2]) )

    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
_IrrGetCameraOrientation

Gets the camera orientation (forward, upward and sideways vectors of the camera)

```
#include <au3Irrlicht2.au3>
_IrrGetCameraOrientation($h_Camera, ByRef $a_Vector1, ByRef $a_Vector2, ByRef $a_Vector3)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Camera</td>
<td>Handle of a camera object</td>
</tr>
<tr>
<td>$a_Vector1,</td>
<td>Any variables to populate with the camera orientation vectors, must not</td>
</tr>
<tr>
<td>$a_Vector2,</td>
<td>explicitly be arrays.</td>
</tr>
<tr>
<td>$a_Vector3</td>
<td></td>
</tr>
</tbody>
</table>

### Return Value

- **Success:** True
- **Failure:** False

### Remarks

Returned vectors will be different lengths depending on how much the camera is rotated. The described camera direction is useful after the camera has been revolved.

### Related

- _IrrGetCameraTarget
- _IrrSetCameraTarget
- _IrrSetCameraUpAtRightAngle
- _IrrSetCameraUpDirection
- _IrrGetCameraUpDirection
Example

```c
#include "au3Irrlicht2.au3"

; better example welcome :)

_IrrStart()

_IrrAddTestSceneNode()
local $nodeCamera = _IrrAddFPSCamera()
_IrrSetNodePosition($nodeCamera, 0, 0, -50)

local $aVectForward, $aVectUpward, $aVectSide

WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 50)

   _IrrGetCameraOrientation($nodeCamera, $aVectForward, $aVectUpward, $aVectSide)
   _IrrSetWindowCaption('Upward vector: ' & _

   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
_IrrRevolveCamera

Revolve the camera using quaternion calculations.

```au3
#include <au3Irrlicht2.au3>
_IrrRevolveCamera($h_Camera, $f_Yaw, $f_Pitch, $f_Roll, $f_Drive, $f_Strafe, $f_Elevate)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Camera</td>
<td>Handle of a camera node.</td>
</tr>
<tr>
<td>$f_Yaw</td>
<td>Affects turning left and right.</td>
</tr>
<tr>
<td>$f_Pitch</td>
<td>Affects tilting up and down.</td>
</tr>
<tr>
<td>$f_Roll</td>
<td>Affects rolling left and right.</td>
</tr>
<tr>
<td>$f_Drive</td>
<td>Affects moving forwards and backward.</td>
</tr>
<tr>
<td>$f_Strafe</td>
<td>Affects moving left and right.</td>
</tr>
<tr>
<td>$f_Elevate</td>
<td>Affects moving up and down.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True
Failure: False

**Remarks**

This will help avoid gimbal lock associated with normal Rotations and is ideal for spacecraft and aircraft.

**Related**
[todo: functionName, functionName]
_IrrSetCameraUpAtRightAngle

Set the camera up at a right angle to the camera vector.

#include <au3Irrlicht2.au3>
_IrrSetCameraUpAtRightAngle($h_Camera)

Parameters

$h_Camera  Handle of a camera node.

Return Value

Success: True
Failure: False

Remarks

todo

Related

todo: functionName, functionName
_IrrSetCameraOrthogonal

Set the projection of the camera to an orthogonal view, where there is no sense of perspective.

```
#include <au3Irrlicht2.au3>
_IrrSetCameraOrthogonal($h_Camera, $f_DistanceX, $f_DistanceY, $f_DistanceZ)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Camera</td>
<td>Handle of a camera node.</td>
</tr>
<tr>
<td>$f_DistanceX</td>
<td>X Distance</td>
</tr>
<tr>
<td>$f_DistanceY</td>
<td>Y Distance</td>
</tr>
<tr>
<td>$f_DistanceZ</td>
<td>Z Distance</td>
</tr>
</tbody>
</table>

### Return Value

Success: True  
Failure: False

### Remarks

The distance to the target adjusts the width and height of the camera view, essentially the smaller it is the larger the object will appear.

### Related

[todo: functionName, functionName]
_IrrSetCameraClipDistance

Defines far and near distances for camera clipping

#include <au3Irrlicht2.au3>
_IrrSetCameraClipDistance($h_Camera, $f_Distance, $f_NearDistance = 1.0)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Camera</td>
<td>Handle of a camera node</td>
</tr>
<tr>
<td>$f_Distance</td>
<td>Defines the far distance for clipping</td>
</tr>
<tr>
<td>$f_NearDistance</td>
<td>[optional] Defines the near distance for clipping (towards the camera)</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False

Remarks

The clipping distances of a camera are the distances beyond and before which no triangles are rendered. Default clipping is before 1.0 and behind 2000.0.
This speeds the scene up by not showing geometry that is beyond or before the defined distances and increases rendering performance without requiring you to manage adding and deleting the objects from the view.
To make the far distance clipping less abrupt you can use it in combination with _IrrSetFog.
Related

_IrrAddCamera, _IrrAddFPSCamera, _IrrSetFog

Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeCube = _IrrAddCubeSceneNode(12)
_IrrSetNodeMaterialTexture( $nodeCube,
   _IrrGetTexture("./media/au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_LIGHTING, $IRR_OFF )

local $nodeCamera = _IrrAddCamera(10, 10, 10, 0, 0, 0)
_IrrSetCameraClipDistance($nodeCamera, 15)

WHILE _IrrRunning()
  _IrrBeginScene(50, 50, 50)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
_IrrSetActiveCamera

When you have several camera objects in the scene you can use this call to define which of them is to be used to look through when drawing the scene.

```
#include <au3Irrlicht2.au3>
_IrrSetActiveCamera($h_Camera)
```

**Parameters**

$h_Camera | Handle of a camera node.

**Return Value**

Success: True
Failure: False

**Remarks**

None.

**Related**

[/todo: functionName, functionName]
_IrrSetCameraFOV

Sets the field of vision of the camera.

#include <au3Irrlicht2.au3>
_IrrSetCameraFOV($h_Camera, $f_FOV)

Parameters

$\texttt{h}$.\texttt{Camera}  | Handle of a camera node.
$\texttt{f}$.\texttt{FOV}    | The value is in radians and has a default value of $\pi / 2.5$

Return Value

Success: True
Failure: False

Remarks

A wide field of vision will give a distorted perspective, if the angle is too narrow the display will feel restricted.

Related

[todo: functionName, functionName]
_IrrSetCameraAspectRatio

Sets the aspect ratio of the camera in the same way you think of standard screens and widescreens.

```
#include <au3Irrlicht2.au3>
_IrrSetCameraAspectRatio($h_Camera, $f_AspectRatio)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Camera</td>
<td>Handle of a camera node.</td>
</tr>
<tr>
<td>$f_AspectRatio</td>
<td>Aspect ratio as a float value.</td>
</tr>
</tbody>
</table>

### Return Value

- **Success:** True
- **Failure:** False

### Remarks

A widescreen usually has an aspect ratio of 16:9 or 16/9 = 1.78. The camera aspect ratio is set up automatically. However if you are using split screen effects you may need to change the camera aspect ratio.

### Related

[todo: functionName, functionName]
__CreatePtrKeyMapArray

Helper function: returns pointer to a keymap for _IrrAddFPSCamera.

```c
#include <au3Irrlicht2.au3>
__CreatePtrKeyMapArray(ByRef $keyStruct, $i_kForward = $KEY_KEY_W, $i_kBackward = $KEY_KEY_S, $i_kLeft = $KEY_KEY_A, $i_kRight = $KEY_KEY_D, $i_kJump = $KEY_SPACE)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$keyStruct</td>
<td>Any variable which is then returned as a keymap for _IrrAddFPSCamera.</td>
</tr>
<tr>
<td>$i_kForward</td>
<td>Key for forward movement, default is $KEY_KEY_W</td>
</tr>
<tr>
<td>$i_kBackward</td>
<td>Key for backward movement, default is $KEY_KEY_S</td>
</tr>
<tr>
<td>$i_kLeft</td>
<td>Key for left movement, default is $KEY_KEY_A</td>
</tr>
<tr>
<td>$i_kRight</td>
<td>Key for right movement, default is $KEY_KEY_D</td>
</tr>
<tr>
<td>$i_kJump</td>
<td>Key for jumping, default is $KEY_SPACE</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Pointer to Dll struct containing key struct useable by _IrrAddFPSCamera.
Failure: False

**Remarks**

$keyStruct$ can be set to '0' after call of _IrrAddFPSCamera to delete the Dll struct. Usage with defaults creates WASD keys, e.g.:

```c
_IrrAddFPSCamera (..., __CreatePtrKeyMapArray($keyStruct), 4, ...)
```
Related

_IrraddFPSCamera

Example

```plaintext
#include "au3Irrlicht2.au3"

_IrrStart()

local $keyStruct
local $camera = _IrrAddFPSCamera( 0, 150.0, 0.1, -1, _
                      __CreatePtrKeyMapArray($keyStruct), 4 )
$keyStruct = 0

local $mesh = _IrrGetMesh( ".\media\capsuleX.obj" )
local $sceneNode = _IrrAddMeshToScene( $mesh )
_IrrSetNodePosition($sceneNode, 0, 0, 5 )

local $texture = _IrrGetTexture(".\media\default_texture.png")
_IrrSetNodeMaterialTexture( $sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag( $sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrSetWindowCaption("Move with mouse + WASD keys - quit with ALT-F4!")
WHILE _IrrRunning()
    _IrrBeginScene(50, 50, 50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()```
_IrrGetCollisionGroupFromMesh

(todo)

#include <au3Irrlicht2.au3>
_IrrGetCollisionGroupFromMesh($h_Mesh, $h_Node, $i_Frame = 0)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

todo

Related

_IrrGetCollisionGroupFromComplexMesh,
_IrrGetCollisionGroupFromBox, _IrrGetCollisionGroupFromTerrain,
_IrrRemoveCollisionGroup, _IrrCreateCombinedCollisionGroup
_IrrGetCollisionGroupFromComplexMesh

Creates an optimized triangle selection group from a large complex mesh like a map.

#include <au3Irrlicht2.au3>
_IrrGetCollisionGroupFromComplexMesh($h_Mesh, $h_Node, $i_Frame = 0)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Mesh</td>
<td>Handle of mesh the node was created from.</td>
</tr>
<tr>
<td>$h_Node</td>
<td>Handle of the node to create a selector from.</td>
</tr>
<tr>
<td>$i_Frame</td>
<td>[optional] Number of mesh frame to use.</td>
</tr>
</tbody>
</table>

Return Value

success: Handle to a selector object
failure: False

Remarks

The returned triangle selection group can then be used in collision functions to collide objects against this node.

Related

_IrrGetCollisionGroupFromMesh, _IrrGetCollisionGroupFromBox, _IrrGetCollisionGroupFromTerrain, _IrrRemoveCollisionGroup, _IrrCreateCombinedCollisionGroup
Example

```plaintext
#include "au3Irrlicht2.au3"

_IrrStart()

巡视搜查文件("./media/map-20kdm2.pk3", $IRR_IGNORE_CASE, $IRR_IGNORE_PATHS)
$meshBSP = _IrrGetMesh("20kdm2.bsp")
local $nodeBSP = _IrrAddMeshToSceneAsOcttree($meshBSP)

local $nodeCamera = _IrrAddFPSCamera()
_IrrSetNodePosition($nodeCamera, 1750, 149, 1369)

local $selectorMap = _IrrGetCollisionGroupFromComplexMesh($meshBSP, $nodeBSP)
local $animator = _IrrAddCollisionAnimator($selectorMap, $nodeCamera, _30.0, 30.0, 30.0, 0.0, -9.8, 0.0, 0.0, 50.0, 0.0)

WHILE _IrrRunning()
    _IrrBeginScene(50, 50, 50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
au3Irr2 Function Reference

_IrrGetCollisionGroupFromBox

[todo]

#include <au3Irrlicht2.au3>
_IrrGetCollisionGroupFromBox($h_Node)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

_IrrGetCollisionGroupFromMesh,
_IrrGetCollisionGroupFromComplexMesh,
_IrrGetCollisionGroupFromTerrain, _IrrRemoveCollisionGroup,
_IrrCreateCombinedCollisionGroup
# _IrrCreateCombinedCollisionGroup

[todo]

```c
#include <au3Irrlicht2.au3>
_IrrCreateCombinedCollisionGroup()
```

## Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[moreTextForParam1]</td>
<td></td>
</tr>
</tbody>
</table>

## Return Value

- **[success]**: [explanation]
- **[failure]**: [explanation]
  - [moreExplanationIndented]

## Remarks

[todo]

## Related

- _IrrGetCollisionGroupFromMesh,
- _IrrGetCollisionGroupFromComplexMesh,
- _IrrGetCollisionGroupFromBox,
- _IrrGetCollisionGroupFromTerrain,
- _IrrRemoveCollisionGroup
# IrrAddCollisionGroupToCombination

(todo)

```c
#include <au3Irrlicht2.au3>
_IrrAddCollisionGroupToCombination($h_CombinedCollisionGroup,
$h_CollisionGroup)
```

## Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

## Return Value

[success]: explanation

[failure]: explanation

  [moreExplanationIndented]

## Remarks

(todo)

## Related

[todo: functionName, functionName]
__IrrGetCollisionPoint

[todo]

```
#include <au3Irrlicht2.au3>
__IrrGetCollisionPoint($a_StartVector, $a_EndVector, $h_CollisionGroup, 
byRef $a_CollisionVector)
```

### Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

### Return Value

[success]: [explanation]

[failure]: [explanation]

  [moreExplanationIndented]

### Remarks

[todo]

### Related

[todo: functionName, functionName]
_IrrGetCollisionNodeFromCamera

[todo]

#include <au3Irrlicht2.au3>
_IrrGetCollisionNodeFromCamera($h_Camera)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]

[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetCollisionNodeFromRay

(todo)

#include <au3Irrlicht2.au3>
_IrrGetCollisionNodeFromRay(byRef $h_StartVector, byRef $h_EndVector)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[succeed]: [explanation]
[fail]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetCollisionNodeFromScreenCoordinates

[todo]

#include <au3Irrlicht2.au3>
_IrrGetCollisionNodeFromScreenCoordinates($i_X, $i_Y)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetScreenCoordinatesFrom3DPosition

Screen co-ordinates are returned for the position of the specified 3D co-ordinates.

```
#include <au3Irrlicht2.au3>
_IrrGetScreenCoordinatesFrom3DPosition(ByRef $i_ScreenX, ByRef $i_ScreenY, $a_3DPositionVector)
```

**Parameters**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_ScreenX, $i_ScreenY</td>
<td>Variables which will contain coordinates after call of the function.</td>
</tr>
<tr>
<td>$a_3DPositionVector</td>
<td>1D array with three elements for x, y, z values of a position in space.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True and sets passed $i_ScreenX and $i_ScreenY
Failure: False and @error = 1

**Remarks**

Screen co-ordinates are returned for the position of the specified 3D co-ordinates as if an object were drawn at them on the screen, this is ideal for drawing 2D bitmaps or text around or on your 3D object on the screen for example in the HUD of an aircraft.

**Related**

_IrrGet3DPositionFromScreenCoordinates,  
_IrrGet2DPositionFromScreenCoordinates
Example

```
#include "au3Irrlicht2.au3"

_IrrStart()

local $x, $y, $vect[3]
local $BitmapFont = _IrrGetFont( "./media/bitmapfont.bmp" )
local $SceneNode = _IrrAddMeshToScene( _IrrGetMesh( "/media/zumlin.md2" ) )
_IrrSetNodeMaterialTexture( $SceneNode, _IrrGetTexture( "/media/zumlin.pcx" ), 0 )
_IrrSetNodeMaterialFlag( $SceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

local $nodeCam = _IrrAddFPSCamera( $IRR_NO_OBJECT, 100.0, 0.1 )
_IrrSetNodePosition( $nodeCam, 80,0,0 )
_IrrSetCameraTarget( $nodeCam, 0,0,0 )

$vect[1] = 35 ; move the vector to just over the nodes head

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 25)
    _IrrDrawScene()

    ; using 3D coordinate get x/y position of this coordinate; it appears on the screen
    _IrrGetScreenCoordinatesFrom3DPosition( $x, $y, $vect )
    ; draw the name of the model over the head of the model
    _Irr2DFontDraw( $BitmapFont, "ZUMLIN", $x-15, $y-8, $x+35, $y )

_IrrEndScene()
WEND
```
_IrrGet3DPositionFromScreenCoordinates

[todo]

```
#include <au3Irrlicht2.au3>
_IrrGet3DPositionFromScreenCoordinates($i_X, $i_Y, ByRef $a_Vector3df, $h_Camera, $f_NormalX=0.0, $f_NormalY=0.0, $f_NormalZ=1.0, $f_DistanceFromOrigin=0.0)
```

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGet2DPositionFromScreenCoordinates

[todo]

#include <au3Irrlicht2.au3>
_IrrGet2DPositionFromScreenCoordinates($i_X, $i_Y, ByRef $f_X, ByRef $f_Y, $h_Camera)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetChildCollisionNodeFromRay

[todo]

#include <au3Irrlicht2.au3>
_IrrGetChildCollisionNodeFromRay($h_Node, $i_Mask, $i_Recurse, $a_StartVector, $a_EndVector)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetChildCollisionNodeFromPoint

[todo]

#include <au3Irrlicht2.au3>
_IrrGetChildCollisionNodeFromPoint($h_Node, $i_Mask, $i_Recurse, $a_PointVector)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
#Include <au3Irrlicht2.au3>
_IrrGetNodeAndCollisionPointFromRay($a_StartVector, $a_EndVector, ByVal $h_Node, ByRef $f_PosX, ByRef $f_PosY, ByRef $f_PosZ, ByRef $f_NormalX, ByRef $f_NormalY, ByRef $f_NormalZ, $i_ID = 0, $h_RootNode = $IRR_NO_OBJECT)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetDistanceBetweenNodes

[todo]

#include <au3Irrlicht2.au3>
_IrrGetDistanceBetweenNodes($h_NodeA, $h_NodeB)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
---

**au3Irr2 Function Reference**

__IrrAreNodesIntersecting__

[todo]

```
#include <au3Irrlicht2.au3>
_IrrAreNodesIntersecting($h_NodeA, $h_NodeB)
```

**Parameters**

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

**Return Value**

[success]: [explanation]
[failure]: [explanation]

   [moreExplanationIndented]

**Remarks**

[todo]

**Related**

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrIsPointInsideNode

[todo]

#include <au3Irrlicht2.au3>
_IrrIsPointInsideNode($h_NodeA, $f_X, $f_Y, $f_Z)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetCollisionResultPosition

[todo]

```c
#include <au3Irrlicht2.au3>
_IrrGetCollisionResultPosition($h_Selector, ByRef $a_EllipsoidPosition, ByRef $a_EllipsoidRadius, ByRef $a_Velocity, ByRef $a_Gravity, $f_SlidingSpeed, ByRef $a_OutPosition, ByRef $a_OutHitPosition, ByRef $i_OutFalling)
```

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]

Remarks

[todo]

Related

[todo: functionName, functionName]
### _IrrAddZipFile

Adds a zip archive to the filing system allowing to load files out of the zip file.

```au3
#include <au3Irrlicht2.au3>
_IrrAddZipFile($s_Zipfile, $i_IgnoreCase, $i_IgnorePaths)
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_ZipFile</td>
<td>Path to the zipfile (or pk3 file)</td>
</tr>
<tr>
<td>$i_IgnoreCase</td>
<td>Should be one of the following values:</td>
</tr>
<tr>
<td></td>
<td>$IRR_USE_CASE</td>
</tr>
<tr>
<td></td>
<td>$IRR_IGNORE_CASE</td>
</tr>
<tr>
<td>$i_IgnorePaths</td>
<td>Ignore paths allows you to simply use the filename without the path, the</td>
</tr>
<tr>
<td></td>
<td>filename should always be unique in the archive when using this option. The</td>
</tr>
<tr>
<td></td>
<td>value should be one of the following:</td>
</tr>
<tr>
<td></td>
<td>$IRR_USE_PATHS</td>
</tr>
<tr>
<td></td>
<td>$IRR.Ignore_PATHS</td>
</tr>
</tbody>
</table>

#### Return Value

Success: True  
Failure: False

#### Remarks

Files inside the .zip can be opened as if they were in the current working directory.  
Common pk3 files are simply zip files.
Related
None.

Example

```
#include "au3Irrlicht2.au3"

_IrrStart()

_IrrAddZipFile("./media/map-20kdm2.pk3", $IRR_IGNORE_CASE, $IRR_IGNORE_PATHS)
$BSPMesh = _IrrGetMesh("20kdm2.bsp")
local $BSPNode = _IrrAddMeshToSceneAsOcttree($BSPMesh)

local $camera = _IrrAddFPSCamera()
_IrrSetNodePosition($camera, 1750, 149, 1369)

WHILE _IrrRunning()
   _IrrBeginScene(50, 50, 50)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
au3Irr2 Function Reference

_IrrChangeWorkingDirectory

[todo]

#Include <au3Irrlicht2.au3>
_IrrChangeWorkingDirectory($s_WorkingDir)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetWorkingDirectory

[todo]

#include <au3Irrlicht2.au3>
_IrrGetWorkingDirectory()

Parameters

[param1] [explanation] [moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
__getGuiEvt

Helper function: returns value of $i_Element inside a GuiEvent-structure.

#include <au3Irrlicht2.au3>
__getGuiEvt($p_GUIEvent, $i_Element = $EVT_GUI_IID)

Parameters

<table>
<thead>
<tr>
<th>$p_GUIEvent</th>
<th>A pointer as returned from _IrrReadGUIEvent.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Element</td>
<td>[optional] Event type to return: $EVT_GUI_IID - ID of GUI element as integer, $EVT_GUI_IEVENT - Event type from enum $IRR_EGUI_EVENT_TYPE, $EVT_GUI_IX - X as integer, $EVT_GUI_IY - Y as integer</td>
</tr>
</tbody>
</table>

Return Value

Success: Value of selected event type
Failure: Returns False and sets @error = 1

Remarks
[todo]

Related
_IrrReadGUIEvent
au3Irr2 Function Reference

_IrrGUIClear

[todo]

#include <au3Irrlicht2.au3>
_IrrGUIClear()

Parameters

| [param1] | [explanation] | [moreTextForParam1] |

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGUIEvents

[todo]

#include <au3Irrlicht2.au3>
_IrrGUIEvents($i_EventsForGUI)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
  [moreExplanationIndented]

Remarks

[todo]

Related

todo: functionName, functionName
_IrrGUIEventAvailable

[todo]

#Include <au3Irrlicht2.au3>
_IrrGUIEventAvailable()  

Parameters

[param1] [explanation] [moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
`_IrrReadGUIEvent`

(todo)

```
#include <au3Irrlicht2.au3>
_IrrReadGUIEvent()
```

### Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

### Return Value

- **[success]**: [explanation]
- **[failure]**: [explanation]
  
  moreExplanationIndented

### Remarks

(todo)

### Related

(todo: functionName, functionName)
au3Irr2 Function Reference

_IrrGUIRemove

[todo]

#include <au3Irrlicht2.au3>
_IrrGUIRemove($h_Widget)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
### _IrrGUIGetText

[todo]

```au3
#Include <au3Irrlicht2.au3>
_IrrGUIGetText($h_Widget)
```

#### Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

#### Return Value

- [success]: explanation
- [failure]: explanation
  - moreExplanationIndented

#### Remarks

[todo]

#### Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGUISetText

[todo]

#include <au3Irrlicht2.au3>
_IrrGUISetText($h_Widget, $s_Text)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
           [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGUISetFont

[todo]

#include <au3Irrlicht2.au3>
_IrrGUISetFont($h_Font)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGUISetColor

[todo]

#include <au3Irrlicht2.au3>
_IrrGUISetColor($i_Element, $i_Red, $i_Green, $i_Blue, $i_Alpha)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrAddWindow

todo

#include <au3Irrlicht2.au3>
_IrrAddWindow($s_Title, $i_TopX, $i_TopY, $i_BottomX, $i_BottomY, $i_Modal, $h_Parent = 0)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[succcess]: [explanation]
[failure]: [explanation]

Remarks

todo

Related

todo: functionName, functionName
_IrrAddStaticText

Creates a static text object on the Graphical User Interface.

```au3
#include <au3Irrlicht2.au3>
[IrrAddStaticText($s_Text, $i_TopX, $i_TopY, $i_BottomX, $i_BottomY, $i_Border, $i_WordWrap, $h_Parent = 0)]
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_Text</td>
<td>String that contains the text you want to display.</td>
</tr>
<tr>
<td>$i_TopX</td>
<td>Top X position of a box in which the text is drawn</td>
</tr>
<tr>
<td>$i_TopY</td>
<td>Top Y position of a box in which the text is drawn</td>
</tr>
<tr>
<td>$i_BottomX</td>
<td>Bottom X position of a box in which the text is drawn</td>
</tr>
<tr>
<td>$i_BottomY</td>
<td>Bottom Y position of a box in which the text is drawn</td>
</tr>
<tr>
<td>$i_Border</td>
<td>Border is used to draw a visible box around the text, its value should be either of: $IRR_GUI_NO_BORDER $IRR_GUI_BORDER</td>
</tr>
<tr>
<td>$i_WordWrap</td>
<td>Word wrap is used to define whether text is to be wrapped around into a second line when it fills the width of the text box, its value should be either of: $IRR_GUI_NO_WRAP $IRR_GUI_WRAP</td>
</tr>
<tr>
<td>$h_Parent</td>
<td>[optional] Parent defines the parent object of this window. This can be omminated if the object has no parent.</td>
</tr>
</tbody>
</table>

### Return Value

- **Success:** Pointer to the static text Object
- **Failure:** False
Remarks
This function simply displays the specified text in the specified box.

Related
_IrrDrawGUI

Example

```cpp
#include "au3Irrlicht2.au3"

_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _
            $IRR_WINDOWED, $IRR_SHADOWS, $IRR_CAPTURE_EVENTS,
            $IRR_VERTICAL_SYNC_ON )

_IrrAddStaticText( "Hello AU3-World ;-)", 8, 8, 200, 20,
                    $IRR_GUI_NO_BORDER, $IRR_GUI_NO_WRAP )

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 0)
    _IrrDrawGUI()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
au3Irr2 Function Reference

_IrrAddButton

[todo]

#include <au3Irrlicht2.au3>
_IrrAddButton($i_TopX, $i_TopY, $i_BottomX, $i_BottomY, $i_ID, $s_Text = "", $s_TextTip = "", $h_Parent = 0)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3lrr2 Function Reference

_IrrAddScrollBar

todo

#include <au3lrrlicht2.au3>
_IrrAddScrollBar($i_Horizontal, $i_TopX, $i_TopY, $i_BottomX, $i_BottomY, $i_ID, $i_CurrentValue, $i_MaxValue, $h_Parent = 0)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
  moreExplanationIndented

Remarks
todo

Related
todo: functionName, functionName
au3Irr2 Function Reference

_IrrAddListBox

[todo]

#include <au3Irrlicht2.au3>
_IrrAddListBox($i_TopX, $i_TopY, $i_BottomX, $i_BottomY, $i_ID, $i_Background, $h_Parent=0)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
Function Reference

_IrrAddListBoxItem

todo]

- Include <au3Irrlicht2.au3>
- _IrrAddListBoxItem($h_ListBox, $s_Text)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

- [success]: [explanation]
- [failure]: [explanation]
  - [moreExplanationIndented]

Remarks

todo]

Related

todo: functionName, functionName
# Include <au3Irrlicht2.au3>

/_IrrSelectListBoxItem($h_ListBox, $i_Index)

## Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

## Return Value

[success]: [explanation]

[failure]: [explanation]

  [moreExplanationIndented]

## Remarks

[todo]

## Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrAddEditBox

Parameter

Return Value

Remarks

Related
au3Irr2 Function Reference

_IrrAddCheckBox

todo

#include <au3Irrlicht2.au3>
_IrrAddCheckBox($s_Text, $i_TopX, $i_TopY, $i_BottomX, $i_BottomY, $i_ID, $i_Checked, $h_Parent = 0)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

todo

Related

todo: functionName, functionName
au3Irr2 Function Reference

_IrrAddImage

[todo]

```
#include <au3Irrlicht2.au3>
_IrrAddImage($h_Texture, $i_X, $i_Y, $i_UseAlpha, $i_ID, $h_Parent = 0)
```

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[comment]</td>
</tr>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrAddFileOpen

[todo]

#include <au3Irrlicht2.au3>
_IrrAddFileOpen($s_Title, $i_ID, $i_Modal, $h_Parent = 0)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetLastSelectedFile

[todo]

#include <au3Irrlicht2.au3>
_IrrGetLastSelectedFile()

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[succcess]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
Function Reference

___getKeyEvt

helper function: returns value of $i_Element inside a keyEvent-structure.

```c
#include <au3Irrlicht2.au3>
__getKeyEvt($p_KeyEvent, $i_Element = $EVT_KEY_IKEY)
```

Parameters

<table>
<thead>
<tr>
<th>$p_KeyEvent</th>
<th>A pointer as returned from _IrrReadKeyEvent.</th>
</tr>
</thead>
</table>
| $i_Element   | [optional] Event type to return:  
|              | $EVT_KEY_IKEY - ID of pressed key (see remarks).  
|              | $EVT_KEY_IDIRECTION - Direction value - can be either  
|              | $IRR_KEY_DOWN or $IRR_KEY_UP  
|              | $EVT_KEY_IFLAGS - Bits are set in this parameter to  
|              | specify whether the shift or control key was keydown at  
|              | the time the key action occured. |

Return Value

Success: Value of selected event element.

Remarks

$p_KeyEvent is a pointer as returned from _IrrReadKeyEvent.

Key code table for $EVT_KEY_IKEY:

<table>
<thead>
<tr>
<th>$KEY_LBUTTON</th>
<th>Left mouse button</th>
</tr>
</thead>
<tbody>
<tr>
<td>$KEY_RBUTTON</td>
<td>Control-break</td>
</tr>
<tr>
<td>Key Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$KEY_CANCEL</td>
<td>processing</td>
</tr>
<tr>
<td>$KEY_XBUTTON1</td>
<td>Windows 2000/XP: X1 mouse button</td>
</tr>
<tr>
<td>$KEY_BACK</td>
<td>BACKSPACE key</td>
</tr>
<tr>
<td>$KEY_CLEAR</td>
<td>Clear key</td>
</tr>
<tr>
<td>$KEY_SHIFT</td>
<td>SHIFT key</td>
</tr>
<tr>
<td>$KEY_MENU</td>
<td>ALT key</td>
</tr>
<tr>
<td>$KEY_CAPITAL</td>
<td>CAPS Lock key</td>
</tr>
<tr>
<td>$KEY_HANGUEUL</td>
<td>IME Hanguel mode (maintained for compatibility use KEY_HANGUL)</td>
</tr>
<tr>
<td>$KEY_JUNJA</td>
<td>IME Junja mode</td>
</tr>
<tr>
<td>$KEY_HANJA</td>
<td>IME Hanja mode</td>
</tr>
<tr>
<td>$KEY_ESCAPE</td>
<td>ESC key</td>
</tr>
<tr>
<td>$KEY_NONCONVERT</td>
<td>IME nonconvert</td>
</tr>
<tr>
<td>$KEY_MODECHANGE</td>
<td>IME mode change request</td>
</tr>
<tr>
<td>$KEY_PRIOR</td>
<td>PAGE UP key</td>
</tr>
<tr>
<td>$KEY_END</td>
<td>End key</td>
</tr>
<tr>
<td>$KEY_LEFT</td>
<td>Left ARROW key</td>
</tr>
<tr>
<td>Key Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>$KEY_RIGHT</td>
<td>Right ARROW key</td>
</tr>
<tr>
<td>$KEY_SELECT</td>
<td>Select key</td>
</tr>
<tr>
<td>$KEY_EXECUT</td>
<td>EXECUTE key</td>
</tr>
<tr>
<td>$KEY_INSERT</td>
<td>INSERT key</td>
</tr>
<tr>
<td>$KEY_HELP</td>
<td>HELP key</td>
</tr>
<tr>
<td>$KEY_KEY_1</td>
<td>1 key</td>
</tr>
<tr>
<td>$KEY_KEY_9</td>
<td>9 key</td>
</tr>
<tr>
<td>$KEY_KEY_B</td>
<td>B key</td>
</tr>
<tr>
<td>$KEY_KEY_Z</td>
<td>Z key</td>
</tr>
<tr>
<td>$KEY_RWIN</td>
<td>Right Win key (Natural keyboard)</td>
</tr>
<tr>
<td>$KEY_SLEEP</td>
<td>Computer Sleep key</td>
</tr>
<tr>
<td>$KEY_NUMPAD1</td>
<td>Numeric keypad 1 key</td>
</tr>
<tr>
<td>$KEY_NUMPAD9</td>
<td>Numeric keypad 9 key</td>
</tr>
<tr>
<td>$KEY_ADD</td>
<td>Add key</td>
</tr>
<tr>
<td>$KEY_SUBTRACT</td>
<td>Subtract key</td>
</tr>
<tr>
<td>$KEY_DIVIDE</td>
<td>Divide key</td>
</tr>
<tr>
<td>$KEY_F2</td>
<td>F2 key</td>
</tr>
<tr>
<td>$KEY_F24</td>
<td>F24 key</td>
</tr>
</tbody>
</table>
$KEY_SCROLL | SCROLL Lock key | $KEY_LSHIFT | LOCK
$KEY_RSHIFT | Right SHIFT key | $KEY_LCONTROL | LOCK
$KEY_RCONTROL | Right CONTROL key | $KEY_LMENU | LOCK
$KEY_RMENU | Right MENU key | $KEY_PLUS | RIGHT
$KEY_COMMA | Comma Key (,) | $KEY_MINUS | RIGHT
$KEY_PERIOD | Period Key (.) | $KEY_ATTN | RIGHT
$KEY_CRSEL | CrSel key | $KEY_EXSEL | RIGHT
$KEY_EREOF | Erase Eof key | $KEY_PLAY | RIGHT
$KEY_ZOOM | Zoom key | $KEY_PA1 | RIGHT
$KEY_OEM_CLEAR | Clear key | $KEY_KEY_CODES_COUNT | RIGHT

Related
_IrrReadKeyEvent, _IrrKeyEventAvailable

Example

```
#include "au3Irrlicht2.au3"

local $pKeyEvent;

template _IrrStart($IRR_EDT_DIRECT3D9, 800, 600, $IRR_BITS_PER_PIXEL_32,
```

; enable event capturing:
_IrrStart($IRR_EDT_DIRECT3D9, 800, 600, $IRR_BITS_PER_PIXEL_32,
$IRR_WINDOWED, $IRR_NO_SHADOWS, $IRR_CAPTURE_EVENTS)

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 25)

    ; process all available key events and check for ESC:
    while _IrrKeyEventAvailable()
      $pKeyEvent = _IrrReadKeyEvent()
      if __getKeyEvt($pKeyEvent, $EVT_KEY_IKEY) = $KEY_ESCAPE
        then _IrrStop($pKeyEvent, $EVT_KEY_IKEY) = $KEY_ESCAPE
      WEnd

  _IrrEndScene()
WEND
__getMouseEvt

helper function: returns value of $i_Element inside a MouseEvent-structure.

```
#include <au3Irrlicht2.au3>
__getMouseEvt($p_MouseEvent, $i_Element = $EVT_MOUSE_IACTION)
```

**Parameters**

<table>
<thead>
<tr>
<th>$p_MouseEvent</th>
<th>A pointer as returned from __IrrReadMouseEvent.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Element</td>
<td>[optional] Event type to return:</td>
</tr>
<tr>
<td></td>
<td>$EVT_MOUSE_IACTION - ID of mouse action (see remarks).</td>
</tr>
<tr>
<td></td>
<td>$EVT_MOUSE_FDELTA - Amount of movement of the mouse wheel (&gt; 0 means wheel up, &lt; 0 means wheel down).</td>
</tr>
<tr>
<td></td>
<td>$EVT_MOUSE_IX - Horizontal screen coordinate at which the event took place.</td>
</tr>
<tr>
<td></td>
<td>$EVT_MOUSE_IY - Vertical screen coordinate at which the event took place.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Value of selected event element.

**Remarks**

$p_MouseEvent is a pointer as returned from __IrrReadMouseEvent.

**Mouse actions table for $EVT_MOUSE_IACTION:**

<table>
<thead>
<tr>
<th></th>
<th>Left mouse button pressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>__IRR_EMIE_LMOUSE_PRESSED_DOWN</td>
<td>Right mouse button</td>
</tr>
</tbody>
</table>
$IRR_EMIE_RMOUSE_PRESSED_DOWN | pressed
$IRR_EMIE_MMOUSE_PRESSED_DOWN | Middle mouse button pressed
$IRR_EMIE_LMOUSE_LEFT_UP | Left mouse button released
$IRR_EMIE_RMOUSE_LEFT_UP | Right mouse button released
$IRR_EMIE_MMOUSE_LEFT_UP | Middle mouse button released
$IRR_EMIE_MOUSE_MOVED | Mouse was moved horizontal and/or vertical
$IRR_EMIE_MOUSE_WHEEL | Mouse wheel was moved up or down

Related
_IrrReadMouseEvent, _IrrMouseEventAvailable

Example

```
#include "au3Irrlicht2.au3"

local $pMouseEvent

; enable event capturing:
_IrrStart($IRR_EDT_DIRECT3D9, 800, 600, $IRR_BITS_PER_PIXEL_32,
    _IRR_WINDOWED, $IRR_NO_SHADOWS,
    $IRR_CAPTURE_EVENTS)

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 25)
    ; process all available mouse events:
```
while _IrrMouseEventAvailable()
    $pMouseEvent = _IrrReadMouseEvent()

    ; check for mousewheel event, report up or down movement inside this window title:
    if __getMouseEvt($pMouseEvent, $EVT_MOUSE_IACTION) = $IRR_EMIE_MOUSE_WHEEL then
        if __getMouseEvt($pMouseEvent, $EVT_MOUSE_FDELTA) > 0
            _IrrSetWindowCaption("Mousewheel up")
        Else
            _IrrSetWindowCaption("Mousewheel down")
        EndIf
    endif
wend

_IrrEndScene()
WEND
_IrrKeyEventAvailable

Determine if there are any keystrokes waiting to be read.

```au3
#include <au3Irrlicht2.au3>
_IIrrKeyEventAvailable()
```

**Parameters**

None.

**Return Value**

Success: True if there are keystrokes, otherwise False.

**Remarks**

Event capturing needs to be enabled before with _IrrStart or _IrrStartAdvanced!

**Related**

_IIrrReadKeyEvent, __getKeyEvt, _IrrStart

**Example**

```au3
#include "au3Irrlicht2.au3"
local $pKeyEvent
```
; enable event capturing:
_IrrStart($IRR_EDT_DIRECT3D9, 800, 600, $IRR_BITS_PER_PIXEL_32, 
-  $IRR_WINDOWED, $IRR_NO_SHADOWS, 
$IRR_CAPTURE_EVENTS)

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 25)

  ; process all available key events and check for ESC:
  while _IrrKeyEventAvailable()
    $pKeyEvent = _IrrReadKeyEvent()
    if __getKeyEvt($pKeyEvent, $EVT_KEY_IKEY) = $KEY_ESCAPE
      then _IrrStop($pKeyEvent, $EVT_KEY_IKEY) = $KEY_ESCAPE
  WEnd

  _IrrEndScene()
WEND
_IrrReadKeyEvent

Read a key event from the Irrlicht window.

```
#include <au3Irrlicht2.au3>
_IrrReadKeyEvent()
```

**Parameters**

None.

**Return Value**

success: Pointer of a key event.
failure: False

**Remarks**

The properties of the returned key event are readable with the helper function `__getKeyEvt`.

**Related**

_`_IrrKeyEventAvailable`, `__getKeyEvt`

**Example**

```
#include "au3Irrlicht2.au3"

local $pKeyEvent
```
; enable event capturing:
_IrrStart($IRR_EDT_DIRECT3D9, 800, 600, $IRR_BITS_PER_PIXEL_32,
- $IRR_WINDOWED, $IRR_NO_SHADOWS,
$IRR_CAPTURE_EVENTS)

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 25)

    ; process all available key events and check for ESC:
    while _IrrKeyEventAvailable()
        $pKeyEvent = _IrrReadKeyEvent()
        if __getKeyEvt($pKeyEvent, $EVT_KEY_IKEY) = $KEY_ESCAPE
            then _IrrStop($pKeyEvent, $EVT_KEY_IKEY) = $KEY_ESCAPE
            WEnd

            _IrrEndScene()
        WEND

Copy to Clipboard
_IrrMouseEventAvailable

Determine if there are any mouse events waiting to be read.

```
#include <au3Irrlicht2.au3>
_IrrMouseEventAvailable()
```

**Parameters**

None.

**Return Value**

Success: True if there are mouse events, otherwise False.

**Remarks**

Event capturing needs to be enabled before with _IrrStart or _IrrStartAdvanced!

**Related**

_İrrReadMouseEvent, __getMouseEvt, _İrrStart

**Example**

```
#include "au3Irrlicht2.au3"

local $pMouseEvent
```
; enable event capturing:
_IrrStart($IRR_EDT_DIRECT3D9, 800, 600, $IRR_BITS_PER_PIXEL_32, 
- $IRR_WINDOWED, $IRR_NO_SHADOWS, $IRR_CAPTURE_EVENTS)

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 25)

    ; process all available mouse events:
    while _IrrMouseEventAvailable()
        $pMouseEvent = _IrrReadMouseEvent()

    ; check for mousewheel event, report up or down movement inside this window title:
    if __getMouseEvt($pMouseEvent, $EVT_MOUSE_IACTION) = $IRR_EMIE_MOUSE_WHEEL then
        if __getMouseEvt($pMouseEvent, $EVT_MOUSE_FDELTA) > 0
            _IrrSetWindowCaption("Mousewheel up")
        Else
            _IrrSetWindowCaption("Mousewheel down")
        EndIf
    endif
wend

_IrrEndScene()
WEND
__IrrReadMouseEvent

Read a key event from the Irrlicht window.

#include <au3Irrlicht2.au3>
__IrrReadMouseEvent()

Parameters

None.

Return Value

success: Pointer of a mouse event.
failure: False

Remarks

The properties of the returned mouse event are readable with the helper function __getMouseEvt.

Related

__IrrMouseEventAvailable, __getMouseEvt

Example

#include "au3Irrlicht2.au3"

local $pMouseEvent
; enable event capturing:
_IrrStart($IRR_EDT_DIRECT3D9, 800, 600, $IRR_BITS_PER_PIXEL_32,
  $IRR_WINDOWED, $IRR_NO_SHADOWS, $IRR_CAPTURE_EVENTS)

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 25)

; process all available mouse events:
while _IrrMouseMoveAvailable()
  $pMouseEvent = _IrrReadMouseMove()

; check for mousewheel event, report up or down movement inside this window title:
if __getMouseEvt($pMouseEvent, $EVT_MOUSE_IACTION) = $IRR_EMIE_MOUSE_WHEEL then
  if __getMouseEvt($pMouseEvent, $EVT_MOUSE_FDELTA) > 0
    _IrrSetWindowCaption("Mousewheel up")
  Else
    _IrrSetWindowCaption("Mousewheel down")
  EndIf
endif
wend

_IrrEndScene()
WEND
_IrrSetMousePosition

Sets relative position of the mouse pointer and returns relative position before this change.

```
#include <au3Irrlicht2.au3>
_IrrSetMousePosition(ByRef $f_XPos, ByRef $f_YPos)
```

**Parameters**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_XPos</td>
<td>Fractional value for new horizontal position (0-1).</td>
</tr>
<tr>
<td>$f_YPos</td>
<td>Fractional value for new vertical position (0-1).</td>
</tr>
</tbody>
</table>

**Return Value**

success: True and sets $f_XPos and $f_YPos to relative position where the mouse was before (both 0-1).

failure: False

**Remarks**

This function works independent from the resolution of current Irrlicht display. Expected and returned values are fractional values, where 0/0 is top left and 1/1 bottom right of the Irrlicht display.

**Related**

_ _IrrReadMouseEvent, _IrrGetAbsoluteMousePosition

**Example**
#include "au3Irrlicht2.au3"

_IrrStart()

_IrrAddSkyDomeToScene(_IrrGetTexture("./media/au3irr2_logo.jpg"), 16, 16, 1, 2)
local $nodeCamera = _IrrAddCamera(0,0,5, 0,0,0)
local $X, $Y, $camY

_IrrHideMouse()
WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 25)

   ; center mouse cursor
   $X = .5
   $Y = .5
   _IrrSetMousePosition($X, $Y)

   ; add any mouse movement to the cam-Y position ...
   $camY += (0.5 - $Y)

   ; ... and reset it slowly back to centre of the display:
   if $camY > 0 then $camY -= 0.005
   if $camY < 0 then $camY += 0.005
   _IrrSetCameraTarget($nodeCamera, 0, $camY, 0)

   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
IrrGetAbsoluteMousePosition

Get the absolute mouse X and Y position

#include <au3Irrlicht2.au3>
_IrrGetAbsoluteMousePosition()

Parameters

None.

Return Value

success: 1D Array containing absolute mouse position.
  $Array[0] = X position of the mouse
  $Array[1] = Y position of the mouse
failure: Empty 1D Array and Set @error 1

Remarks

None.

Related

Example

#include <au3Irrlicht2.au3>

Global $aGAMP
_IrrStart()

While _IrrRunning()

$aGAMP = _IrrGetAbsoluteMousePosition()
_IrrSetWindowCaption("Absolute Mouse Poition - X: " & $aGAMP[0] & " Y: " & $aGAMP[1])

WEnd

_IrrStop()

Copy to Clipboard
_IrrHideMouse

Hides the mouse pointer

```
#include <au3Irrlicht2.au3>
_IrrHideMouse()
```

**Parameters**

None.

**Return Value**

Success: True  
Failure: False

**Remarks**

None

**Related**

_**IrrShowMouse**,  _**IrrDisplayMouse**

**Example**

```
#include "au3Irrlicht2.au3"

_IrrStart()
```
local $camera = _IrrAddCamera(2,2,2, 0,0,0 )

local $time = TimerInit()
WHILE _IrrRunning()

    if int(mod( TimerDiff($time) / 1000, 2 )) = true Then
        _IrrHideMouse()
    Else
        _IrrShowMouse()
    EndIf

    _IrrBeginScene(50, 50, 50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
_IrrShowMouse

Shows the mouse pointer

```
#include <au3Irrlicht2.au3>
_IrrShowMouse()
```

Parameters
None.

Return Value
Success: True
Failure: False

Remarks
None

Related
_IrrHideMouse, _IrrDisplayMouse

Example
```
#include "au3Irrlicht2.au3"

_IrrStart()
```
local $camera = _IrrAddCamera(2, 2, 2, 0, 0, 0)

local $time = TimerInit()
WHILE _IrrRunning()

   if int(mod(TimerDiff($time) / 1000, 2)) = true Then
      _IrrHideMouse()
   Else
      _IrrShowMouse()
   EndIf

   _IrrBeginScene(50, 50, 50)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
_IrrDisplayMouse

Hide or show the mouse pointer while it is within the Irrlicht display.

```au3
#include <au3Irrlicht2.au3>
_IrrDisplayMouse($i_HideShow)
```

**Parameters**

$i_HideShow True shows and False hides the mouse pointer.

**Return Value**

Success: True
Failure: False

**Remarks**

There are two functions available to simply hide or show the mouse: IrrHideMouse and IrrShowMouse.

**Related**

_IrrShowMouse, _IrrHideMouse

**Example**

```au3
#include "au3Irrlicht2.au3"
_IrrStart()
```
local $camera = _IrrAddCamera(0,0,0,0,0,0)

local $time = TimerInit()
WHILE _IrrRunning()

  _IrrDisplayMouse( int(mod(TimerDiff($time) / 1000), 2))

  _IrrBeginScene(50,50,50)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
_IrrAddLight

Adds a light node into scene to naturally illuminate your scene.

```au3
#include <au3Irrlicht2.au3>
_IrrAddLight($h_parentNode, $f_X, $f_Y, $f_Z, $f_Red, $f_Green, $f_Blue, $f_Size)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_parentNode</td>
<td>Handle of the node to attach the light to. $IRR_NO_PARENT attaches to the root node of the scene.</td>
</tr>
<tr>
<td>$f_X, $f_Y, $f_Z</td>
<td>Coordinates of the light in the scene</td>
</tr>
<tr>
<td>$f_Red, $f_Green, $f_Blue</td>
<td>Intensity of the light. <strong>Red/green/blue are fractional values from 0 to 1!</strong></td>
</tr>
<tr>
<td>$f_Size</td>
<td>Radius of effect of the light</td>
</tr>
</tbody>
</table>

### Return Value

- **Success:** Handle of light node in the scene
- **Failure:** False

### Remarks

When using shadows you probably only want one or two lights - they can be time consuming.

### Related
Example

```c
#include "au3Irrlicht2.au3"

_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _
    $IRR_WINDOWED, $IRR_SHADOWS, $IRR_IGNORE_EVENTS,
    $IRR_VERTICAL_SYNC_ON )

local $nodeSphere = _IrrAddSphereSceneNode(4, 32)
_IrrSetNodeMaterialTexture( $nodeSphere,
    _IrrGetTexture("\media\au3irr2_logo.jpg"), 0)

local $nodeLight = _IrrAddLight($IRR_NO_PARENT, 0, 50, 0, 0.9, 0.9, 0.9, 1000.0)
_IrrAddFlyCircleAnimator($nodeLight, 0, 0, 0, 500, 0.001)

local $nodeCamera = _IrrAddCamera(-5,5,-6, 0,0,0 )

WHILE _IrrRunning()  
    _IrrBeginScene(100, 100, 100) 
    _IrrDrawScene() 
    _IrrEndScene() 
WEND

_IrrStop()

Copy to Clipboard
```
au3Irr2 Function Reference

_IrrSetAmbientLight

Sets the ambient lighting level across entire scene.

```
#include <au3Irrlicht2.au3>
_IrrSetAmbientLight($f_Red, $f_Green, $f_Blue)
```

**Parameters**

| $i_Red, $i_Green, $i_Blue | Colour values for ambient lighting. Red/green/blue are fractional values from 0 to 1! |

**Return Value**

Success: True  
Failure: False

**Remarks**

Ambient light illuminates all surfaces in the scene uniformly. This is usually a low value to increase the overall lighting level. It should never be greater than the brightness of the darkest area of your scene, it can however reduce the number of lights you need in the scene.

**Related**

_IrrAddLight, _IrrSetLightAmbientColor

**Example**

#include "au3Irrlicht2.au3"

_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _
      $IRR_WINDOWED, $IRR_SHADOWS, $IRR_IGNORE_EVENTS, $IRR_VERTICAL_SYNC_ON )

local $nodeSphere = _IrrAddSphereSceneNode(4, 32)
_IrrSetNodeMaterialTexture( $nodeSphere, _IrrGetTexture("\media\au3irr2_logo.jpg"), 0)

local $nodeLight = _IrrAddLight($IRR_NO_PARENT, 0, 0, 50, 0, 0, 0.9, 0.9, 0.9, 1000.0)
_IrrSetAmbientLight(8, 0, 0)
_IrrAddFlyCircleAnimator($nodeLight, 0, 0, 0, 500, 0.001)

local $nodeCamera = _IrrAddCamera(-5, -5, -6, 0, 0, 0 )

WHILE _IrrRunning() 
      _IrrBeginScene(100, 100, 100)
      _IrrDrawScene()
      _IrrEndScene()
WEND

_IrrStop()
au3Irr2 Function Reference

_IrrSetLightAmbientColor

[todo]

#include <au3Irrlicht2.au3>
_IrrSetLightAmbientColor($h_Light, $f_Red, $f_Green, $f_Blue)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrSetLightAttenuation

[todo]

#include <au3Irrlicht2.au3>
_IrrSetLightAttenuation($h_Light, $f_Red, $f_Green, $f_Blue)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
# _IrrSetLightDiffuseColor

[Include `<au3Irrlicht2.au3>`]

```au3
_IrrSetLightDiffuseColor($h_Light, $f_Red, $f_Green, $f_Blue)
```

## Parameters

<table>
<thead>
<tr>
<th>Param1</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Return Value

- **Success**: Explanation
- **Failure**: Explanation
  - More Explanation Indented

## Remarks

[Todo]

## Related

[Todo: functionName, functionName]
#au3Irr2 Function Reference

##_IrrSetLightFalloff

[todo]

```
#include <au3Irrlicht2.au3>
_IrrSetLightFalloff($h_Light, $f_Falloff)
```

### Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

### Return Value

- [success]: [explanation]
- [failure]: [explanation]
  - [moreExplanationIndented]

### Remarks

[todo]

### Related

[todo: functionName, functionName]
_IrrSetLightInnerCone

[todo]

#include <au3Irrlicht2.au3>
_IrrSetLightInnerCone($h_Light, $f_InnerCone)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrSetLightOuterCone

[todo]

#include <au3Irrlicht2.au3>
_IrrSetLightOuterCone($h_Light, $f_OuterCone)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[ failure]: [explanation]
[   moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetLightType

[todo]

#include <au3Irrlicht2.au3>
_IrrSetLightType($h_Light, $i_Type)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[TODO]

Related

[TODO: functionName, functionName]
_IrrSetNodeAmbientColor

[todo]

```c
#include <au3Irrlicht2.au3>
_IrrSetNodeAmbientColor($h_Node, $i_Color)
```

**Parameters**

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

**Return Value**

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

**Remarks**

[todo]

**Related**

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetNodeDiffuseColor

[todo]

#Include <au3Irrlicht2.au3>
_IrrSetNodeDiffuseColor($h_Node, $i_Color)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrSetNodeSpecularColor

[todo]

#include <au3Irrlicht2.au3>
_IrrSetNodeSpecularColor($h_Node, $i_Color)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetNodeEmissiveColor

[todo]

#include <au3Irrlicht2.au3>
_IrrSetNodeEmissiveColor($h_Node, $i_Color)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrSetNodeColorByVertex

[todo]

#include <au3Irrlicht2.au3>
_IrrSetNodeColorByVertex($h_Node, $i_ColorMaterial)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
#_IrrMaterialVertexColorAffects

[.todo]

```c
#include <au3Irrlicht2.au3>
#include <au3Irrlicht2.au3>
_ IrrMaterialVertexColorAffects($h_Material, $i_AffectedProperty)
```

## Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

## Return Value

[success]: explanation

[failure]: explanation

   [moreExplanationIndented]

## Remarks

[.todo]

## Related

[.todo: functionName, functionName]
Function Reference

_IrrSetMaterialBlend

[ todo ]

#include <au3Irrlicht2.au3>
_IrrSetMaterialBlend($h_Material, $i_SrcBlend, $i_DstBlend)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
  moreExplanationIndented

Remarks

[ todo ]

Related

[ todo: functionName, functionName ]
_IrrMaterialSetShininess

[todo]

#include <au3Irrlicht2.au3>
_IrrMaterialSetShininess($h_Material, $f_Shininess)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrMaterialSetSpecularColor

[todo]

`#Include <au3Irrlicht2.au3>
_IrrMaterialSetSpecularColor($h_Material, $i_Alpha, $i_Red, $i_Green, $i_Blue)`

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrMaterialSetDiffuseColor

[todo]

```c
#include <au3Irrlicht2.au3>
void _IrrMaterialSetDiffuseColor($h_Material, $i_Alpha, $i_Red, $i_Green, $i_Blue)
```

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
        [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrMaterialSetAmbientColor

[todo]

#include <au3irrlicht2.au3>
_IrrMaterialSetAmbientColor($h_Material, $i_Alpha, $i_Red, $i_Green, $i_Blue)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
#au3Irr2 Function Reference

**_IrrMaterialSetEmissiveColor**

[todo]

```cpp
#include <au3Irrlicht2.au3>
_IrrMaterialSetEmissiveColor($h_Material, $i_Alpha, $i_Red, $i_Green, $i_Blue)
```

**Parameters**

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

**Return Value**

[success]: [explanation]

[failure]: [explanation]

  [moreExplanationIndented]

**Remarks**

[todo]

**Related**

[todo: functionName, functionName]
_IrrMaterialSetMaterialTypeParam

[todo]

#include <au3Irrlicht2.au3>
_IrrMaterialSetMaterialTypeParam($h_Material, $f_Param)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrXEffectsStart

[todo]

```
#include <au3Irrlicht2.au3>
_IrrXEffectsStart($i_Vsm=$IRR_OFF, $i_SoftShadows=$IRR_OFF, $iBitdepth32=$IRR_OFF)
```

### Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

### Return Value

- [success]: [explanation]
- [failure]: [explanation]
  - [moreExplanationIndented]

### Remarks

[todo]

### Related

[todo: functionName, functionName]
_IrrXEffectsAddShadowToNode

[todo]

#include <au3Irrlicht2.au3>
_IrrXEffectsAddShadowToNode($h_Node, $i_FilterType=$EFT_NONE, $i_shadowType=$ESM_BOTH)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrXEffectsAddShadowLight

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

todo

Related

todo: functionName, functionName
### _IrrXEffectsSet Ambient Color

[todo]

```au3
#include <au3Irrlicht2.au3>
_IrrXEffectsSetAmbientColor($i_R, $i_G, $i_B, $i_Alpha)
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
<th>More Text For Param1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[param1]</td>
<td>[explanation]</td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

#### Return Value

- [success]: [explanation]
- [failure]: [explanation]
  - [moreExplanationIndented]

#### Remarks

[todo]

#### Related

[todo: functionName, functionName]
_IrrXEffectsSetClearColor

[todo]

#include <au3Irrlicht2.au3>
_IrrXEffectsSetClearColor($i_R, $i_G, $i_B, $i_Alpha)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
IrrXEffectsSetShadowLightPosition

(todo)

#include <au3Irrlicht2.au3>
_IrrXEffectsSetShadowLightPosition($i_Index, $f_PosX, $f_PosY, $f_PosZ)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

(todo)

Related

[todo: functionName, functionName]
_IrrXEffectsEnableDepthPass

todo

#include <au3Irrlicht2.au3>
_IrrXEffectsEnableDepthPass($i_Enable)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
  [moreExplanationIndented]

Remarks

todo

Related

todo: functionName, functionName
_IrrXEffectsAddPostProcessingFromFile

[todo]

#include <au3Irrlicht2.au3>
_IrrXEffectsAddPostProcessingFromFile($s_Name, $i_Enable=0)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

Return Value

[success]: explanation
[failure]: explanation
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrXEffectsAddNodeToDepthPass

[todo]

#include <au3Irrlicht2.au3>
_IrrXEffectsAddNodeToDepthPass($h_Node)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrXEffectsSetPostProcessingUserTexture

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

Return Value

[success]: explanation
[failure]: explanation
  [moreExplanationIndented]

Remarks

todo]

Related

todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetNodeName

[todo]

#include <au3Irrlicht2.au3>
_IrrGetNodeName($h_Node)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetNodeName

[todo]

#include <au3Irrlicht2.au3>
_IrrSetNodeName($h_Node, $s_Name)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetNodeMesh

[todo]

#include <au3Irrlicht2.au3>
_IrrGetNodeMesh ($h_Node)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetMaterialCount

[/todo]

#include <au3Irrlicht2.au3>
_IrrGetMaterialCount($h_Node)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetMaterial

[todo]

#include <au3Irrlicht2.au3>
_IrrGetMaterial($h_Node, $i_Material)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetNodeMaterialTexture

Applies a texture to a node in the scene.

```
#include <au3Irrlicht2.au3>
_IrrSetNodeMaterialTexture($h_Node, $h_Texture, $i_Index)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a node in the scene</td>
</tr>
<tr>
<td>$h_Texture</td>
<td>Handle of a texture object</td>
</tr>
<tr>
<td>$i_Index</td>
<td>Material index number of the material layer, usually 0 or 1.</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False

Remarks

How the texture is applied across the surface of the node will depend on the texturing coordinates in each of the vectors of the mesh and how they are plotted across the surface of the texture.
Some nodes can have several textures applied to them to create special material effects.

Related

_IrrSetNodeMaterialFlag, _IrrSetNodeMaterialType

Example
```c
#include "au3Irrlicht2.au3"

!IrrStart()

local $camera = !_IrrAddCamera(2,2,2, 0,0,0 )

local $mesh = !_IrrGetMesh( "\media\capsuleX.obj" )
local $sceneNode = !_IrrAddMeshToScene( $mesh )

local $texture = !_IrrGetTexture("\media\default_texture.png")
!IrrSetNodeMaterialTexture( $sceneNode, $texture, 0)
!IrrSetNodeMaterialFlag( $sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

WHILE !_IrrRunning()
   !_IrrBeginScene(50, 50, 50)
   !_IrrDrawScene()
   !_IrrEndScene()
WEND

!IrrStop()
```
_IrrSetNodeMaterialFlag

Sets material properties of a node that will affect its appearance on the screen.

```
#include <au3Irrlicht2.au3>
_ IrrSetNodeMaterialFlag($h_Node, $i_Type, $i_Flag)
```

**Parameters**

<table>
<thead>
<tr>
<th>$h_Node</th>
<th>Handle of a node in the scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Type</td>
<td>Material type is one of the following properties:</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_WIREFRAME - Render as wireframe outline</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_GOURAUD_SHADING - Render smoothly across polygons</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_LIGHTING - Material is effected by lighting</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_ZBUFFER - Enable z buffer</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_ZWRITE_ENABLE - Can write as well as read z buffer</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_BACK_FACE_CULLING - Cull polygons facing away</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_BILINEAR_FILTER - Enable bilinear filtering</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_TRILINEAR_FILTER - Enable trilinear filtering</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_ANISOTROPIC_FILTER - Reduce blur in distant textures</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_FOG_ENABLE - Enable fogging in the distance</td>
</tr>
<tr>
<td></td>
<td>$IRR_EMF_NORMALIZE_NORMALS - Use when scaling dynamically lighted models</td>
</tr>
<tr>
<td>$i_Flag</td>
<td>Switches selected property on or off:</td>
</tr>
<tr>
<td></td>
<td>$IRR_ON</td>
</tr>
<tr>
<td></td>
<td>$IRR_OFF</td>
</tr>
</tbody>
</table>

**Return Value**
Success: True
Failure: False

Remarks
None.

Related
_IrrSetNodeMaterialTexture, _IrrSetNodeMaterialType

Example

```plaintext
#include "au3Irrlicht2.au3"

_IrrStart()

local $camera = _IrrAddCamera(2,2,2,0,0,0)

local $mesh = _IrrGetMesh("\media\capsuleX.obj")
local $sceneNode = _IrrAddMeshToScene( $mesh )

local $texture = _IrrGetTexture("\media\default_texture.png")
_IrrSetNodeMaterialTexture( $sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag( $sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

WHILE _IrrRunning()
   _IrrBeginScene(50, 50, 50)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
```
_IrrSetNodeMaterialType

Set the way that materials are applied to a scene node

```
#Include <au3Irrlicht2.au3>
_IrrSetNodeMaterialType($h_Node, $i_Type)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene node</td>
</tr>
<tr>
<td>$i_Type</td>
<td>Material property to apply to the scene node (see table in remarks)</td>
</tr>
</tbody>
</table>

**Return Value**

- success: True
- failure: False

**Remarks**

**Valid material properties:**

<table>
<thead>
<tr>
<th>Material Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$IRR_EMT_SOLID</td>
<td>Standard solid rendering uses one texture</td>
</tr>
<tr>
<td>$IRR_EMT_SOLID_2_LAYER</td>
<td>2 blended textures using vertex alpha value</td>
</tr>
<tr>
<td>$IRR_EMT_LIGHTMAP</td>
<td>2 textures: 0=color, 1=lighting level and ignores vertex lighting</td>
</tr>
<tr>
<td>$IRR_EMT_LIGHTMAP_ADD</td>
<td>... as above but adds levels instead of modulating between them</td>
</tr>
<tr>
<td>$IRR_EMT_LIGHTMAP_M2</td>
<td>... as above but color levels are multiplied by 2 for brightening</td>
</tr>
<tr>
<td>$IRR_EMT_LIGHTMAP_M4</td>
<td>... as above but color levels are multiplied by 4 for brightening</td>
</tr>
<tr>
<td>$IRR_EMT_LIGHTMAP_LIGHTING</td>
<td>2 textures: 0=color 1=lighting level button supports dynamic lighting</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>$IRR_EMT_LIGHTMAP_LIGHTING_M2</td>
<td>... as above but color levels are multiplied by 2 for brightening</td>
</tr>
<tr>
<td>$IRR_EMT_LIGHTMAP_LIGHTING_M4</td>
<td>... as above but color levels are multiplied by 4 for brightening</td>
</tr>
<tr>
<td>$IRR_EMT_DETAIL_MAP</td>
<td>2 blended textures: the first is a color map the second at a different scale adds and subtracts from the color to add detail</td>
</tr>
<tr>
<td>$IRR_EMT_SPHERE_MAP</td>
<td>makes the material appear reflective</td>
</tr>
<tr>
<td>$IRR_EMT_REFLECTION_2_LAYER</td>
<td>a reflective material blended with a color texture</td>
</tr>
<tr>
<td>$IRR_EMT_TRANSPARENT_ADD_COLOR</td>
<td>a transparency effect that simply adds a color texture to the background. the darker the color the more transparent it is.</td>
</tr>
<tr>
<td>$IRR_EMT_TRANSPARENT_ALPHA_CHANNEL</td>
<td>a transparency effect that uses the color texture's alpha as a transparency level</td>
</tr>
<tr>
<td>$IRR_EMT_TRANSPARENT_ALPHA_CHANNEL_REF</td>
<td>a transparency effect that uses the color texture's alpha, the pixel is only drawn if the alpha is &gt; 127. this is a fast effect that does not blur edges and is ideal for leaves &amp; grass etc.</td>
</tr>
<tr>
<td>$IRR_EMT_TRANSPARENT_VERTEX_ALPHA</td>
<td>a transparency effect that uses the vertex alpha value</td>
</tr>
<tr>
<td>$IRR_EMT_TRANSPARENT_REFLECTION_2_LAYER</td>
<td>a transparent &amp; reflecting effect. the first texture is a reflection map, the second a color map. transparency is from vertex alpha</td>
</tr>
<tr>
<td>$IRR_EMT_NORMAL_MAP_SOLID</td>
<td>A solid normal map renderer. First texture is color, second is normal map. Only used as added to the irrscene with IrrAddStaticMeshForNormalMappingToScene. Only supports nearest two lights. Requires vertex and pixel shaders 1.1</td>
</tr>
<tr>
<td>$IRR_EMT_NORMAL_MAP_TRANSPARENT_ADD_COLOR</td>
<td>... as above only with a transparency effect that simply adds the color to the background. the darker the color the more transparent it is.</td>
</tr>
</tbody>
</table>
$IRR_EMT_NORMAL_MAP_TRANSPARENT_VERTEX_ALPHA
... as above only with a transparency effect that uses the vertex alpha value.

$IRR_EMT_PARALLAX_MAP_SOLID
similar to the solid normal map but more realistic providing virtual displacement of the surface. Uses the alpha channel of the normal map for the height field displacement. Requires vertex shader 1.1 and pixel shader 1.4.

$IRR_EMT_PARALLAX_MAP_TRANSPARENT_ADD_COLOR
... as above only with a transparency effect that simply adds the color to the background. The darker the color the more transparent it is.

$IRR_EMT_PARALLAX_MAP_TRANSPARENT_VERTEX_ALPHA
... as above only with a transparency effect that uses the vertex alpha value.

Related

_ IrrSetNodeMaterialTexture, _IrrSetNodeMaterialFlag

Example

```plaintext
#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeCamera = _IrrAddCamera(2,2,2,0,0,0)
local $nodeCube = _IrrAddCubeSceneNode(2); _IrrAddMeshToScene( $mesh )
_IrrSetNodeMaterialTexture( $nodeCube, _IrrGetTexture("\media\au3irr2_logo.jpg"), 0 )

_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodeMaterialType( $nodeCube, $IRR_EMT_SPHERE_MAP )

_IrrAddRotationAnimator( $nodeCube, 0.1, 0.1, 0.1 )
```
WHILE _IrrRunning()
  _IrrBeginScene(50, 50, 50)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
au3Irr2 Function Reference

_IrrSetNodePosition

Moves the node to the specified position

```
#include <au3Irrlicht2.au3>
_IrrSetNodePosition($h_Node, $f_X, $f_Y, $f_Z)
```

Parameters

<table>
<thead>
<tr>
<th>$h_Node</th>
<th>Handle of a scene node</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_X, $f_Y, $f_Z</td>
<td>X, Y, Z values of new position</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False

Remarks

None

Related

_IrrGetNodePosition, _IrrGetNodeRotation, _IrrSetNodeRotation, _IrrGetNodeAbsolutePosition

Example

```
#include "au3Irrlicht2.au3"
```
_IrrStart()

local $nodeCube = _IrrAddCubeSceneNode(5)
_IrrSetNodeMaterialTexture( $nodeCube,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrSetNodePosition($nodeCube, 5, 5, 5)
local $nodeCamera = _IrrAddCamera(0, 0, 0, 5, 5, 5)

WHILE _IrrRunning()
    _IrrBeginScene(50, 50, 50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
**au3Irr2 Function Reference**

_**IrrSetNodeRotation**_

Rotate a node to the specified orientation through its X, Y and Z axis

```c
#include <au3Irrlicht2.au3>
_IrrSetNodeRotation($h_Node, $f_X, $f_Y, $f_Z)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene node</td>
</tr>
<tr>
<td>$f_X, $f_Y, $f_Z</td>
<td>Values of rotation along X, Y, Z axes in degrees (0-360)</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True
Failure: False

**Remarks**

None

**Related**


**Example**

```c
#include "au3Irrlicht2.au3"
```
_IrrStart()

local $nodeCamera = _IrrAddCamera(10, 0, 0, 0, 0)
local $nodeCube = _IrrAddCubeSceneNode(5)
_IrrSetNodeMaterialTexture( $nodeCube,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrSetNodeRotation($nodeCube, 90, 45, 45 )

WHILE _IrrRunning()
  _IrrBeginScene(50, 50, 50)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
_IrrSetNodeScale

Sets the scale of the scene node

```c
#Include <au3Irrlicht2.au3>
_IrrSetNodeScale($h_Node, $f_X, $f_Y, $f_Z)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene node</td>
</tr>
<tr>
<td>$f_X, $f_Y, $f_Z</td>
<td>Scaling factors for X, Y, Z axes.</td>
</tr>
</tbody>
</table>

### Return Value

Success: True  
Failure: False

### Remarks

None

### Related

- _IrrSetNodePosition
- _IrrSetNodeRotation

### Example

```c
#include "au3Irrlicht2.au3"
```
local $nodeCamera = _IrrAddCamera(5,10,10, 0,0,0 )
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrSetNodeScale($nodeTest, 0.5, 1, 1.5)

WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 0)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
_IrrDebugDataVisible

Displays debugging data around a node (typically the bounding box around edges of the node).

```au3
#include <au3Irrlicht2.au3>
_IrrDebugDataVisible($h_Node, $i_Visible = $EDS_BBOX)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a scene node.</td>
</tr>
<tr>
<td>$i_Visible</td>
<td>Type of debugging information (not all of them are supported on all node types): $EDS_OFF: No Debugging $EDS_BBOX: Bounding Box $EDS_NORMALS: Normals $EDS_SKELETON: Skeleton $EDS_MESH_WIRE_OVERLAY: Wireframe $EDS_HALF_TRANSPARENCY: Transparency $EDS_BBOX_BUFFERS: Bounding Box Buffers $EDS_FULL: Everything</td>
</tr>
</tbody>
</table>

### Return Value
None.

### Remarks
None.

### Related
Example

```c
#include "au3Irrlicht2.au3"

_IrrStart()

local $camera = _IrrAddFPSCamera()
local $mesh = _IrrGetMesh("\media\capsuleX.obj")
local $nodeTest = _IrrAddMeshToScene($mesh)
_IrrSetNodeMaterialTexture($nodeTest, _IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag($nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF)
_IrrSetNodePosition($nodeTest, 0, 0, 8)
_IrrDebugDataVisible($nodeTest, $EDS_FULL)

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
_IrrGetPosition

Returns array with position coordinates of a scene node.

```cpp
#include <au3Irrlicht2.au3>
_IrrGetPosition($h_Node)
```

**Parameters**

$\texttt{h\_Node}$ | Handle of a scene node

**Return Value**

*success:* 0-based array with three elements for X, Y, Z coordinates.
*failure:* False

**Remarks**

None.

**Related**

_IrrSetNodePosition, _IrrGetNodeRotation, _IrrSetNodeRotation, _IrrGetPositionAbsolutePosition

**Example**

```cpp
#include "au3Irrlicht2.au3"
_IrrStart()
```
local $nodeTest = _IrrAddTestSceneNode()
local $nodeCamera = _IrrAddFPSCamera()
_IrrSetNodePosition($nodeCamera, 0, 0, -50)

local $aVector3df

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 50)
    _IrrDrawScene()

    $aVector3df = _IrrGetNodePosition($nodeCamera)
    _IrrSetWindowCaption("Camera position (x/y/z): " & int($aVector3df[0]) & " / " & int($aVector3df[1]) & " / " & int($aVector3df[2]))

    _IrrEndScene()
WEND

_IrrStop()
_IrrGetNodeAbsolutePosition

Get the absolute position of the node in the scene.

#include <au3Irrlicht2.au3>
_IrrGetNodeAbsolutePosition($h_Node)

Parameters

$h_Node Handle of a scene node

Return Value

success: 0-based array with three elements for X, Y, Z coordinates.
failure: False

Remarks

The absolute position includes the position changes of all of the nodes parents too.

Related

_IrrGetNodePosition, _IrrSetNodePosition, _IrrGetNodeRotation, _IrrSetNodeRotation
_IrrGetNodeRotation

Returns array with rotation values of a scene node.

```
#include <au3Irrlicht2.au3>
_IrrGetNodeRotation($h_Node)
```

Parameters

$\textit{h\_Node}$ | Handle of a scene node

Return Value

success: 0-based array with three elements for X, Y, Z rotation.
failure: False

Remarks

None.

Related

_\texttt{IrrGetNodePosition}, _\texttt{IrrSetNodePosition}, _\texttt{IrrSetNodeRotation}, _\texttt{IrrGetNodeAbsolutePosition}

Example

```
#include "au3Irrlicht2.au3"

_IrrStart()
```
local $nodeTest = _IrrAddTestSceneNode()
local $nodeCamera = _IrrAddFPSCamera()
_IrrSetNodePosition($nodeCamera, 0, 0, -50)

local $aVector3df

WHILE _IrrRunning()
  _IrrBeginScene( 0, 0, 50 )
  _IrrDrawScene()

  $aVector3df = _IrrGetNodeRotation($nodeCamera)
  _IrrSetWindowCaption("Camera rotation (x/y/z): " & _
                         int($aVector3df[0]) & " / " & int($aVector3df[1]) & " / " &
                         int($aVector3df[2]))

  _IrrEndScene()
WEND

_IrrStop()
au3Irr2 Function Reference

_IrrGetJointNode

[todo]

#include <au3Irrlicht2.au3>
_IrrGetJointNode($h_Node, $s_Joint)

### Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Return Value

- **success**: explanation
- **failure**: explanation
  - moreExplanationIndented

### Remarks

[todo]

### Related

[todo: functionName, functionName]
_IrrAddChildToParent

[todo]

#include <au3Irrlicht2.au3>
_IrrAddChildToParent($h_ChildNode, $h_ParentNode)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetNodeFirstChild

[todo]

```
#include <au3Irrlicht2.au3>
_IrrGetNodeFirstChild($h_Node, ByRef $h_Position)
```

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

Return Value

[success]: explanation
[failure]: explanation
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetNodeNextChild

Get the next child node of this node, returns 0 if there is no child.

```c
#include <au3Irrlicht2.au3>
_IrrGetNodeNextChild($h_Node, ByRef $h_Position)
```

**Parameters**

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

**Return Value**

[success]: explanation
[failure]: explanation
  [moreExplanationIndented]

**Remarks**

todo

**Related**

todo: functionName, functionName
_IrrIsNodeLastChild

[todo]

```c
#include <au3Irrlicht2.au3>
_IrrIsNodeLastChild($h_Node, ByRef $h_Position)
```

**Parameters**

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

**Return Value**

[success]: [explanation]
[failure]: [explanation]

[moreExplanationIndented]

**Remarks**

[todo]

**Related**

[todo: functionName, functionName]
_IrrAddNodeShadow

Adds shadows to a node that are cast across other nodes in the scene.

```
#include <au3Irrlicht2.au3>
_IrrAddNodeShadow($h_Node, $h_mesh = 0)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Node</td>
<td>Handle of a node in the scene</td>
</tr>
<tr>
<td>$h_mesh</td>
<td>[optional] Handle of mesh casting the shadow (0 uses mesh of $h_Node, see remarks)</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True
Failure: False

**Remarks**

_IrrAddShadows will only work when shadowing has been activated with _IrrStart or _IrrStartAdvanced.
You should analyse the performance of your scene carefully when using this function as it can have a significant effect on your frame rate.
You can supply a different mesh to the one used to display the node, this shadow mesh could be a much lower resolution than that used for your model thereby improving performance.

_IrrAddNodeShadow does NOT work with buildin nodes types _IrrAddTestSceneNode, _IrrAddCubeSceneNode, and _IrrAddSphereSceneNode. If you need a cube or a sphere casting shadows, create a node from a loaded cube or sphere mesh as workaround.
Related
_IrrStart, _IrrStartAdvanced, _IrrSetShadowColor, _IrrAddLight,
_IrrSetAmbientLight

Example

```plaintext
#include "au3Irrlicht2.au3"

_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _
    $IRR_WINDOWED, $IRR_SHADOWS, $IRR_IGNORE_EVENTS, $IRR_VERTICAL_SYNC_ON )

local $texture = _IrrGetTexture("\media\au3irr2_logo.jpg")

local $meshReceive = _IrrAddHillPlaneMesh( "Plane", 8, 8, 1, 1)
local $nodeReceive = _IrrAddMeshToScene( $meshReceive )
_IrrSetNodeMaterialTexture( $nodeReceive, $texture, 0)

local $meshCast = _IrrGetMesh("\media\sphere.obj")
local $nodeCast = _IrrAddMeshToScene($meshCast)
_IrrSetNodeMaterialTexture( $nodeCast, $texture, 0)
_IrrAddFlyCircleAnimator($nodeCast, 1, 3, 1, 3, 0.0005)

local $nodeLight = _IrrAddLight($IRR_NO_PARENT, 0,50,0, 0.9,0.3,0.3,
    1000.0)
_IrrAddNodeShadow($nodeCast)

local $nodeCamera = _IrrAddCamera(-5,5,-6, 0,0,0 )

WHILE _IrrRunning()
    _IrrBeginScene(100, 100, 100)
    _IrrDrawScene()
```
_IrrEndScene()
WEND

_IrrStop()
Copy to Clipboard
au3Irr2 Function Reference

_IrrSetNodeVisibility

[.todo]

#include <au3Irrlicht2.au3>
_IrrSetNodeVisibility($h_Node, $i_Visible)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrRemoveNode

Removes a node from the scene deleting it.

#include <au3Irrlicht2.au3>
_IrrRemoveNode($h_Node)

Parameters

$h_Node | Handle of a scene node

Return Value

Success: True
Failure: False

Remarks

None

Related

None.

Example

#include "au3Irrlicht2.au3"

_IrrStart()
local $camera = _IrrAddCamera(2,2,2,0,0,0)

local $mesh = _IrrGetMesh("media\capsuleX.obj")
local $sceneNode = _IrrAddMeshToScene($mesh)

local $texture = _IrrGetTexture("media\default_texture.png")
_IrrSetNodeMaterialTexture($sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag($sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)

local $time = TimerInit()
WHILE _IrrRunning()
    _IrrBeginScene(50, 50, 50)
    _IrrDrawScene()
    _IrrEndScene()

    if $sceneNode <> 0 AND TimerDiff($time) > 3000 then
        _IrrRemoveNode($sceneNode)
        _IrrRemoveTexture($texture) ; no longer needed
        $sceneNode = 0
    EndIf

WEND

_IrrStop()
au3Irr2 Function Reference

_IrrRemoveAllNodes

[todo]

#include <au3Irrlicht2.au3>
_IrrRemoveAllNodes()  

Parameters

[explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetNodeID

[todo]

#include <au3Irrlicht2.au3>
_IrrSetNodeID($h_Node, $i_ID)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
__CreateParticleSettings

Helper function: Creates particle settings as required from _IrrAddParticleEmitter.

#include <au3Irrlicht2.au3>
__CreateParticleSettings($minBoxX, $minBoxY, $minBoxZ, $maxBoxX, $maxBoxY, $maxBoxZ, $directionX, $directionY, $directionZ, $minParticlesSecond, $maxParticlesSecond, $minStartColorR, $minStartColorG, $minStartColorB, $maxStartColorR, $maxStartColorG, $maxStartColorB, $minLifetime, $maxLifetime, $minStartSizeX, $minStartSizeY, $maxStartSizeX, $maxStartSizeY, $maxAngleDegrees)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$minBoxX, $minBoxY, $minBoxZ</td>
<td>Minimal positions of a a box in space inside which the position of a particle is randomly created.</td>
</tr>
<tr>
<td>$maxBoxX, $maxBoxY, $maxBoxZ</td>
<td>Maximal positions of a a box in space inside which the position of a particle is randomly created.</td>
</tr>
<tr>
<td>$directionX, $directionY, $directionZ</td>
<td>Define a direction into which the particles will be ejected as the animation plays.</td>
</tr>
<tr>
<td>$minParticlesSecond, $maxParticlesSecond</td>
<td>A range defining the minimum and maximum number of particles that will be created each second.</td>
</tr>
<tr>
<td>$minStartColorR, $minStartColorG, $minStartColorB, $maxStartColorR, $maxStartColorG, $maxStartColorB</td>
<td>Although particles can be textured by texturing the particle system node, these can be used to apply a range that tints the color of the particles.</td>
</tr>
</tbody>
</table>
$minLifetime, $maxLifetime

How long the particle will live, long lifespans can create very large numbers of particles.

$minStartSizeX, $minStartSizeY, $maxStartSizeX, $maxStartSizeY

The minimum and maximum start sizes for the particles.

$maxAngleDegrees

The maximum number of degrees that the ejected particles will deviate from the defined direction.

**Return Value**

None.

**Remarks**

None

**Related**

_`IrrAddParticleSystemToScene`, `_IrrAddParticleEmitter`

**Example**

```plaintext
#include "au3Irrlicht2.au3"

_IrrStart()

local $particleSystem =
_IrrAddParticleSystemToScene($IRR_NO_EMITTER, 0, 0, 0, 0, 0, 0, 6, 3, 6)

local $SmokeEmitter = __CreateParticleSettings( -7.0, 0, -7.0, 7.0, 1.0, 7.0, 0, 0.04, 0, 80, 100, _
```
_IrrAddParticleEmitter($particleSystem, $SmokeEmitter)

local $ParticleTexture = _IrrGetTexture( "./media/ParticleGrey.bmp" )
_IrrSetNodeMaterialTexture( $particleSystem, $ParticleTexture, 0 )
_IrrSetNodeMaterialFlag( $particleSystem, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodeMaterialType( $particleSystem, $IRR_EMT_TRANSPARENT_ADD_COLOR )

local $nodeCamera = _IrrAddCamera( 80,0,0, 20,40,0 )

WHILE _IrrRunning()
  _IrrBeginScene( 0,0,50 )
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
_IrrAddParticleEmitter

Adds a particle emitter to a particle system.

```
#include <au3Irrlicht2.au3>
_IrrAddParticleEmitter($h_ParticleSystem, $a_Settings)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_ParticleSystem</td>
<td>Handle of particle system the emitter shall be attached to.</td>
</tr>
<tr>
<td>$a_Settings</td>
<td>Array with particle emitter settings created with __CreateParticleSettings.</td>
</tr>
</tbody>
</table>

**Return Value**

success: Handle of the created particle emitter

failure: False

**Remarks**

The emitter creates particles and controls how they move and when they are to be removed.

**Related**

_IrrAddParticleSystemToScene, __CreateParticleSettings

**Example**

```
```lua
#include "auIrrlicht2.au3"

_IrrStart()

local $particleSystem = 
_IrrAddParticleSystemToScene($IRR_NO_EMITTER, 0, 0, 0, 0, 0, 0, 6, 3, 6)

local $SmokeEmitter = __CreateParticleSettings( -7.0, 0, -7.0, 7.0, 1.0, 7.0, 0, 0.04, 0, 80, 100, _
                                                255, 255, 255, 255, 255, _
                                                800, 2000, 15.0, 15.0, 15.0, 15.0, 15.0, 15.0 )

_IrrAddParticleEmitter($particleSystem, $SmokeEmitter)

local $ParticleTexture = _IrrGetTexture( "./media/ParticleGrey.bmp" )
_IrrSetNodeMaterialTexture( $particleSystem, $ParticleTexture, 0 )
_IrrSetNodeMaterialFlag( $particleSystem, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodeMaterialType( $particleSystem, $IRR_EMT_TRANSPARENT_ADD_COLOR )

local $nodeCamera = _IrrAddCamera( 80,0,0, 20,40,0 )

WHILE _IrrRunning()
    _IrrBeginScene( 0,0,50 )
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
au3Irr2 Function Reference

_IrrAddAnimatedMeshSceneNodeEmitter

(todo)

#include <au3Irrlicht2.au3>
_IrrAddAnimatedMeshSceneNodeEmitter($h_ParticleSystem, $h_Node, $b_UseNormals, $f_NormalModifier, $b_FromAllVertices, $a_Settings)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
  [moreExplanationIndented]

Remarks

(todo)

Related

(todo: functionName, functionName)
_IrrAddRotationAffector

Adds a an affector to a particle system rotating the particles.

```c
#include <au3Irrlicht2.au3>
_IrrAddRotationAffector($h_ParticleSystem, $f_SpeedX, $f_SpeedY, $f_SpeedZ, $f_PivotX, $f_PivotY, $f_pivotZ)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_ParticleSystem</td>
<td>Handle of the particle system the created affector is attached to.</td>
</tr>
<tr>
<td>$f_SpeedX,</td>
<td>Set the speed in degrees per second in all 3 dimensions.</td>
</tr>
<tr>
<td>$f_SpeedY,</td>
<td></td>
</tr>
<tr>
<td>$f_SpeedZ</td>
<td></td>
</tr>
<tr>
<td>$f_PivotX,</td>
<td>Set the point that particles will rotate around.</td>
</tr>
<tr>
<td>$f_PivotY,</td>
<td></td>
</tr>
<tr>
<td>$f_pivotZ</td>
<td></td>
</tr>
</tbody>
</table>

### Return Value

- success: Handle of the created affector.
- failure: false

### Remarks

This affector modifies the positions of the particles and attracts them to a specified point at a specified speed per second.
Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

; add particle system with default emitter to the irrlicht scene manager
local $particleSystem = _IrrAddParticleSystemToScene($IRR_DEFAULT_EMITTER)
local $ParticleTexture = _IrrGetTexture("./media/ParticleGrey.bmp")

; setup affector rotating particles around all three axes with particles attracted to 30/0/0:
_IrrAddRotationAffector( $particleSystem, 50.0, -120.0, 50.0, 30.0, 0.0, 0.0 )

_IrrSetNodeMaterialTexture( $particleSystem, $ParticleTexture, 0 )
_IrrSetNodeMaterialFlag( $particleSystem, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodeMaterialType( $particleSystem, $IRR_EMT_TRANSPARENT_ADD_COLOR )

local $nodeCamera = _IrrAddCamera( 80,0,0, 0,50,50)

WHILE _IrrRunning()
    _IrrBeginScene( 0,0,50 )
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
_IrrAddFadeOutParticleAffector

Adds a fade out affector to a particle system gradually fading particles out so they are invisible when they are deleted.

```
#include <au3Irrlicht2.au3>
_IrrAddFadeOutParticleAffector($h_ParticleSystem, $i_FadeFactor, $i_Red, $i_Green, $i_Blue)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_ParticleSystem</td>
<td>Handle of the particle system the created affector is attached to.</td>
</tr>
<tr>
<td>$i_FadeFactor</td>
<td>Milliseconds the fade out effect will take place.</td>
</tr>
<tr>
<td>$i_Red, $i_Green,</td>
<td>Values of the colour the particles are faded to (0-255)</td>
</tr>
<tr>
<td>$i_Blue</td>
<td></td>
</tr>
</tbody>
</table>

**Return Value**

- success: Handle of the created affector.
- failure: false

**Remarks**

The fade out affector fades the particles out as they come to the end of their lifespan and stops them 'popping' out of existence. This creates a convincing effect for fire and smoke in particular.

**Related**

_ IrrAddParticleSystemToScene,  _IrrSetParticleAffectorEnable,
Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

; add particle system with default emitter to the irrlicht scene manager
local $particleSystem = _IrrAddParticleSystemToScene($IRR_DEFAULT_EMITTER)
local $ParticleTexture = _IrrGetTexture("./media/ParticleGrey.bmp")

; setup this affector for a simple flashing-out effect
_IrrAddFadeOutParticleAffecter($particleSystem, 250, 255, 255, 0)

_IrrSetNodeMaterialTexture($particleSystem, $ParticleTexture, 0)
_IrrSetNodeMaterialFlag($particleSystem, $IRR_EMF_LIGHTING, $IRR_OFF)
_IrrSetNodeMaterialType($particleSystem, $IRR_EMT_TRANSPARENT_ADD_COLOR)

local $nodeCamera = _IrrAddCamera(80,0,0,20,40,0)

WHILE _IrrRunning()
    _IrrBeginScene(0,0,50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
_IrrAddGravityParticleAffector

Adds a gravity affector to a particle system gradually pulling the particles in the direction of the effect.

```au3
#include <au3Irrlicht2.au3>
_IrrAddGravityParticleAffector($h_ParticleSystem, $f_X, $f_Y, $f_Z, $i_TimeForceLost = 1000)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_ParticleSystem</td>
<td>Handle of the particle system the created affector is attached to.</td>
</tr>
<tr>
<td>$f_X, $f_Y, $f_Z</td>
<td>Set the direction and force of gravity in all 3 dimensions.</td>
</tr>
<tr>
<td>$i_TimeForceLost</td>
<td>[optional] Set the time in milliseconds when the gravity force is totally lost. At that point the particle does not move any more.</td>
</tr>
</tbody>
</table>

**Return Value**

success: Handle of the created affector.
failure: false

**Remarks**

The gravity affector is adding a small amount of velocity to the particles each frame. Although its called a gravity affector it can be used to push the particles in any direction so you can have drifting smoke bubbling fountains, to make a wind effect and have the particles drift off to the side, etc.
Related

_IrrAddParticleSystemToScene, _IrrSetParticleAffectEnable, _IrrRemoveAffectors

Example

```
#include "au3Irrlicht2.au"

_IrrStart()

; add particle system with default emitter to the irrlicht scene manager
local $particleSystem = _IrrAddParticleSystemToScene($IRR_DEFAULT_EMITTER)
local $ParticleTexture = _IrrGetTexture( "./media/ParticleGrey.bmp" )

; setup this affector to let the particles drift off
_IrrAddGravityParticleAffectector( $particleSystem, -0.4, 0, 0.8, 3000)

_IrrSetNodeMaterialTexture( $particleSystem, $ParticleTexture, 0 )
_IrrSetNodeMaterialFlag( $particleSystem, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodeMaterialType ( $particleSystem, $IRR_EMT_TRANSPARENT_ADD_COLOR )

local $nodeCamera = _IrrAddCamera( 80,0,0, 0,40,40 )

WHILE _IrrRunning()
    _IrrBeginScene( 0,0,50 )
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```
_IrrAddParticleAttractionAfferctor

Adds an affector to a particle system attracting particles to a specified point at a specified speed.

```
#include <au3Irrlicht2.au3>
_IrrAddParticleAttractionAfferctor($h_ParticleSystem, $f_X, $f_Y, $f_Z, $f_Speed = 1, $i_Attract = $IRR_ATTRACT, $b_AffectX = true, $b_AffectY = true, $b_AffectZ = true)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_ParticleSystem</td>
<td>Handle of the particle system the created affector is attached to.</td>
</tr>
<tr>
<td>$f_X, $f_Y, $f_Z</td>
<td>Set the point that particles will attract to.</td>
</tr>
<tr>
<td>$f_Speed</td>
<td>[optional] Speed in units per second, to attract to the specified point.</td>
</tr>
<tr>
<td>$i_Attract</td>
<td>[optional] Set whether or not the particles are attracting or detracting. Values are: $IRR_ATTRACT - particles are attracting. IRR_REPEL - particles are detracting.</td>
</tr>
<tr>
<td>$b_AffectX, $b_AffectY, $b_AffectZ</td>
<td>[optional] Set whether or not this will affect particles in the X, Y, Z direction.</td>
</tr>
</tbody>
</table>

### Return Value

- **success:** Handle of the created affector.
- **failure:** false

### Remarks
Related

_IrrAddParticleSystemToScene, _IrrSetParticleAffectorEnable, _IrrRemoveAffectors

Example

#include "au3Irrlicht2.au3"

_IrrStart()

; add particle system with default emitter to the irrlicht scene manager
local $particleSystem =
_IrrAddParticleSystemToScene($IRR_DEFAULT_EMITTER)
local $ParticleTexture = _IrrGetTexture("./media/ParticleGrey.bmp")

; setup this affector to detract particles downwards
_IrrAddParticleAttractionAffecter( $particleSystem, 0,100.0,0.0, 80.0, $IRR_REPEL )

_IrrSetNodeMaterialTexture( $particleSystem, $ParticleTexture, 0 )
_IrrSetNodeMaterialFlag( $particleSystem, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodeMaterialType( $particleSystem,
IRR_EMT_TRANSPARENT_ADD_COLOR )

local $nodeCamera = _IrrAddCamera( 80,0,0, 0,-20,0)

WHILE _IrrRunning()
    _IrrBeginScene( 0,0,50 )
    _IrrDrawScene()
    _IrrEndScene()
WEND
_IrrStop()
au3Irr2 Function Reference

_IrrAddStopParticleAffecter

[todo]

#include <au3Irrlicht2.au3>
,IrrAddStopParticleAffecter($h_ParticleSystem, $i_Time, $h_Emitter)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
#Include <au3Irrlicht2.au3>
_IrrAddParticlePushAffect($h_ParticleSystem, $f_X, $f_Y, $f_Z, $f_SpeedX, $f_SpeedY, $f_SpeedZ, $f_Far, $f_Near, $f_Column, $i_Distant)

**Parameters**

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

**Return Value**

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

**Remarks**

todo

**Related**
todo: functionName, functionName
au3Irr2 Function Reference

_IrrAddColorMorphAffecter

[.todo]

#include <au3Irrlicht2.au3>
_IrrAddColorMorphAffecter($h_ParticleSystem, $a_ParticleColors, $a_ParticleTimes, $b_Smooth)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
#Include <au3Irrlicht2.au3>
_IrrAddSplineAffector($h_ParticleSystem, $tVectors, $f_Speed, $f_Tightness, $f_Attraction, $b_DeleteAtEnd)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

todo

Related

todo: functionName, functionName
_IrrRemoveAffectors

Removes all affectors from a particle system.

```
#include <au3Irrlicht2.au3>
_IrrRemoveAffectors($h_ParticleSystem)
```

**Parameters**

$h_ParticleSystem | Handle of an Irrlicht particle system.

**Return Value**

success: true
failure: false

**Remarks**

You might use this if you want to change the direction or strength of the wind for example.

**Related**

_IrrAddParticleSystemToScene, _IrrSetParticleAffecterEnable

**Example**

```
#include "au3Irrlicht2.au3"

_IrrStart()
```
; add particle system with default emitter to the irrlicht scene manager
local $particleSystem = _IrrAddParticleSystemToScene($IRR_DEFAULT_EMITTER)
local $ParticleTexture = _IrrGetTexture( "/media/ParticleGrey.bmp" )

; add two affectors to the particle system:
_IrrAddRotationAffecttor ( $particleSystem, 50.0, -120.0, 50.0, 30.0,0.0,0.0 )
_IrrAddFadeOutParticleAffecttor( $particleSystem, 100, 255, 0, 0 )

_IrrSetNodeMaterialTexture( $particleSystem, $ParticleTexture, 0 )
_IrrSetNodeMaterialFlag( $particleSystem, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodeMaterialType ( $particleSystem, $IRR_EMT_TRANSPARENT_ADD_COLOR )

local $nodeCamera = _IrrAddCamera( 80,0,0, 0,50,50)

local $time = TimerInit()
WHILE _IrrRunning()

    if TimerDiff($time) > 8000 then _IrrRemoveAffecttors($particleSystem)

    _IrrBeginScene( 0,0,50 )
    _IrrDrawScene()
    _IrrEndScene()

WEND

_IrrStop()
_IrrSetParticleEmitterMinParticlesPerSecond

[rename]

#include <au3Irrlicht2.au3>
_IrrSetParticleEmitterMinParticlesPerSecond($h_ParticleEmitter, $i_Min)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[rename]

Related

[rename: functionName, functionName]
_IrrSetParticleEmitterMaxParticlesPerSecond

[todo]

#include <au3Irrlicht2.au3>
_IrrSetParticleEmitterMaxParticlesPerSecond($h_ParticleEmitter, $i_Max)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[TODO: functionName, functionName]
au3Irr2 Function Reference

_IrrSetParticleEmitterMinStartColor

[ todo ]

#Include <au3Irrlicht2.au3>
_IrrSetParticleEmitterMinStartColor($h_ParticleEmitter, $i_Red, $i_Green, $i_Blue)

Parameters

[ param1 ] [ explanation ]
[ moreTextForParam1 ]

Return Value

[ success ]: [ explanation ]
[ failure ]: [ explanation ]
[ moreExplanationIndented ]

Remarks

[ todo ]

Related

[ todo: functionName, functionName ]
### _IrrSetParticleEmitterMaxStartColor

[todo]

```cpp
#include <au3Irrlicht2.au3>
_IrrSetParticleEmitterMaxStartColor($h_ParticleEmitter, $i_Red, $i_Green, $i_Blue)
```

#### Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

#### Return Value

- [success]: explanation
- [failure]: explanation
  - moreExplanationIndented

#### Remarks

[todo]

#### Related

[todo: functionName, functionName]
Function Reference

__IrrSetParticleAffectoreEnable

Enables or disables an Irrlicht affector.

```
#include <au3Irrlicht2.au3>
_IrrSetParticleAffectoreEnable($h_ParticleAffectore, $b_Enabled)
```

Parameters

<table>
<thead>
<tr>
<th>$h_ParticleAffectore</th>
<th>Handle of an Irrlicht particle affector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Enabled</td>
<td>$IRR_ON (or true) enables the affector, $IRR_OFF (or false) disables it.</td>
</tr>
</tbody>
</table>

Return Value

success: true
failure: false

Remarks

None

Related

__IrrAddParticleSystemToScene__, __IrrRemoveAffectors__

Example

```
#include "au3Irrlicht2.au3"
```
_IrrStart()

; add particle system with default emitter to the irrlicht scene manager
local $particleSystem = _IrrAddParticleSystemToScene($IRR_DEFAULT_EMITTER)
local $ParticleTexture = _IrrGetTexture("./media/ParticleGrey.bmp")

local $affector = _IrrAddRotationAffector($particleSystem, 50.0, -120.0, 50.0, 30.0, 0.0, 0.0)

_IrrSetNodeMaterialTexture($particleSystem, $ParticleTexture, 0)
_IrrSetNodeMaterialFlag($particleSystem, $IRR_EMF_LIGHTING, $IRR_OFF)
_IrrSetNodeMaterialType($particleSystem, $IRR_EMT_TRANSPARENT_ADD_COLOR)

local $nodeCamera = _IrrAddCamera(80, 0, 0, 0, 50, 50)

local $time = TimerInit()
local $enabled = true
WHILE _IrrRunning()

; enable/disable affector every 2 seconds:
if TimerDiff($time) > 2000 then
  $enabled = NOT $enabled
  _IrrSetParticleAffectorEnable($affector, $enabled)
  $time = TimerInit()
EndIf

_IrrBeginScene(0, 0, 50)
_IrrDrawScene()
_IrrEndScene()

WEND

_IrrStop()
**au3Irr2 Function Reference**

_**_IrrSetFadeOutParticleAffectorTime

[todo]

```c
#Include <au3Irrlicht2.au3>
_IrrSetFadeOutParticleAffectorTime($h_ParticleAffecto, $f_Time)
```

**Parameters**

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

**Return Value**

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

**Remarks**

[todo]

**Related**

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetGravityParticleAffectorDirection

[TODO]

#include <au3Irrlicht2.au3>
_IrrSetGravityParticleAffectorDirection($h_ParticleAffector, $f_X, $f_Y, $f_Z)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[TODO]

Related

[TODO: functionName, functionName]
function reference

_IrrSetCenterOfEffect

todo

```c
#include <au3Irrlicht2.au3>
_IrrSetCenterOfEffect($h_ParticleAffector, $f_X, $f_Y, $f_Z)
```

parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

return value

- success: explanation
- failure: explanation
  - moreExplanationIndented

remarks
todo

related
todo: functionName, functionName
_IrrGetRootSceneNode

Get the scenes root node, all scene nodes are children of this node.

```
#include <au3Irrlicht2.au3>
_IrrGetRootSceneNode()
```

**Parameters**

None.

**Return Value**

Success: Handle of root node in the scene
Failure: False and @error 1

**Remarks**

[todo]

**Related**

_IrrAddMeshToScene
_IrrGetMesh

Loads the specified mesh ready to be added to the scene.

```plaintext
#include <au3Irrlicht2.au3>
_IrrGetMesh($s_MeshFile)
```

**Parameters**

$s_MeshFile  Filename of the mesh object to load

**Return Value**

Success: Handle of the loaded mesh object  
Failure: False

**Remarks**

The Irrlicht engine supports a wide range of mesh types.

### Static objects:

<table>
<thead>
<tr>
<th>Irrlicht static meshes (.irrmesh, r/w)</th>
<th>3D Studio meshes (.3ds, r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias Wavefront Maya (.obj, r/w)</td>
<td>Lightwave Objects (.lwo, r)</td>
</tr>
<tr>
<td>COLLADA 1.4 (.xml, .dae, r/w)</td>
<td>OGRE meshes (.mesh, r)</td>
</tr>
<tr>
<td>My3DTools 3 (.my3D, r)</td>
<td>LMTools (.lmts, r)</td>
</tr>
<tr>
<td>Quake 3 levels (.bsp, r)</td>
<td>DeleD (.dmf, r)</td>
</tr>
<tr>
<td>FSRad oct (.oct, r)</td>
<td>Cartography shop 4 (.csm, r)</td>
</tr>
<tr>
<td>STL 3D files (.stl, r/w)</td>
<td>PLY 3D files (.ply, r/w)</td>
</tr>
</tbody>
</table>

### Animated objects:

Microsoft DirectX (.x, r) (binary & text,
Related

_IrrAddMeshToScene, _IrrRemoveMesh

Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

local $camera = _IrrAddCamera(2,2,2, 0,0,0)

copy code to clipboard
_IrrCreateMesh

Create a new mesh from lists of vertices and indices.

#include <au3Irrlicht2.au3>
_IrrCreateMesh($s_MeshName, $tVertexArray, $a_Indices)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_MeshName</td>
<td>String name for the newly created mesh object.</td>
</tr>
<tr>
<td>$tVertexArray</td>
<td>Vertex array struct as created with __CreateVertStruct or returned from _IrrGetMeshVertices.</td>
</tr>
<tr>
<td>$a_Indices</td>
<td>3D-array with list of indices as returned from _IrrGetMeshIndices or created e.g. with DIM $aIndices[indicesNumber] = [0,1,4, 1,2,4, ...]</td>
</tr>
</tbody>
</table>

Return Value

Success: Handle to the newly created mesh object
Failure: False and set @error:
  @error 1 : either AutoIt DllCall or IrrCreateMesh call failed
  @error 2 : $tVertexArray param is not a Struct
  @error 2 : $a_Indices param is not an Array

Remarks

You must supply a list of vertices inside a vertex array struct and an array of indices that refer to these vertices.
The indices are taken in groups of three joining up the dots defined by the vertices and forming a collection of triangles.
Related

_IrrGetMeshVertices, _IrrGetMeshIndices, _IrrAddMeshToScene, __CreateVertStruct

Example

```
#include "au3Irrlicht2.au3"

__IrrStart()

local $orgMesh = _IrrGetMesh( "./media/zumlin.md2" )

local $tVertex; variable for the vertex array struct
; copy the vertex information into the array:
local $vertex_count = _IrrGetMeshVertices( $orgMesh, 0, $tVertex)

local $i
for $i = 0 to $vertex_count - 1 ; iterate through all of the vertices
  ; shrink vertex Y location by half its size, then change vertex colour value
  __SetVertStruct($tVertex, $i, $VERT_Y, __GetVertStruct($tVertex, $i, $VERT_Y) * 0.5 )
  __SetVertStruct($tVertex, $i, $VERT_VCOLOR, _IrrMakeARGB(0, Random(0,255), Random(0,255), Random(0,255) ) )
next ; $i

; create a second mesh with the modified vertices data and unmodified indices:
local $aIndices ; variable that will hold array of indices:
_IrrGetMeshIndices($orgMesh, 0, $aIndices)
local $secondMesh = _IrrCreateMesh("secondMesh", $tVertex, $aIndices )

local $texture = _IrrGetTexture("\media\default_texture.png")
```
; add both meshes and a camera to the scene:
local $nodeCube1 = _IrrAddMeshToScene( $orgMesh )
_IrrSetNodeMaterialTexture( $nodeCube1,
_IrrGetTexture(".\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeCube1, $IRR_EMF_LIGHTING, $IRR_OFF )
local $nodeCube2 = _IrrAddMeshToScene( $secondMesh )
_IrrSetNodeMaterialFlag( $nodeCube2, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodePosition($nodeCube2, 0, 0, 30)

_IrrAddCamera(50, 0, 30, 0, 18)

WHILE _IrrRunning()
   _IrrBeginScene( 0, 0, 25 )
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
_IrrAddHillPlaneMesh

Creates a hill plane mesh that represents a simple terrain.

```c
#include <au3Irrlicht2.au3>
_IrrAddHillPlaneMesh($s_Name, $f_TileSizeX, $f_TileSizeY,
$i_TileCountX, $i_TileCountY, $h_Material = 0, $f_HillHeight = 0,
$f_CountHillsX = 0, $f_CountHillsY = 0, $f_TextureRepeatCountX = 1,
$f_TextureRepeatCountY = 1)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_Name</td>
<td>Name of the mesh</td>
</tr>
<tr>
<td>$f_TileSizeX</td>
<td>X size of a tile</td>
</tr>
<tr>
<td>$f_TileSizeY</td>
<td>Y size of a tile</td>
</tr>
<tr>
<td>$i_TileCountX</td>
<td>X count of tiles</td>
</tr>
<tr>
<td>$i_TileCountY</td>
<td>Y count of tiles</td>
</tr>
<tr>
<td>$h_Material</td>
<td>Material handle</td>
</tr>
<tr>
<td>$f_HillHeight</td>
<td>Height of the hills</td>
</tr>
<tr>
<td>$f_CountHillsX</td>
<td>X count of hills</td>
</tr>
<tr>
<td>$f_CountHillsY</td>
<td>Y count of hills</td>
</tr>
<tr>
<td>$f_TextureRepeatCountX</td>
<td>X repeat count of texture</td>
</tr>
<tr>
<td>$f_TextureRepeatCountY</td>
<td>Y repeat count of texture</td>
</tr>
</tbody>
</table>

### Return Value

- **Success:** Handle to a Terrain Mesh
- **Failure:** False and @error 1

### Remarks
Many properties have default values allowing a mesh to be created with a simple call.

Related
[todo: functionName, functionName]
_IrrWriteMesh

Write the first frame of the supplied animated mesh out to a file using the specified file format.

```
#include <au3Irrlicht2.au3>
_IrrWriteMesh($h_Mesh, $i_FileFormat, $s_Filename)
```

Parameters

<table>
<thead>
<tr>
<th>$h_Mesh</th>
<th>Handle to mesh object</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Filename</td>
<td>Format to write the file as:</td>
</tr>
<tr>
<td></td>
<td>$EMWT_IRR_MESH - Irrlicht Native mesh writer, for static .irrmesh files.</td>
</tr>
<tr>
<td></td>
<td>$EMWT_COLLADA - COLLADA mesh writer for .dae and .xml files.</td>
</tr>
<tr>
<td></td>
<td>$EMWT_STL - STL mesh writer for .stl files.</td>
</tr>
<tr>
<td>$s_Filename</td>
<td>File name to save as.</td>
</tr>
</tbody>
</table>

Return Value

Success: True

Failure: False and set @error

  @error 1 - AutoIt DllCall failed.
  @error 2 - Could not get mesh writer object.
  @error 3 - Could not open file.

Remarks

None
Related

_IrrCreateMesh, _IrrGetMesh

Example

```c
#include "au3Irrlicht2.au3"

Global $hMD2Mesh
Global $hMeshTexture
Global $hSceneNode
Global $hCamera
Global $hIrrMesh
Global $sIrrMesh = "ZumlinStaticMesh.irmesh"

_IrrStart()

$hMD2Mesh = _IrrGetMesh("./media/zumlin.md2")
$hMeshTexture = _IrrGetTexture("./media/zumlin.pcx")
$hSceneNode = _IrrAddMeshToScene($hMD2Mesh)
_IrrSetNodeMaterialTexture($hSceneNode, $hMeshTexture, 0)
_IrrSetNodeMaterialFlag($hSceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)
_IrrSetNodePosition($hSceneNode, 0, 0, 20)

If _IrrWriteMesh($hMD2Mesh, $EMWT_IRR_MESH, $sIrrMesh) And FileExists($sIrrMesh) Then
    $hIrrMesh = _IrrGetMesh($sIrrMesh)
    $hSceneNode = _IrrAddMeshToScene($hIrrMesh)
    _IrrSetNodeMaterialTexture($hSceneNode, $hMeshTexture, 0)
    _IrrSetNodeMaterialFlag($hSceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)
    _IrrSetNodePosition($hSceneNode, 0, 0, -20)
    FileDelete($sIrrMesh)
EndIf
```
$hCamera = _IrrAddCamera(50, 0, 0, 0, 0, 0)

While _IrrRunning()
   _IrrBeginScene(240, 255, 255)
   _IrrDrawScene()
   _IrrEndScene()
WEnd

_IrrStop()

Copy to Clipboard
_IrrRemoveMesh

Removes a mesh from the scene cache, freeing up resources.

```c
#include <au3Irrlicht2.au3>
_IrrRemoveMesh($h_Mesh)
```

**Parameters**

$\text{h} \_\text{Mesh}$ | Handle of a mesh object
---|---

**Return Value**

Success: True
Failure: False

**Remarks**

None

**Related**

_\text{IrrGetMesh}

**Example**

```c
#include "au3Irrlicht2.au3"

_IrrStart()
```
local $camera = _IrrAddCamera(2,2,2,0,0,0)

local $mesh = _IrrGetMesh( ",\media\capsuleX.obj" )
local $sceneNode = _IrrAddMeshToScene( $mesh )

_IrrRemoveMesh($mesh) ; it's in the scene, mesh no longer needed!

local $texture = _IrrGetTexture("\media\default_texture.png")
_IrrSetNodeMaterialTexture( $sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag( $sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

WHILE _IrrRunning()
    _IrrBeginScene(50,50,50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
_IrrClearUnusedMeshes

Clears all meshes that are held in the mesh cache but not used anywhere else.

```
#include <au3Irrlicht2.au3>
_IrrClearUnusedMeshes()
```

**Parameters**

None.

**Return Value**

Success: True
Failure: False

**Remarks**

Any references to these meshes will become invalid.

**Related**

[(todo: functionName, functionName)]
_IrrSetMeshHardwareAccelerated

Set the supplied mesh as a Hardware Accelerated object.

```c
#include <au3Irrlicht2.au3>
_IrrSetMeshHardwareAccelerated($h_mesh, $i_frame = 0)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_mesh</td>
<td>Handle of a mesh object</td>
</tr>
<tr>
<td>$i_frame</td>
<td>Frame number</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True  
Failure: False

**Remarks**

This offloads the vertices and indicies to hardware support on the graphics card, making the process of rendering those meshes much faster.

The feature must be supported on the graphics card and the object must contain over 500 vertices for the operation to be successful.

This operation is applied to all mesh buffers in the mesh.

**Related**

[todo: functionName, functionName]
_IrrGetMeshIndexCount

[todo]

#Include <au3Irrlicht2.au3>
_IrrGetMeshIndexCount($h_Mesh, $i_Frame, $i_MeshBuffer = 0)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetMeshIndices

Gets the list of indices in a mesh and copies them into the supplied variable.

```c
#include <au3Irrlicht2.au3>
_IrrGetMeshIndices($h_Mesh, $i_FrameNumber, ByRef $a_IndicesArray, $i_MeshBuffer = 0)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Mesh</td>
<td>Handle to a mesh object</td>
</tr>
<tr>
<td>$i_FrameNumber</td>
<td>Frame number of the mesh to get indices from (should be 0 for static meshes).</td>
</tr>
<tr>
<td>$i_MeshBuffer</td>
<td>[optional] Mesh buffer to access.</td>
</tr>
</tbody>
</table>

### Return Value

Success: Number of indices returned in the array $a_IndicesArray.
Failure: False and @error = 1

### Remarks

[todo]

### Related

_`_IrrSetMeshIndices`, _`_IrrGetMeshIndexCount`, _`_IrrGetMeshVertices`
_IrrSetMeshIndices

[todo]

#include <au3Irrlicht2.au3>
_IrrSetMeshIndices($h_Mesh, $i_FrameNumber, ByRef $a_IndicesArray, $i_MeshBuffer = 0)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

todo

Related

todo: functionName, functionName
_IrrGetMeshVertexCount
[done]

#include <au3Irrlicht2.au3>
_IrrGetMeshVertexCount($h_Mesh, $i_Frame, $i_MeshBuffer = 0)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[done]

Related

[done: functionName, functionName]
_IrrGetMeshVertices

Gets the list of vertices in a mesh and copies them into the supplied variable.

#include <au3Irrlicht2.au3>
_IrrGetMeshVertices($h_Mesh, $i_FrameNumber, ByRef $tVertex, $i_MeshBuffer = 0)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Mesh</td>
<td>Handle to a mesh object</td>
</tr>
<tr>
<td>$i_FrameNumber</td>
<td>Frame number of the mesh to get vertices from (should be 0 for static meshes).</td>
</tr>
<tr>
<td>$i_MeshBuffer</td>
<td>[optional] Mesh buffer to access.</td>
</tr>
</tbody>
</table>

Return Value

Success: Number of vertices returned in the vertex array struct $tVertex.
Failure: False and @error = 1

Remarks

Each vertex represents a point in the mesh that is the corner of one of the group of triangles that is used to construct the mesh.
If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0.
If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.
Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

local $mesh = _IrrGetMesh( "./media/cube.x" )

local $tVertex; variable for the vertex array struct
; copy the vertex information into the array
local $vertex_count = _IrrGetMeshVertices( $mesh, 0, $tVertex )

local $i
for $i = 0 to $vertex_count - 1 ; iterate through all of the vertices
    ; shrink vertex X location by half its size, then change vertex colour value
    __SetVertStruct($tVertex, $i, $VERT_X, __GetVertStruct($tVertex, $i, $VERT_X) * 0.5 )
    __SetVertStruct($tVertex, $i, $VERT_VCOLOR, _IrrMakeARGB(0, Random(0,255), Random(0,255), Random(0,255) ) )
next ; $i

; copy the altered vertex infomation back to the mesh
_IrrSetMeshVertices( $mesh, 0, $tVertex )

; add mesh and camera to the scene:
local $nodeCube = _IrrAddMeshToScene( $mesh )
_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodePosition($nodeCube, -0.5, -0.5, 5)

_IrrAddFPSCamera($IRR_NO_OBJECT, 5, 0.01 )
```
WHILE _IrrRunning()
    _IrrBeginScene( 0, 0, 25 )
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
_IrrSetMeshVertices

This sets the value of the list of vertices in a mesh copying them from the supplied vertex array struct.

```
#Include <au3Irrlicht2.au3>
_IrrSetMeshVertices($h_Mesh, $i_FrameNumber, ByRef $tVertex, $i_MeshBuffer = 0)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Mesh</td>
<td>Handle to a mesh object</td>
</tr>
<tr>
<td>$i_FrameNumber</td>
<td>Frame number of the mesh to write vertices to (should be 0 for static meshes).</td>
</tr>
<tr>
<td>$i_MeshBuffer</td>
<td>[optional] Mesh buffer to access.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: None.
Failure: False and @error = 1

**Remarks**

Each vertex represents a point in the mesh that is the corner of one of the group of triangles that is used to construct the mesh.
If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.
Related

_IrrGetMeshVertices, __CreateVertStruct

Example

```
#include "au3Irrlicht2.au3"

_IrrStart()

local $mesh = _IrrGetMesh( "/media/cube.x" )

local $tVertex; variable for the vertex array struct
; copy the vertex information into the array
local $vertex_count = _IrrGetMeshVertices( $mesh, 0, $tVertex)

local $i
for $i = 0 to $vertex_count - 1 ; iterate through all of the vertices
; shrink vertex X location by half its size, then change vertex colour value
   __SetVertStruct($tVertex, $i, $VERT_X, __GetVertStruct($tVertex, $i, $VERT_X) * 0.5 )
   __SetVertStruct($tVertex, $i, $VERT_VCOLOR, _IrrMakeARGB(0, Random(0,255), Random(0,255), Random(0,255) ) )
next ; $i

; copy the altered vertex infomation back to the mesh
_IrrSetMeshVertices( $mesh, 0, $tVertex )

; add mesh and camera to the scene:
local $nodeCube = _IrrAddMeshToScene( $mesh )
_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodePosition($nodeCube, -0.5, -0.5, 5)

_IrrAddFPSCamera($IRR_NO_OBJECT, 5, 0.01 )

WHILE _IrrRunning()
```
_IrrAddMeshToScene

Adds a mesh to the scene as a new 3D node.

```
#Include <au3Irrlicht2.au3>
_IrrAddMeshToScene($h_Mesh)
```

**Parameters**

| $h_Mesh     | Handle of a mesh object |

**Return Value**

Success: Handle of the new node in the scene  
Failure: False

**Remarks**

None

**Related**

_`_IrrGetMesh

**Example**

```
#include "au3Irrlicht2.au3"

_IrrStart()
```
local $camera = _IrrAddCamera(2,2,2,0,0,0)

local $mesh = _IrrGetMesh("./media\capsuleX.obj")
local $sceneNode = _IrrAddMeshToScene($mesh)

local $texture = _IrrGetTexture("./media\default_texture.png")
_IrrSetNodeMaterialTexture($sceneNode, $texture, 0)
_IrrSetNodeMaterialFlag($sceneNode, $IRR_EMF_LIGHTING, $IRR_OFF)

WHILE _IrrRunning()
    _IrrBeginScene(50, 50, 50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
_IrrAddMeshToSceneAsOcttree

Adds a mesh to the scene as a new 3D node, optimised with an Octtree.

```
#include <au3Irrlicht2.au3>
_IrrAddMeshToSceneAsOcttree($h_Mesh)
```

**Parameters**

$h_Mesh$ | Handle of a mesh object

**Return Value**

Success: Handle to the irrlicht node object
Failure: False

**Remarks**

This method optimise's the mesh with an Octtree, this is particularly useful for maps where there is a lot of geometry in the mesh but little of it can be seen at any one time.
Optimizing your node with this function will result in a large increase in performance.

**Related**

None.

**Example**

```au3```
#include "au3Irrlicht2.au3"

_IrrStart()

_IrrAddZipFile( "/media/map-20kdm2.pk3", $IRR_IGNORE_CASE, $IRR_IGNORE_PATHS )
$BSPMesh = _IrrGetMesh( "20kdm2.bsp" )
local $BSPNode = _IrrAddMeshToSceneAsOcttree( $BSPMesh )

local $camera = _IrrAddFPSCamera()
_IrrSetNodePosition( $camera, 1750, 149, 1369 )

WHILE _IrrRunning()
  _IrrBeginScene(50, 50, 50)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
_IrrAddStaticMeshForNormalMappingToScene

[todo]

```au3
#include <au3Irrlicht2.au3>
_IrrAddStaticMeshForNormalMappingToScene($h_Mesh)
```

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrLoadScene

[todo]

#include <au3Irrlicht2.au3>
_IrrLoadScene($s_Filename)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related
[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSaveScene

[todo]

#include <au3Irrlicht2.au3>
_IrrSaveScene($s_Filename)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetSceneNodeFromId

[todo]

#include <au3Irrlicht2.au3>
_IrrGetSceneNodeFromId($i_ID)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetSceneNodeFromName

[todo]

#include <au3Irrlicht2.au3>
_IrrGetSceneNodeFromName($s_Name)

Parameters

[param1] [explanation] [moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrAddBillBoardToScene

Adds a billboard to the scene of the specified size and at the specified position.

```cpp
#include <au3Irrlicht2.au3>
_IrrAddBillBoardToScene($f_XSize, $f_YSize, $f_XPos = 0.0, $f_YPos = 0.0, $f_ZPos = 0.0)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_XSize</td>
<td>X size of the node</td>
</tr>
<tr>
<td>$f_YSize</td>
<td>Y size of the node</td>
</tr>
<tr>
<td>$f_XPos</td>
<td>[optional] X position</td>
</tr>
<tr>
<td>$f_YPos</td>
<td>[optional] Y position</td>
</tr>
<tr>
<td>$f_ZPos</td>
<td>[optional] Z position</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Handle of the new billboard scene node
Failure: False

**Remarks**

A billboard is a flat 3D textured sprite that always faces towards the camera. You need to texture this element with a separate command.

**Related**

_IrrSetNodeMaterialTexture, _IrrSetNodeMaterialFlag
Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

local $BillboardTexture = _IrrGetTexture( "./media/au3irr2_logo.jpg" )
local $Billboard = _IrrAddBillBoardToScene( 200.0,102, 0.0,0.0,100.0 )

_IrrSetNodeMaterialTexture( $Billboard, $BillboardTexture, 0 )
_IrrSetNodeMaterialFlag( $Billboard, $IRR_EMF_LIGHTING, $IRR_OFF )

local $Camera = _IrrAddFPSCamera()

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 0)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard```
au3Irr2 Function Reference

_IrrSetBillBoardSize

[todo]

#include <au3Irrlicht2.au3>
_IrrSetBillBoardSize($h_Node, $f_Width, $f_Height)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrAddBillboardTextSceneNode

(todo)

#include <au3Irrlicht2.au3>
/include <au3Irrlicht2.au3>
/IrrAddBillboardTextSceneNode($h_Font, $s_Text, $f_XSize, $f_YSize,
$f_XPos=0, $f_YPos=0, $f_ZPos=0, $h_Parent=0,
$i_TopRGBA=0xFFFFFFFF, $i_BottomRGBA=0xFFFFFFFF)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]

Example
```c
#include "au3Irrlicht2.au3"

_IrrStart()

local $font = _IrrGetFont( "./media/fonthaettenschweiler.bmp" )

_IrrAddBillboardTextSceneNode( $font, "au3Irrlicht2 ...", 30.0, 15.0, 0.0, 0.0, 20.0, 0, _
    _IrrMakeARGB(0, 255, 0, 0), _IrrMakeARGB(0, 255, 255, 0) )
_IrrAddBillboardTextSceneNode( $font, "... what else?", 30.0, 15.0, 0.0, -5.0, 17.0, 0, _
    _IrrMakeARGB(0, 255, 255, 0), _IrrMakeARGB(0, 0, 255, 0) )

_IrrAddFPSCamera()

WHILE _IrrRunning()
    _IrrBeginScene(0,0,25)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
### _IrrAddParticleSystemToScene

Adds a particle system to the irrlicht scene manager.

```au3
#include <au3Irrlicht2.au3>
_IrrAddParticleSystemToScene($b_AddEmitter, $h_Parent = 0, $i_Id = -1, $f_PosX = 0, $f_PosY = 0, $f_PosZ = 0, $f_RotX = 0, $f_RotY = 0, $f_RotZ = 0, $f_ScaleX = 1, $f_ScaleY = 1, $f_ScaleZ = 1)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_AddEmitter</td>
<td>Whether default emitter shall be created or not: $IRR_NO_EMITTER - For no default emitter (this is probably the option you will use and you will then add a specific emitter later). $IRR_DEFAULT_EMITTER - To create a default emitter that ejects a thin vertical stream of particles.</td>
</tr>
<tr>
<td>$h_Parent</td>
<td>[optional] Handle of scene node the particle shall be attached to (0 means attach to the root scene node)</td>
</tr>
<tr>
<td>$i_Id</td>
<td>[optional] Assigns given integer as ID to the created particle system.</td>
</tr>
<tr>
<td>$f_RotX, $f_RotY, $f_RotZ</td>
<td>[optional] Rotate the particle system along x, y, z axes (0-360).</td>
</tr>
<tr>
<td>$f_ScaleX, $f_ScaleY, $f_ScaleZ</td>
<td>[optional] Scaling factors for created particle system.</td>
</tr>
</tbody>
</table>

### Return Value
success: Handle of the created particle system.
failure: False

Remarks
A particle system is an object that creates and manages hundreds of small billboard like objects that are used to represent smoke, rain and other natural effects. Once created you then need to add emitters and affectors to create and control the particles.

Related
_IrrAddParticleEmitter, _IrrAddFadeOutParticleAffector, _IrrAddGravityParticleAffector, _IrrAddParticleAttractionAffector, _IrrAddRotationAffector

Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

; add particle system with default emitter to the irrlicht scene manager and scale it for more depth
local $particleSystem = _IrrAddParticleSystemToScene($IRR_DEFAULT_EMITTER, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 3, 6)

; load a grey smoke like image for the particle
local $ParticleTexture = _IrrGetTexture( "./media/ParticleGrey.bmp" )

; apply the texture to the particles system to be drawn across each particles surface
_IrrSetNodeMaterialTexture( $particleSystem, $ParticleTexture, 0 )

; particle system is not affected by lighting so make it self illuminating
```
_IrrSetNodeMaterialFlag($particleSystem, $IRR_EMF_LIGHTING, $IRR_OFF)
;

; don't draw black parts of the particle texture:
_IrrSetNodeMaterialType($particleSystem, $IRR_EMT_TRANSPARENT_ADD_COLOR)

local $nodeCamera = _IrrAddCamera(80,0,0,20,40,0)

WHILE _IrrRunning()
    _IrrBeginScene(0,0,50)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
_IrrAddSkyBoxToScene

Adds a skybox node to the scene.

```
#include <au3Irrlicht2.au3>
_IrrAddSkyBoxToScene($h_UpTexture, $h_DownTexture, $h_LeftTexture,
$h_RightTexture, $h_FrontTexture, $h_BackTexture)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_UpTexture</td>
<td>Handle of the top side texture</td>
</tr>
<tr>
<td>$h_DownTexture</td>
<td>Handle of the bottom side texture</td>
</tr>
<tr>
<td>$h_LeftTexture</td>
<td>Handle of the left side texture</td>
</tr>
<tr>
<td>$h_RightTexture</td>
<td>Handle of the right side texture</td>
</tr>
<tr>
<td>$h_FrontTexture</td>
<td>Handle of the front side texture</td>
</tr>
<tr>
<td>$h_BackTexture</td>
<td>Handle of the back side texture</td>
</tr>
</tbody>
</table>

**Return Value**

success: Handle of the skybox node
failure: False

**Remarks**

A skybox is a huge hollow cube that encapsulates the entire scene and has a different texture applied to each of its six surfaces to represent a distant sky or matte scene.

**Related**

_IrrAddSkyDomeToScene
#include <au3Irrlicht2.au3>

_IrrStart( )

local $nodeSkyBox = _IrrAddSkyBoxToScene( _
    _IrrGetTexture("./media/irrlicht2_up.jpg"), _
    _IrrGetTexture("./media/irrlicht2_dn.jpg"), _
    _IrrGetTexture("./media/irrlicht2_rt.jpg"), _
    _IrrGetTexture("./media/irrlicht2_lf.jpg"), _
    _IrrGetTexture("./media/irrlicht2_ft.jpg"), _
    _IrrGetTexture("./media/irrlicht2_bk.jpg"))

local $nodeCamera = _IrrAddFPSCamera()

WHILE _IrrRunning()
    _IrrBeginScene(240, 255, 255)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
au3Irr2 Function Reference

_IrrAddSkyDomeToScene

todo

#include <au3Irrlicht2.au3>

_IrrAddSkyDomeToScene($h_Texture, $i_HorizontalRes, $i_VerticalRes, $d_TexturePercent, $d_SpherePercent, $d_SphereRadius = 1000.0)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]

Remarks

todo

Related

_IrrAddSkyBoxToScene, _IrrSetSkyDomeColor,
_IrrSetSkyDomeColorBand, _IrrSetSkyDomeColorPoint
_IrrAddTestSceneNode

Adds a simple cube node to the scene

```
#include <au3Irrlicht2.au3>
_IrrAddTestSceneNode()
```

### Parameters

None.

### Return Value

- **success**: Handle of the cube scene node
- **failure**: False

### Remarks

The test scene node is a cube with fixed dimensions mainly for test purposes.

### Related

- _IrrAddCubeSceneNode
- _IrrAddSphereSceneNode

### Example

```
#include "au3Irrlicht2.au3"

_IrrStart()
```
local $nodeCamera = _IrrAddCamera(10,10,10, 0,0,0)
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF)

WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 0)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
_IrrAddCubeSceneNode

Adds a cube object to the scene with the specified dimensions.

```
#include <au3Irrlicht2.au3>
_IrrAddCubeSceneNode($f_Size)
```

**Parameters**

$\text{f\_Size}$  
Edge length of the cube.

**Return Value**

success: Handle of the cube scene node
failure: False

**Remarks**

None

**Related**

_IrrAddTestSceneNode, _IrrAddSphereSceneNode

**Example**

```
#include "au3Irrlicht2.au3"

_IrrStart()
```
local $nodeCamera = _IrrAddCamera(10, 10, 10, 0, 0)
local $nodeTest = _IrrAddCubeSceneNode(10)
_IrrSetNodeMaterialTexture($nodeTest,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag($nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF)

WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 0)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
_IrrAddSphereSceneNode

Adds a simple sphere object to the scene

```c
#include <au3Irrlicht2.au3>
_IrrAddSphereSceneNode($f_Size, $i_PolyCount = 16)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_Size</td>
<td>Radius of the sphere</td>
</tr>
<tr>
<td>$i_PolyCount</td>
<td>[optional] Level of detail for the sphere. Too high values could produce a very high density mesh and affect your frame rate adversely.</td>
</tr>
</tbody>
</table>

**Return Value**

- success: Handle of the sphere scene node
- failure: False

**Remarks**

None

**Related**

_`IrrAddTestSceneNode`,  _`IrrAddCubeSceneNode`

**Example**

```c
#include "au3Irrlicht2.au3"
```
local $nodeCamera = _IrrAddCamera(10, 10, 10, 0, 0, 0)
local $nodeTest = _IrrAddSphereSceneNode(8, 32)
_IrrSetNodeMaterialTexture($nodeTest, _IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag($nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF)

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 0)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
#Include <au3Irrlight2.au3>
_IrrAddWaterSurfaceSceneNode($h_Mesh, $f_WaveHeight = 2.0, $f_WaveSpeed = 300.0, $f_WaveLength = 10.0, $h_Parent = 0, $i_ID = -1, $f_PosX = 0, $f_PosY = 0, $f_PosZ = 0, $f_RotX = 0, $f_RotY = 0, $f_RotZ = 0, $f_ScaleX = 1.0, $f_ScaleY = 1.0, $f_ScaleZ = 1.0)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrAddZoneManager

.todo

#include <au3Irrlicht2.au3>
_IrrAddZoneManager($f_NearDistance=0, $f_FarDistance=12000)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

.success: explanation
.failure: explanation
  moreExplanationIndented

Remarks

todo

Related

todo: functionName, functionName
au3Irr2 Function Reference

_IrrAddClouds

[todo]

#include <au3Irrlicht2.au3>
_IrrAddClouds($h_Texture, $i_Lod, $i_Depth, $i_Density)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]

Example

#include "au3Irrlicht2.au3"
Global $hCloudNode, $hCloudTexture

…IrrStart()

$hCloudTexture = _IrrGetTexture("./media/cloud4.png")
$hCloudNode = _IrrAddClouds($hCloudTexture, 10, 1, 200)
_IrrSetNodePosition($hCloudNode, 0, 200, 0)

_IrrAddCamera(50, 0, 0, 0, 0)

While _IrrRunning() And Sleep(10)
   _IrrBeginScene(128, 128, 255)
   _IrrDrawScene()
   _IrrEndScene()
WEnd

_IrrStop()
_IrrAddLensFlare

Adds a lens flare patch object to the scene, this object simulates the effect of bright lights on the optics of a camera.

#include <au3Irrlicht2.au3>
_IrrAddLensFlare($h_Texture)

Parameters

| $h_Texture | Handle to texture (image containing a series of 128x128 images representing stages of the lens flare). |

Return Value

Success: Handle to a flare node
Failure: False and @error 1

Remarks

todo

Related

_IrrSetFlareScale, _IrrGetTexture

Example

#include "au3Irrlicht2.au3"
_IrrStart()

local $hFlare = _IrrAddLensFlare(_IrrGetTexture("./media/flares.jpg"))
_IrrSetNodePosition($hFlare, 300, 100, 1000)

_IrrAddFPSCamera()
_IrrHideMouse()

While _IrrRunning()
  _IrrBeginScene(180, 225, 255)
  _IrrDrawScene()
  _IrrEndScene()
WEnd

_IrrStop()
_IrrAddGrass

Adds a grass object to the scene.

```
#include <au3Irrlicht2.au3>
_IrrAddGrass($h_Terrain, $i_X, $i_Y, $i_PatchSize, $f_FadeDistance, $i_Crossed, $f_GrassScale, $i_MaxDensity, $i_DataPosX, $i_DataPosY, $h_HeightMap, $h_TextureMap, $h_GrassMap, $h_GrassTexture)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Terrain</td>
<td>[explanation]</td>
</tr>
<tr>
<td>$i_X</td>
<td>X Position of grass patch.</td>
</tr>
<tr>
<td>$i_Y</td>
<td>Y Position of grass patch.</td>
</tr>
<tr>
<td>$i_PatchSize</td>
<td>Size of grass patch</td>
</tr>
<tr>
<td>$f_FadeDistance</td>
<td>Distance at which the number of displayed grass elements in that patch are reduced. If this is set to 1.0 then when the camera is inside the patch all of grass will be displayed but once outside less and less will be shown. By increasing this to 2.0 then all of the grass is shown until the camera is two patches distant. This gives a better appearance but reduces performance as more grass has to be drawn.</td>
</tr>
<tr>
<td>$i_Crossed</td>
<td>IRR_ON or IRR_OFF. When off each piece of grass is a separate entity with its own position and rotation. When On grass is paired up and placed to form a cross. Crossed grass can have a better appearance as you rotate around it. However individual grass can give the impression that there is more of it and you can therefore reduce the number of grass blades and increase performance.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$f_{\text{GrassScale}}$</td>
<td>Scale of a grass patch.</td>
</tr>
<tr>
<td>$i_{\text{MaxDensity}}$</td>
<td>number of individual clumps of foliage that are created.</td>
</tr>
<tr>
<td>$i_{\text{DataPosX}}$</td>
<td>X position of large bitmap associated with a tiled terrain and allow the color information to be taken from an offset position on the bitmap.</td>
</tr>
<tr>
<td>$i_{\text{DataPosY}}$</td>
<td>Y position of large bitmap associated with a tiled terrain and allow the color information to be taken from an offset position on the bitmap.</td>
</tr>
<tr>
<td>$h_{\text{TextureMap}}$</td>
<td>Handle to an image that contains the height of the terrain onto which the grass is placed.</td>
</tr>
<tr>
<td>$h_{\text{GrassMap}}$</td>
<td>Handle to an image used to adjust the height and density of the grass. For example you might have a patch where you dont want to see any grass or a barren patch where you want short stubble.</td>
</tr>
<tr>
<td>$h_{\text{GrassTexture}}$</td>
<td>Handle to a Texture used for the grass. This RGBA image is automatically broken up into a number of sections that are used to texture different clumps of grass.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: Handle to a Grass node.
Failure: False and @error 1

**Remarks**

Grass objects are associated with terrain and tile terrain objects and are used to place small billboard objects into the scene representing foliage, this implementation of grass creates a large number of grass objects already positioned across the terrain and then dynamically shows or hides them depending on where the camera is within the scene.

The grass is also affected with a wind modifier that gently moves the grass as if it were caught in the wind.

By setting the speed of the wind to zero the grass will become static and you will see an increase in performance.
Grass usually looks best when it is closely matched to the color of the terrain and to assist with this a new Material Type has been added IRR_EMT_TRANSPARENT_ADD_ALPHA_CHANNEL_REF that adds the color of grass texture to the color of the grass which is automatically set to the color of the terrain that it lies upon.

**Related**

-IrrGetGrassDrawCount, _IrrSetGrassDensity, _IrrSetGrassWind
_IrrSetShadowColor

Sets the colour of shadows cast by objects in the scene.

#include <au3Irrlicht2.au3>
_IrrSetShadowColor($i_Alpha, $i_Red, $i_Green, $i_Blue)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Alpha</td>
<td>Alpha blend for the shadow. Value of 128 would mean a half washed out shadow which gives the appearence of ambient light in the room illuminating the shadowed surface.</td>
</tr>
<tr>
<td>$i_Red, $i_Green, $i_Blue</td>
<td>Colour values for shadow colour from 0 to 255.</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False

Remarks

The shadow colour is a global property for the whole scene (however you can change it when moving into different areas of your scene). If you are observing a bright scene you might use a light grey shadow instead of a heavy black shadow to add to realism.

Related

_IrrAddNodeShadow, _IrrAddLight
Example

```
#include "au3Irrlicht2.au3"

_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _
        $IRR_WINDOWED, $IRR_SHADOWS, $IRR_IGNORE_EVENTS,
        $IRR_VERTICAL_SYNC_ON )

local $texture = _IrrGetTexture("\media\au3irr2_logo.jpg")

local $meshReceive = _IrrAddHillPlaneMesh( "Plane", 8, 8, 1, 1)
local $nodeReceive = _IrrAddMeshToScene( $meshReceive )
    _IrrSetNodeMaterialTexture( $nodeReceive, $texture, 0)

local $meshCast = _IrrGetMesh("\media\sphere.obj")
local $nodeCast = _IrrAddMeshToScene($meshCast)
    _IrrSetNodeMaterialTexture( $nodeCast, $texture, 0)
    _IrrAddFlyCircleAnimator($nodeCast, 1, 3, 1, 3, 0.0005)

local $nodeLight = _IrrAddLight($IRR_NO_PARENT, 0,50,0, 1,0.1,0.1, 1000.0)
    _IrrAddNodeShadow($nodeCast)
    _IrrSetShadowColor(128, 255, 0, 0)

local $nodeCamera = _IrrAddCamera(-5,5,-6, 0,0,0 )

WHILE _IrrRunning()
    _IrrBeginScene(100, 100, 100)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
au3Irr2 Function Reference

__IrrSetFog__

Set the properties of fog in the scene

```au3Irrlicht<au3>
_IrrSetFog($i_Red, $i_Green, $i_Blue, $i_FogType, $f_FogStart,
$f_FogEnd, $f_Density = 0.025)
```

**Parameters**

- `$i_Red, $i_Green, $i_Blue` Define colour of the fog (0-255). Should be set to the same colour as the scene sky so the scene fogs out nicely into nothing.
- `$i_FogType` Defines how the fog is calculated: $IRR_LINEAR_FOG - computed as [end - distance / end - start], density value is not used. $IRR_EXPONENTIAL_FOG - computed as [1 / (2.718^(distance * density))], both start and end values are not used.
- `$f_FogStart, $f_FogEnd` Distances at which the fog starts and at which it reaches its maximum density. Values are ignored for exponential fog.
- `$f_Density` [optional] Determines how quickly the exponential change takes place, with value from 0 to 1. Example: A value of 0.025 equals 20% visibility at 50 units distance. Value is ignored for linear fog.

**Return Value**

success: True
failure: False
Remarks
None.

Related
None.

Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeCamera = _IrrAddCamera(10, 10, 10, 0, 0, 0)
local $nodeCube = _IrrAddCubeSceneNode(12)
_IrrSetNodeMaterialTexture( $nodeCube,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_FOG_ENABLE, $IRR_ON )
_IrrSetFog ( 100,100,100, $IRR_EXPONENTIAL_FOG, 0.0, 0.0, 0.15)

WHILE _IrrRunning()
    _IrrBeginScene(100, 100, 100)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
_IrrDraw3DLine

Draws a line onto the display using 3D co-ordinates and a specified colour.

#include <au3Irrlicht2.au3>
_IrrDraw3DLine($f_XStart, $f_YStart, $f_ZStart, $f_XEnd, $f_YEnd, $f_ZEnd, $i_Red, $i_Green, $i_Blue)

Parameters

| $f_XStart, $f_YStart, $f_ZStart | Defines start point for the 3D-line. |
| $f_XEnd, $f_YEnd, $f_ZEnd | Defines end point for the 3D-line. |
| $i_Red, $i_Green, $i_Blue | Colour values for the 3D-Line (0-255). |

Return Value

success: True
failure: False

Remarks

The lines are not part of the Irrlicht scene but drawn before and separately. They need to be redrawn for every new frame.
Related

_IrrBeginScene, _IrrEndScene()

Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

local $i
local $nodeCamera = _IrrAddFPSCamera()
_IrrSetCameraTarget($nodeCamera, 50, 0, 50)

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 50)

    ; draw a grid to screen before drawing the scene:
    for $i = 0 to 250 step 25
        _IrrDraw3DLine($i, -25, 0, $i, -25, 250, 255, 255, 0)
        _IrrDraw3DLine(0, -25, $i, 250, -25, $i, 255, 255, 0)
    next ; $i

    _IrrDrawScene()
    _IrrEndScene()

WEND

_IrrStop()
```

Copy to Clipboard
_IrrSetSkyDomeColor

Set the color of the vertices in the skydome.

```au3
#include <au3Irrlicht2.au3>
_IrrSetSkyDomeColor($h_Dome, $i_HorizontalRed, $i_HorizontalGreen, $i_HorizontalBlue, $i_ZenithRed, $i_ZenithGreen, $i_ZenithBlue)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Dome</td>
<td>[explanation]</td>
</tr>
<tr>
<td>$i_HorizontalRed</td>
<td>Red color 0 - 255</td>
</tr>
<tr>
<td>$i_HorizontalGreen</td>
<td>Green color 0 - 255</td>
</tr>
<tr>
<td>$i_HorizontalBlue</td>
<td>Blue color 0 - 255</td>
</tr>
<tr>
<td>$i_ZenithRed</td>
<td>Red color 0 - 255</td>
</tr>
<tr>
<td>$i_ZenithGreen</td>
<td>Green color 0 - 255</td>
</tr>
<tr>
<td>$i_ZenithBlue</td>
<td>Blue color 0 - 255</td>
</tr>
</tbody>
</table>

**Return Value**

- Success: True
- Failure: False

**Remarks**

Two colors are defined one for the horizon and another for the top of the sky dome, this simulates the type of coloring effects you see in the sky. If you are using a full spherical skydome the horizontal color will be the color at the bottom of the skydome.
Related

__IrrSetSkyDomeColorPoint, __IrrSetSkyDomeColorBand, __IrrSAddSkyDome
_IrrSetSkyDomeColorBand

Creates a horizontal band of color in the skydome.

```au3
#include <au3Irrlicht2.au3>
_IrrSetSkyDomeColorBand($h_Dome, $i_HorizontalRed, $i_HorizontalGreen, $i_HorizontalBlue, $i_BandVerticalPosition, $f_BandFade, $i_Additive)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Dome</td>
<td>[explanation]</td>
</tr>
<tr>
<td>$i_HorizontalRed</td>
<td>Red color 0 - 255</td>
</tr>
<tr>
<td>$i_HorizontalGreen</td>
<td>Green color 0 - 255</td>
</tr>
<tr>
<td>$i_HorizontalBlue</td>
<td>Blue color 0 - 255</td>
</tr>
<tr>
<td>$i_BandVerticalPosition</td>
<td>Vertex at which you wish to create the band.</td>
</tr>
<tr>
<td>$f_BandFade</td>
<td>Amount that the band is faded into the existing skydome color.</td>
</tr>
<tr>
<td>$i_Additive</td>
<td>IRR_ON to add the color of the band to the existing color of the skydome or IRR_OFF to replace it.</td>
</tr>
</tbody>
</table>

### Return Value

- Success: True
- Failure: False

### Remarks

This is mainly useful for creating additional bands of color at the horizon, where your sky is a graduation of blues and then in the morning you have a brighter golden band as the sun rises.
Related

_IrrSetSkyDomeColorPoint, _IrrSetSkyDomeColor, _IrrSAddSkyDome
_IrrSetSkyDomeColorPoint

Set the color of the vertices in the skydome radiating out from a point.

```csh
#include <au3Irrlicht2.au3>
_IrrSetSkyDomeColorPoint($h_Dome, $i_Red, $i_Green, $i_Blue, $f_PosX, $f_PosY, $f_PosZ, $f_Radius, $f_PointFade, $i_Additive)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Dome</td>
<td>Handle to a SkyDome node.</td>
</tr>
<tr>
<td>$i_Red</td>
<td>Red color 0 - 255</td>
</tr>
<tr>
<td>$i_Green</td>
<td>Green color 0 - 255</td>
</tr>
<tr>
<td>$i_Blue</td>
<td>Blue color 0 - 255</td>
</tr>
<tr>
<td>$f_PosX</td>
<td>X position.</td>
</tr>
<tr>
<td>$f_PosY</td>
<td>Y position.</td>
</tr>
<tr>
<td>$f_PosZ</td>
<td>Z position.</td>
</tr>
<tr>
<td>$f_Radius</td>
<td>Limit the distance of the coloring</td>
</tr>
<tr>
<td>$f_PointFade</td>
<td>Amount that the band is faded into the existing skydome color.</td>
</tr>
<tr>
<td>$i_Additive</td>
<td>IRR_ON to add the color of the band to the existing color of the skydome or IRR_OFF to replace it.</td>
</tr>
</tbody>
</table>

### Return Value

Success: True
Failure: False

### Remarks

This is powerful effect that can be used to color parts of the skydome and
create effects to represent the glows of the rising sun or the moon in the sky.

Related
__IrrSetSkyDomeColorBand, __IrrSetSkyDomeColor, __IrrSAddSkyDome
_IrrSetZoneManagerProperties

Sets the draw distances of nodes in the zone/distance management node and whether or not the zone manager is to accumulate the bounding boxes of its children as they are added.

```
#include <au3Irrlicht2.au3>
_IrrSetZoneManagerProperties($h_ZoneManager, $f_NearDistance, $f_FarDistance, $i_AccumulateBoxes)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_ZoneManager</td>
<td>Handle to the zone node.</td>
</tr>
<tr>
<td>$f_NearDistance</td>
<td></td>
</tr>
<tr>
<td>$f_FarDistance</td>
<td></td>
</tr>
<tr>
<td>$i_AccumulateBoxes</td>
<td></td>
</tr>
</tbody>
</table>

**Return Value**

- Success: True
- Failure: False

**Remarks**

[todo]

**Related**

[todo: functionName, functionName]
_IrrSetZoneManagerBoundingBox

Allows the user to manually set the bounding box of a zone manager node.

```
#include <au3Irrlicht2.au3>
_IrrSetZoneManagerBoundingBox($h_ZoneManager, $f_X, $f_Y, $f_Z, $f_BoxWidth, $f_BoxHeight, $f_BoxDepth)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_ZoneManager</td>
<td>Handle to the zone node.</td>
</tr>
<tr>
<td>$f_X</td>
<td>X position.</td>
</tr>
<tr>
<td>$f_Y</td>
<td>Y position.</td>
</tr>
<tr>
<td>$f_Z</td>
<td>Z position.</td>
</tr>
<tr>
<td>$f_BoxWidth</td>
<td>Width of zone box.</td>
</tr>
<tr>
<td>$f_BoxHeight</td>
<td>Height of zone box.</td>
</tr>
<tr>
<td>$f_BoxDepth</td>
<td>Depth of zone box.</td>
</tr>
</tbody>
</table>

### Return Value

Success: True
Failure: False

### Remarks

None.

### Related
[todo: functionName, functionName]
_IrrSetGrassDensity

Set the grass density.

#include <au3Irrlicht2.au3>
_IrrSetGrassDensity($h_Grass, $f_Density, $f_Distance)

Parameters

$\textit{h}_\text{Grass}$  Handle to the grass node.
$\textit{f}_\text{Density}$  Number of grass nodes visible in the scene.
$\textit{f}_\text{Distance}$  Distance at which they can be seen.

Return Value

Success: True
Failure: False

Remarks

None.

Related

_IrrAddGrass, _IrrGetGrassDrawCount, _IrrSetGrassWind
_IrrSetGrassWind

Set the grass wind effect.

#include <au3Irrlicht2.au3>
_IrrSetGrassWind($h_Grass, $f_Strength, $f_Resolution)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_Grass</td>
<td>Handle to the grass node.</td>
</tr>
<tr>
<td>$f_Strength</td>
<td>Strength of the wind.</td>
</tr>
<tr>
<td>$f_Resolution</td>
<td>How often the effect is calculated.</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False

Remarks

By setting the resolution to zero the wind effect will be stopped and there will be a performance increase, however the wind effect adds significantly to the subtle atmosphere of the scene.

Related

_IrrAddGrass,  _IrrGetGrassDrawCount,  _IrrSetGrassDensity
_IrrCreateBatchingMesh

Create a batching mesh that will be a collection of other meshes into a single source mesh.

```c
#include <au3Irrlicht2.au3>
_IrrCreateBatchingMesh()
```

Parameters

None.

Return Value

Success: Handle to a batching mesh.
Failure: False and @error 1

Remarks

The function of the batching mesh is to avoid the use of large numbers of nodes that adds an overhead to the rendering process that can significantly slow it down.
Where you have a forest with a thousand trees you will see a significant increase in performance by batching all of those trees into a smaller number of node.
While this is handled as an irr_mesh it should only be used with batching mesh commands.

Related

[todo: functionName, functionName]
See Also

_IrrAddToBatchingMesh, _IrrFinalizeBatchingMesh
_IrrAddToBatchingMesh

Adds a mesh to the batching mesh at the specified position, rotation and scale.

```
#include <au3Irrlicht2.au3>
_IrrAddToBatchingMesh($h_meshBatch, $h_mesh, $f_posX = 0.0, $f_posY = 0.0, $f_posZ = 0.0, $f_rotX = 0.0, $f_rotY = 0.0, $f_rotZ = 0.0, $f_scaleX = 1.0, $f_scaleY = 1.0, $f_scaleZ = 1.0)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_meshBatch</td>
<td>Handle to a batching mesh as returned by <code>_IrrCreateBatchingMesh</code>.</td>
</tr>
<tr>
<td>$h_mesh</td>
<td>Handle to the mesh to add.</td>
</tr>
<tr>
<td>$f_posX</td>
<td>X position</td>
</tr>
<tr>
<td>$f_posY</td>
<td>Y position</td>
</tr>
<tr>
<td>$f_posZ</td>
<td>Z position</td>
</tr>
<tr>
<td>$f_rotX</td>
<td>X rotation</td>
</tr>
<tr>
<td>$f_rotY</td>
<td>Y rotation</td>
</tr>
<tr>
<td>$f_rotZ</td>
<td>Z rotation</td>
</tr>
<tr>
<td>$f_scaleX</td>
<td>X scale</td>
</tr>
<tr>
<td>$f_scaleY</td>
<td>Y scale</td>
</tr>
<tr>
<td>$f_scaleZ</td>
<td>Z scale</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False
Remarks

If each of your meshes requires a different texture you should call IrrSetMeshMaterialTexture for the mesh you are about to add prior to adding the mesh to the batch.

Related

_IrrCreate batchingMesh, _IrrFinalize batchingMesh
_IrrFinalizeBatchingMesh

Finalises the batching mesh.

```
#include <au3Irrlicht2.au3>
_IrrFinalizeBatchingMesh($h_meshBatch)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_meshBatch</td>
<td>Handle to a batching mesh as returned by _IrrCreateBatchingMesh.</td>
</tr>
</tbody>
</table>

**Return Value**

- **Success:** Handle to a new mesh
- **Failure:** False and @error 1

**Remarks**

This should be called once all of the meshes have been added to the batching mesh.

The function returns a new mesh object that can be used in all standard mesh calls.

**Related**

_IrrCreateBatchingMesh, _IrrAddToBatchingMesh
au3Irr2 Function Reference

_IrrSetMeshMaterialTexture

[ todo ]

```
#include <au3Irrlicht2.au3>
_IrrSetMeshMaterialTexture($h_mesh, $h_texture, $i_index, $i_buffer = 0)
```

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[ success]: [ explanation]
[ failure]: [ explanation]
  [ moreExplanationIndented ]

Remarks

[ todo ]

Related

[ todo: functionName, functionName ]
au3Irr2 Function Reference

_IrrScaleMesh

[todo]

#include <au3Irrlicht2.au3>
_IrrScaleMesh($h_mesh, $f_scale, $i_frame = 0, $i_meshBuffer = 0, $h_sourceMesh = 0)

Parameters

[par1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrAddBeamSceneNode

Add a beam scene node.

```c
#include <au3Irrlicht2.au3>
_IrrAddBeamSceneNode()
```

### Parameters

None.

### Return Value

Success: Handle to a beam node.
Failure: False and @error 1

### Remarks

The beam is a special scene node that can be used to replicate beam effects like lasers and tracer gun fire.
This command simply adds the beam you should then make calls to set the beams properties.

### Related

_ _IrrSetBeamSize, _IrrSetBeamPosition

### Example

```c
#include "au3Irrlicht2.au3"
```
Global $hBeamNode
Global $hTexture
Global $hCamera

_IrrStart()

$hBeamNode = _IrrAddBeamSceneNode()
_IrrSetBeamPosition($hBeamNode, 15.0, 20.0, 20.0, -15.0, -20.0, -20.0)
_IrrSetBeamSize($hBeamNode, 5.0)
$hTexture = _IrrGetTexture("./media/beam.png")
_IrrSetNodeMaterialTexture($hBeamNode, $hTexture, 0)
_IrrSetNodeMaterialType($hBeamNode, $IRR_EMT_TRANSPARENT_ALPHA_CHANNEL)

$hCamera = _IrrAddCamera(50, 0, 0, 0, 0)

While _IrrRunning() And Sleep(10)
   _IrrBeginScene(0, 0, 0)
   _IrrDrawScene()
   _IrrEndScene()
WEnd

_IrrStop()
_IrrSetBeamSize

Sets the width of a beam node.

```
#include <au3Irrlicht2.au3>
_IrrSetBeamSize($h_BeamNode, $f_Size)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_BeamNode</td>
<td>Handle to a beam node.</td>
</tr>
<tr>
<td>$f_Size</td>
<td>Width of the beam node.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True
Failure: False

**Remarks**

None.

**Related**

_IrrAddBeamSceneNode, _IrrSetBeamPosition

**Example**

```
#include "au3Irrlicht2.au3"

Global $hBeamNode
```
Global $hTexture
Global $hCamera

_IrrStart()

$hBeamNode = _IrrAddBeamSceneNode()
_IrrSetBeamPosition($hBeamNode, 15.0, 20.0, 20.0, -15.0, -20.0, -20.0)
_IrrSetBeamSize($hBeamNode, 5.0)
$hTexture = _IrrGetTexture("./media/beam.png")
_IrrSetNodeMaterialTexture($hBeamNode, $hTexture, 0)
_IrrSetNodeMaterialType($hBeamNode, $IRR_EMT_TRANSPARENT_ALPHA_CHANNEL)

$hCamera = _IrrAddCamera(50, 0, 0, 0, 0)

While _IrrRunning() And Sleep(10)
   _IrrBeginScene(0, 0, 0)
   _IrrDrawScene()
   _IrrEndScene()
WEnd

_IrrStop()
_IrrSetBeamPosition

Sets the start and end positions of a beam node. The beam will stretch between the two nodes.

#include <au3Irrlicht2.au3>
_IrrSetBeamPosition($h_BeamNode, $f_SX, $f_SY, $f_SZ, $f_EX, $f_EY, $f_EZ)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_BeamNode</td>
<td>Handle to a beam node.</td>
</tr>
<tr>
<td>$f_SX</td>
<td>X start position</td>
</tr>
<tr>
<td>$f_SY</td>
<td>Y start position</td>
</tr>
<tr>
<td>$f_SZ</td>
<td>Z start position</td>
</tr>
<tr>
<td>$f_EX</td>
<td>X end position</td>
</tr>
<tr>
<td>$f_EY</td>
<td>X end position</td>
</tr>
<tr>
<td>$f_EZ</td>
<td>X end position</td>
</tr>
</tbody>
</table>

Return Value
Success: True
Failure: False

Remarks
None.

Related
Example

```c
#include "au3Irrlicht2.au3"

Global $hBeamNode
Global $hTexture
Global $hCamera

_IRrStart()

$hBeamNode = _IRrAddBeamSceneNode()
_IRrSetBeamPosition($hBeamNode, 15.0, 20.0, 20.0, -15.0, -20.0, -20.0)
_IRrSetBeamSize($hBeamNode, 5.0)
$hTexture = _IRrGetTexture("./media/beam.png")
_IRrSetNodeMaterialTexture($hBeamNode, $hTexture, 0)
_IRrSetNodeMaterialType($hBeamNode, $IRR_EMT_TRANSPARENT_ALPHA_CHANNEL)

$hCamera = _IRrAddCamera(50, 0, 0, 0, 0)

While _IRrRunning() And Sleep(10)
  _IrbeginScene(0, 0, 0)
  _IrdrawScene()
  _IrEndScene()
WEnd

_IRrStop()
```
_IrrAddBoltSceneNode

The bolt is a special scene node that can be used to replicate electrical effects.

```
#include <au3Irrlicht2.au3>
_IrrAddBoltSceneNode()
```

**Parameters**

None.

**Return Value**

Success: Handle to a Bolt node
Failure: False and @error 1

**Remarks**

This command simply adds the bolt you should then make a call to set the bolts properties.
This node can be used to simulate lightning and other electrical effects.

**Related**

_`IrrSetBoltProperties`

**Example**

```
#include "au3Irrlicht2.au3"
```
Global $hBoltNode, $hTexture, $hCamera

_IrrStart()

$hBoltNode = _IrrAddBoltSceneNode()
_IrrSetBoltProperties($hBoltNode, 0, 90, 0, 0, 0, 50, 10, 2, 10, 6, $IRR_ON, 0xFFFFFFFF)
$hTexture = _IrrGetTexture("./media/ParticleBlue.bmp")
_IrrSetNodeMaterialTexture($hBoltNode, $hTexture, 0)

$hCamera = _IrrAddCamera(40, 50, 40, 0, 50)

While _IrrRunning() And Sleep(10)
  _IrrBeginScene(16, 24, 32)
  _IrrDrawScene()
  _IrrEndScene()
WEnd

_IrrStop()
_IrrSetBoltProperties

This sets the properties of a bolt node that simulates an electrical effect.

```
#include <au3Irrlicht2.au3>
_ IrrSetBoltProperties($h_BoltNode, $f_SX, $f_SY, $f_SZ, $f_EX, $f_EY, $f_EZ, $i_UpdateTime=50, $i_Height=10, $f_Thickness=5.0, $i_Parts=10, $i_Bolts=6, $i_Steadyend=$IRR_OFF, $i_Color=0x0000FFFF)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_BoltNode</td>
<td>[explanation]</td>
</tr>
<tr>
<td>$f_SX, $f_SY, $f_SZ</td>
<td>Starting point that the bolt originates from.</td>
</tr>
<tr>
<td>$f_EX, $f_EY, $f_EZ</td>
<td>End terminating point for the bolt.</td>
</tr>
<tr>
<td>$i_UpdateTime</td>
<td>Number of milliseconds between updates to the appearance of the bolt.</td>
</tr>
<tr>
<td>$i_Height</td>
<td>Radius is the radius of the entire bolt effect.</td>
</tr>
<tr>
<td>$f_Thickness</td>
<td>Thickness of a single electrical element in the bolt.</td>
</tr>
<tr>
<td>$i_Parts</td>
<td>Number of segments the bolt is divided into.</td>
</tr>
<tr>
<td>$i_Bolts</td>
<td>Number of individual electrical arcs that are rendered.</td>
</tr>
<tr>
<td>$i_Steadyend</td>
<td>Set to IRR_ON ends in a tight point, Set to IRR_OFF it ends with the same width as the rest of the bolt.</td>
</tr>
<tr>
<td>$i_Color</td>
<td>Diffuse color that is applied to the bolt.</td>
</tr>
</tbody>
</table>

Return Value

Success: True
Failure: False
Remarks
There are a number of properties that control many aspects of the bolt to produce a wide range of appearances.

Related
_IrrAddBoltSceneNode

Example

```cpp
#include "au3Irrlicht2.au3"

Global $hBoltNode, $hTexture, $hCamera

!IrrStart()

$hBoltNode = _IrrAddBoltSceneNode()
!IrrSetBoltProperties($hBoltNode, 0, 90, 0, 0, 0, 50, 10, 2, 10, 6, $IRR_ON, 0xFFFFFFFF)
$hTexture = _IrrGetTexture("./media/ParticleBlue.bmp")
!IrrSetNodeMaterialTexture($hBoltNode, $hTexture, 0)

$hCamera = _IrrAddCamera(40, 50, 40, 0, 50, 0)

While _IrrRunning() And Sleep(10)
  _IrrBeginScene(16, 24, 32)
  _IrrDrawScene()
  _IrrEndScene()
WEnd

!IrrStop()
```

Copy to Clipboard
_IrrSetBillBoardColor

[|todo|]

#Include <au3Irrlicht2.au3>
_IrrSetBillBoardColor($h_Node, $i_TopColor, $i_BottomColor)

Parameters

[|param1|][explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[|todo|]

Related

[|todo: functionName, functionName|]
_IrrAddBillBoardGroupToScene

Adds a billboard group to the scene.

```cpp
#include <au3Irrlicht2.au3>
_IrrAddBillBoardGroupToScene()
```

Parameters

None.

Return Value

Success: Handle to an irr node object.
Failure: False and @error 1

Remarks

This is a special object that can have billboard like objects added and removed from it and rendered in a very quick an efficient manner. They are all treated as a single object rather than as many individual nodes. This is particuallarly useful for custom particle effects.

Related

_IrrAddBillBoardToGroup, _IrrRemoveBillBoardFromGroup, _IrrAddBillBoardByAxisToGroup, _IrrGetBillBoardGroupCount, IrrBillBoardGroupShadows, _IrrBillBoardForceUpdate

Example
#include "au3Irrlicht2.au3"

Global $hBillboardGroup
Global $hBillboardTexture
Global $aBillboard[6]
Global $hCamera

_IrrStart()

$hBillboardGroup = _IrrAddBillBoardGroupToScene()
For $i = 1 To 5
    $aBillboard[$i] = _IrrAddBillBoardToGroup($hBillboardGroup, 150, 100, 
        -450 + ($i * 150), 0, 400, 0, 255, 255, 255, 255)
Next
$hBillboardTexture = _IrrGetTexture("./media/au3irr2_logo.jpg")
_IrrSetNodeMaterialTexture($hBillboardGroup, $hBillboardTexture, 0)
_IrrSetNodeMaterialFlag($hBillboardGroup, $IRR_EMF_LIGHTING, 
    $IRR_OFF)

$hCamera = _IrrAddFPSCamera($IRR_NO_OBJECT, 100.0, 0.1)
_IrrSetCameraClipDistance($hCamera, 128000)

_IrrHideMouse()

While _IrrRunning() And Sleep(10)
    _IrrBeginScene(0, 0, 255)
    _IrrDrawScene()
    _IrrEndScene()
WEnd
This function adds a billboard to a billboard group. There are several properties that are used to specify the billboard.

```cpp
#include <au3Irrlicht2.au3>
_IrrAddBillBoardToGroup($h_BillboardGroup, $f_XSize, $f_YSize, $f_XPos, $f_YPos, $f_ZPos, $f_Roll, $u_Alpha, $u_Red, $u_Green, $u_Blue)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_BillboardGroup</td>
<td>Handle to the Billboard Group as returned by <code>_IrrAddBillBoardGroupToScene</code></td>
</tr>
<tr>
<td>$f_XSize</td>
<td>Width of the billboard</td>
</tr>
<tr>
<td>$f_YSize</td>
<td>Height of the billboard</td>
</tr>
<tr>
<td>$f_XPos</td>
<td>X position of the billboard</td>
</tr>
<tr>
<td>$f_YPos</td>
<td>Y position of the billboard</td>
</tr>
<tr>
<td>$f_ZPos</td>
<td>Z position of the billboard</td>
</tr>
<tr>
<td>$f_Roll</td>
<td>Specifies the number of degrees that the billboard is spun around its center</td>
</tr>
<tr>
<td>$u_Alpha</td>
<td>Alpha color used for the billboard 0 - 255</td>
</tr>
<tr>
<td>$u_Red</td>
<td>Red color used for the billboard 0 - 255</td>
</tr>
<tr>
<td>$u_Green</td>
<td>Green color used for the billboard 0 - 255</td>
</tr>
<tr>
<td>$u_Blue</td>
<td>Blue color used for the billboard 0 - 255</td>
</tr>
</tbody>
</table>

### Return Value

- **Success:** Handle to the billboard scene node address in the billboard group
- **Failure:** False and @error 1
Remarks
None

Related
_IrrAddBillBoardGroupToScene, _IrrRemoveBillBoardFromGroup,
_IrrGetBillBoardGroupCount, _IrrAddBillBoardByAxisToGroup,
_IrrBillBoardForceUpdate

Example

#include "au3Irrlicht2.au3"

Global $hBillboardGroup
Global $hBillboardTexture
Global $aBillboard[6]
Global $hCamera

_IrrStart()

$hBillboardGroup = _IrrAddBillBoardGroupToScene()
For $i = 1 To 5
    $aBillboard[$i] = _IrrAddBillBoardToGroup($hBillboardGroup, 150, 100,
    -450 + ($i * 150), 0, 400, 0, 255, 255, 255, 255)
Next
$hBillboardTexture = _IrrGetTexture("./media/au3irr2_logo.jpg")
_IrrSetNodeMaterialTexture($hBillboardGroup, $hBillboardTexture, 0)
_IrrSetNodeMaterialFlag($hBillboardGroup, $IRR_EMF_LIGHTING, $IRR_OFF)

$hCamera = _IrrAddFPSCamera($IRR_NO_OBJECT, 100.0, 0.1)
_IrrSetCameraClipDistance($hCamera, 128000)
_IrrHideMouse()

While _IrrRunning() And Sleep(10)
  _IrrBeginScene(0, 0, 255)
  _IrrDrawScene()
  _IrrEndScene()
WEnd
**_IrrAddBillBoardByAxisToGroup**

Adds a billboard to a billboard group that is fixed to a particular axis.

```c
#include <au3Irrlicht2.au3>
_ IrrAddBillBoardByAxisToGroup($h_BillboardGroup, $f_XSize, $f_YSize, $f_XPos, $f_YPos, $f_ZPos, $f_Roll, $u_Alpha, $u_Red, $u_Green, $u_Blue, $f_XAxis, $f_YAxis, $f_ZAxis)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_BillboardGroup</td>
<td>Handle to the Billboard Group as returned by _IrrAddBillBoardGroupToScene</td>
</tr>
<tr>
<td>$f_XSize</td>
<td>Width of the billboard</td>
</tr>
<tr>
<td>$f_YSize</td>
<td>Height of the billboard</td>
</tr>
<tr>
<td>$f_XPos</td>
<td>X position of the billboard</td>
</tr>
<tr>
<td>$f_YPos</td>
<td>Y position of the billboard</td>
</tr>
<tr>
<td>$f_ZPos</td>
<td>Z position of the billboard</td>
</tr>
<tr>
<td>$f_Roll</td>
<td>Specifies the number of degrees that the billboard is spun around its center</td>
</tr>
<tr>
<td>$u_Alpha</td>
<td>Alpha color used for the billboard 0 - 255</td>
</tr>
<tr>
<td>$u_Red</td>
<td>Red color used for the billboard 0 - 255</td>
</tr>
<tr>
<td>$u_Green</td>
<td>Green color used for the billboard 0 - 255</td>
</tr>
<tr>
<td>$u_Blue</td>
<td>Blue color used for the billboard 0 - 255</td>
</tr>
<tr>
<td>$f_XAxis</td>
<td>X direction around which the billboard is spun to face the camera.</td>
</tr>
<tr>
<td>$f_YAxis</td>
<td>Y direction around which the billboard is spun to face the camera.</td>
</tr>
<tr>
<td>$f_ZAxis</td>
<td>Z direction around which the billboard is spun to face the camera.</td>
</tr>
</tbody>
</table>
Return Value
Success: Handle to the billboard scene node address in the billboard group
Failure: False and @error 1

Remarks
These billboards are particularly useful for things like grass.
There are a number of properties that are used to specify the billboard.

Related
_IrrAddBillBoardGroupToScene, _IrrRemoveBillBoardFromGroup, _IrrGetBillBoardGroupCount, _IrrBillBoardForceUpdate

Example

```
#include "au3Irrlicht2.au"

Global $hBillboardGroup
Global $hBillboardTexture
Global $aBillboard[6]
Global $hCamera

_IrrStart()

$hBillboardGroup = _IrrAddBillBoardGroupToScene()
For $i = 1 To 5
    $aBillboard[$i] = _IrrAddBillBoardByAxisToGroup($hBillboardGroup, 150, 100, -450 + ($i * 150), 0, 400, 0, 255, 255, 255, 255, 255, 0, 1, 0)
Next
$hBillboardTexture = _IrrGetTexture("./media/au3irr2_logo.jpg")
_IrrSetNodeMaterialTexture($hBillboardGroup, $hBillboardTexture, 0)
_IrrSetNodeMaterialFlag($hBillboardGroup, $IRR_EMF_LIGHTING,
```

$IRR\_OFF$

$hCamera = _IrrAddFPSCamera($IRR\_NO\_OBJECT, 100.0, 0.1)
_IrrSetCameraClipDistance($hCamera, 128000)

_IrrHideMouse()

While _IrrRunning() And Sleep(10)
  _IrrBeginScene(0, 0, 255)
  _IrrDrawScene()
  _IrrEndScene()
WEnd
_IrrRemoveBillBoardFromGroup

Removes the specified billboard from the billboard group.

```au3
#include <au3Irrlicht2.au3>
_IrrRemoveBillBoardFromGroup($h_BillboardGroup, $h_BillboardGroupSceneNode)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_BillboardGroup</td>
<td>Handle to the Billboard Group as returned by <code>_IrrAddBillBoardGroupToScene</code></td>
</tr>
<tr>
<td>$h_BillboardGroupSceneNode</td>
<td>Handle to the billboard scene node address in the billboard group as returned by <code>_IrrAddBillBoardToGroup</code></td>
</tr>
</tbody>
</table>

**Return Value**

Success: True  
Failure: False

**Remarks**

None.

**Related**

`_IrrAddBillBoardGroupToScene`, `_IrrAddBillBoardToGroup`

**Example**
#include "au3Irrlicht2.au3"

Global $hBillboardGroup
Global $hBillboardTexture
Global $aBillboard[6]
Global $iBillBoards
Global $hCamera
Global $iTimer = TimerInit()

_IrrStart()

$hBillboardGroup = _IrrAddBillBoardGroupToScene()
For $i = 1 To 5
    $aBillboard[$i] = _IrrAddBillBoardToGroup($hBillboardGroup, 150, 100, -450 + ($i * 150), 0, 400, 0, 255, 255, 255)
Next
$hBillboardTexture = _IrrGetTexture("./media/au3irr2_logo.jpg")
_IrrSetNodeMaterialTexture($hBillboardGroup, $hBillboardTexture, 0)
_IrrSetNodeMaterialFlag($hBillboardGroup, $IRR_EMF_LIGHTING, $IRR_OFF)

$hCamera = _IrrAddFPSCamera($IRR_NO_OBJECT, 100.0, 0.1)
_IrrSetCameraClipDistance($hCamera, 128000)

_IrrHideMouse()

While _IrrRunning() And Sleep(10)
    $iBillBoards = _IrrGetBillBoardGroupCount($hBillboardGroup)
    If TimerDiff($iTimer) > 2000 And $iBillBoards > 0 Then
        _IrrRemoveBillBoardFromGroup($hBillboardGroup, $aBillboard[$iBillBoards])
        $iTimer = TimerInit()
    EndIf
    _IrrBeginScene(0, 0, 255)
    _IrrDrawScene()
    _IrrEndScene()
_IrrBillBoardGroupShadows

Applies lighting to the billboards in a cluster of billboards.

```
#include <au3Irrlicht2.au3>
_IrrBillBoardGroupShadows($h_BillboardGroup, $f_X, $f_Y, $f_Z, $f_Intensity, $f_Ambient)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_BillboardGroup</td>
<td>Handle to the Billboard Group as returned by _IrrAddBillBoardGroupToScene</td>
</tr>
<tr>
<td>$f_X</td>
<td>X direction from which the light is arriving.</td>
</tr>
<tr>
<td>$f_Y</td>
<td>Y direction from which the light is arriving.</td>
</tr>
<tr>
<td>$f_Z</td>
<td>Z direction from which the light is arriving.</td>
</tr>
<tr>
<td>$f_Intensity</td>
<td>Strength of the light</td>
</tr>
<tr>
<td>$f_Ambient</td>
<td>Strength of ambient light in the billboard group</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True  
Failure: False

**Remarks**

This can be used for example to shade the particles in a group of billboards representing a cloud.

**Related**

_IrrAddBillBoardGroupToScene, _IrrAddBillBoardToGroup
Example

```
#include "au3Irrlicht2.au3"

Global $hBillboardGroup
Global $hBillboardTexture
Global $aBillboard[6]
Global $hCamera

_IrrStart()

$hBillboardGroup = _IrrAddBillBoardGroupToScene()
For $i = 1 To 5
    $aBillboard[$i] = _IrrAddBillBoardToGroup($hBillboardGroup, 150, 100, -450 + ($i * 150), 0, 400, 0, 255, 255, 255, 255)
Next
_IrrBillBoardGroupShadows($hBillboardGroup, 1, 0, 0, 2.0, 1.7)
$hBillboardTexture = _IrrGetTexture("./media/au3irr2_logo.jpg")
_IrrSetNodeMaterialTexture($hBillboardGroup, $hBillboardTexture, 0)
_IrrSetNodeMaterialFlag($hBillboardGroup, $IRR_EMF_LIGHTING, $IRR_OFF)

$hCamera = _IrrAddFPSCamera($IRR_NO_OBJECT, 100.0, 0.1)
_IrrSetCameraClipDistance($hCamera, 128000)

_IrrHideMouse()

While _IrrRunning() And Sleep(10)
    _IrrBeginScene(0, 0, 255)
    _IrrDrawScene()
    _IrrEndScene()
WEnd
```
_IrrGetBillBoardGroupCount

Get the number of billboards in the billboard group.

```
#include <au3Irrlicht2.au3>
_IrrGetBillBoardGroupCount($h_BillboardGroup)
```

Parameters

- `$h_BillboardGroup`: Handle to the Billboard Group as returned by `_IrrAddBillBoardGroupToScene`

Return Value

Success: The number of billboards in the billboard group
Failure: False and @error 1

Remarks

None.

Related

- `_IrrAddBillBoardGroupToScene`
- `_IrrAddBillBoardToGroup`
- `_IrrRemoveBillBoardFromGroup`

Example

```
#include "au3Irrlicht2.au3"
```
Global $hBillboardGroup
Global $hBillboardTexture
Global $aBillboard[6]
Global $hCamera

_IrrStart()

$hBillboardGroup = _IrrAddBillBoardGroupToScene()
For $i = 1 To 5
   $aBillboard[$i] = _IrrAddBillBoardToGroup($hBillboardGroup, 150, 100, -450 + ($i * 150), 0, 400, 0, 255, 255, 255, 255)
Next
$hBillboardTexture = _IrrGetTexture("./media/au3irr2_logo.jpg")
_IrrSetNodeMaterialTexture($hBillboardGroup, $hBillboardTexture, 0)
_IrrSetNodeMaterialFlag($hBillboardGroup, $IRR_EMF_LIGHTING, $IRR_OFF)

_IrrSetWindowCaption(_IrrGetBillBoardGroupCount($hBillboardGroup) & " Billboards in the Billboard Group")

$hCamera = _IrrAddFPSCamera($IRR_NO_OBJECT, 100.0, 0.1)
_IrrSetCameraClipDistance($hCamera, 128000)

_IrrHideMouse()

While _IrrRunning() And Sleep(10)
   _IrrBeginScene(0, 0, 255)
   _IrrDrawScene()
   _IrrEndScene()
WEnd

Copy to Clipboard
**au3Irr2 Function Reference**

**_IrrBillBoardForceUpdate**

Force the billboard group update the scene. (see remarks for further explanation)

```plaintext
#Include <au3Irrlicht2.au3>
_IrrBillBoardForceUpdate($h_BillboardGroup)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_BillboardGroup</td>
<td>Handle to the Billboard Group as returned by _IrrAddBillBoardGroupToScene</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True  
Failure: False

**Remarks**

Unlike regular billboards the billboard group does not always update the orientation of the billboards every frame.  
If you are a long distance away from the billboard group the camera needs to travel a significant distance before the angle has changed enough to warrant an update of all of the billboards vertices to make them point to the camera once more.  
You may want to force a refresh at some point with this call.

**Related**

_IrrAddBillBoardGroupToScene, _IrrAddBillBoardToGroup,  
_IrrAddBillBoardByAxisToGroup, _IrrRemoveBillBoardFromGroup
_IrrAddLODManager

Adds a level of detail manager to the scene.

```
#include <au3Irrlicht2.au3>
_IrrAddLODManager($u_FadeScale, $u_UseAlpha[, $p_Callback = 0])
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_FadeScale</td>
<td>Number of 1/4 seconds that the node takes to fade out or in. 4 units equals 1 second.</td>
</tr>
<tr>
<td>$u_UseAlpha</td>
<td>Specifies whether or not the Alpha color of the object is faded too.</td>
</tr>
<tr>
<td>$p_Callback</td>
<td><strong>[optional]</strong> Register a callback function that is called whenever a node is made invisible or visible. This allows you to stop processing hidden nodes.</td>
</tr>
</tbody>
</table>

### Return Value

Success: Handle to the LOD Manager node
Failure: False and @error 1

### Remarks

The primary use for this node is to add other scene nodes to it as children and have their level of detail controlled automatically. If those nodes are made from loaded meshes different meshes containing different amounts of detail can be displayed at different distances. The other function of the LOD manager is to fade nodes in an out at a specific distance so they gradually fade rather than disappear abruptly. This is achieved by applying a distance without supplying a mesh.
**Related**

_`IrrAddLODMesh`, `_IrrSetLODMaterialMap`

**Example**

```c
#include "au3Irrlicht2.au3"

Global $ROWS_AND_COLUMNS = 20
Global $hLOD1Mesh
Global $hLOD2Mesh
Global $hMeshTexture
Global $iAmountNodes = $ROWS_AND_COLUMNS * $ROWS_AND_COLUMNS
Global $aSceneNodes[$iAmountNodes]
Global $hMaterial
Global $k = 0
Global $hLODManager
Global $hCamera

_IrrStart()

$hLOD1Mesh = _IrrGetMesh("./media/cylinderY.obj")
$hLOD2Mesh = _IrrGetMesh("./media/cylinderYLow.obj")
_IrrScaleMesh($hLOD1Mesh, 8.0)
_IrrScaleMesh($hLOD2Mesh, 8.0)
_IrrSetMeshHardwareAccelerated($hLOD1Mesh)
_IrrSetMeshHardwareAccelerated($hLOD2Mesh)
$hMeshTexture = _IrrGetTexture("./media/Cross.bmp")
For $i = -($(ROWS_AND_COLUMNS / 2) To $(ROWS_AND_COLUMNS / 2) - 1)
    For $j = -($(ROWS_AND_COLUMNS / 2) To $(ROWS_AND_COLUMNS / 2) - 1)
        $aSceneNodes[$k] = _IrrAddMeshToScene($hLOD1Mesh)
        _IrrSetNodePosition($aSceneNodes[$k], $i * 40.0, 0.0, $j * 40.0)
```

_IrrSetNodeMaterialTexture($aSceneNodes[$k], $hMeshTexture, 0)
_IrrSetNodeMaterialFlag($aSceneNodes[$k], $IRR_EMF_LIGHTING, $IRR_ON)

$hMaterial = _IrrGetMaterial($aSceneNodes[$k], 0)
_ImaterialVertexColorAffects($hMaterial, $ECM_NONE)
_ImaterialSetAmbientColor($hMaterial, 255, 255, 255, 255)
_ImaterialSetDiffuseColor($hMaterial, 255, 255, 255, 255)
$k += 1

Next

$hLODManager = _IrrAddLODManager(2, $IRR_ON)
_ImaterialSetLODMaterialMap($hLODManager, $IRR_EMT_TRANSPARENT_ADD_COLOR, $IRR_EMT_TRANSPARENT_ADD_COLOR)
_ImaterialAddLODMesh($hLODManager, 0.0, $hLOD1Mesh)
_ImaterialAddLODMesh($hLODManager, 300.0, $hLOD2Mesh)
_ImaterialAddLODMesh($hLODManager, 500.0, $IRR_NO_OBJECT)
_ImaterialSetNodeMaterialFlag($hLODManager, $IRR_EMF_LIGHTING, $IRR_OFF)

For $i = 0 To $k - 1
   _IrrAddChildToParent($aSceneNodes[$i], $hLODManager)
Next

$hCamera = _IrrAddFPSCamera($IRR_NO_OBJECT, 100.0, 0.05)
_ImaterialSetNodePosition($hCamera, 0, $ROWS_AND_COLUMNS * 4, $ROWS_AND_COLUMNS * 2)
_ImaterialSetCameraTarget($hCamera, 0, 50, 0)
_ImaterialSetCameraClipDistance($hCamera, 2500.0)

_ImaterialSetAmbientLight(1, 1, 1)

_ImaterialHideMouse()
_IrrAddLODMesh

Set the distance at which a particular mesh is to be applied to child mesh nodes.

```cpp
#include <au3Irrlicht2.au3>
_IrrAddLODMesh($h_LODManager, $f_Distance, $h_Mesh)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_LODManager</td>
<td>Handle to the LOD Manager node.</td>
</tr>
<tr>
<td>$f_Distance</td>
<td>Distance at which this effect will be applied.</td>
</tr>
<tr>
<td>$h_Mesh</td>
<td>Handle to an irr mesh object</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True  
Failure: False

**Remarks**

If no mesh is supplied it specifies the distance at which the node should be faded in and out.

**Related**

_ IrrAddLODManager, _IrrSetLODMaterialMap

**Example**
#include "au3Irrlicht2.au3"

Global $ROWS_AND_COLUMNS = 20
Global $hLOD1Mesh
Global $hLOD2Mesh
Global $hMeshTexture
Global $iAmountNodes = $ROWS_AND_COLUMNS * $ROWS_AND_COLUMNS
Global $aSceneNodes[$iAmountNodes]
Global $hMaterial
Global $k = 0
Global $hLODManager
Global $hCamera

_IrrStart()

$hLOD1Mesh = _IrrGetMesh("./media/cylinderY.obj")
$hLOD2Mesh = _IrrGetMesh("./media/cylinderYLow.obj")
_IrrScaleMesh($hLOD1Mesh, 8.0)
_IrrScaleMesh($hLOD2Mesh, 8.0)
_IrrSetMeshHardwareAccelerated($hLOD1Mesh)
_IrrSetMeshHardwareAccelerated($hLOD2Mesh)
$hMeshTexture = _IrrGetTexture("./media/Cross.bmp")
For $i = -( $ROWS_AND_COLUMNS / 2 ) To ( $ROWS_AND_COLUMNS / 2 ) - 1
    For $j = -( $ROWS_AND_COLUMNS / 2 ) To ( $ROWS_AND_COLUMNS / 2 ) - 1
        $aSceneNodes[$k] = _IrrAddMeshToScene($hLOD1Mesh)
        _IrrSetNodePosition($aSceneNodes[$k], $i * 40.0, 0.0, $j * 40.0)
        _IrrSetNodeMaterialTexture($aSceneNodes[$k], $hMeshTexture, 0)
        _IrrSetNodeMaterialFlag($aSceneNodes[$k], $IRR_EMF_LIGHTING, $IRR_ON)
        $hMaterial = _IrrGetMaterial($aSceneNodes[$k], 0)
        _IrrMaterialVertexColorAffects($hMaterial, $ECM_NONE)
        _IrrMaterialSetAmbientColor($hMaterial, 255, 255, 255, 255)
        _IrrMaterialSetDiffuseColor($hMaterial, 255, 255, 255, 255)
        $k += 1
$hLODManager = _IrrAddLODManager(2, $IRR_ON)
_IrrSetLODMaterialMap($hLODManager, $IRR_EMT_TRANSPARENT_ADD_COLOR, $IRR_EMT_TRANSPARENT_ADD_COLOR)
_IrrAddLODMesh($hLODManager, 0.0, $hLOD1Mesh)
_IrrAddLODMesh($hLODManager, 300.0, $hLOD2Mesh)
_IrrAddLODMesh($hLODManager, 500.0, $IRR_NO_OBJECT)
_IrrSetNodeMaterialFlag($hLODManager, $IRR_EMF_LIGHTING, $IRR_OFF)
For $i = 0 To $k - 1
  _IrrAddChildToParent($aSceneNodes[$i], $hLODManager)
Next

$hCamera = _IrrAddFPSCamera($IRR_NO_OBJECT, 100.0, 0.05)
_IrrSetNodePosition($hCamera, 0, $ROWS_AND_COLUMNS * 4, $ROWS_AND_COLUMNS * 2)
_IrrSetCameraTarget($hCamera, 0, 50, 0)
_IrrSetCameraClipDistance($hCamera, 2500.0)

_IrrSetAmbientLight(1, 1, 1)
_IrrHideMouse()

While _IrrRunning() And Sleep(10)
  _IrrBeginScene(0, 0, 0)
  _IrrDrawScene()
  _IrrEndScene()
End

_IrrStop()
_IrrSetLODMaterialMap

Specifies which material is used to apply the fade effect for another material type.

```plaintext
#Include <au3Irrlicht2.au3>
_IrrSetLODMaterialMap($h_LODManager, $i_SourceType, $i_TargetType)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_LODManager</td>
<td>Handle to the LOD Manager node.</td>
</tr>
<tr>
<td>$u_SourceType</td>
<td>The irr material type your node uses</td>
</tr>
<tr>
<td>$u_TargetType</td>
<td>The material type used for the fade effect.</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True
Failure: False

**Remarks**

How this is used will depend on the effect that you want to achieve. By default fading is applied with the $IRR_EMT_TRANSPARENT_VERTEX_ALPHA material.

**Related**

_IrrAddLODManager, _IrrAddLODMesh

**Example**
#include "au3Irrlicht2.au3"

Global $ROWS_AND_COLUMNS = 20  
Global $hLOD1Mesh  
Global $hLOD2Mesh  
Global $hMeshTexture  
Global $iAmountNodes = $ROWS_AND_COLUMNS * $ROWS_AND_COLUMNS  
Global $aSceneNodes[$iAmountNodes]  
Global $hMaterial  
Global $k = 0  
Global $hLODManager  
Global $hCamera

_IrrStart()

$hLOD1Mesh = _IrrGetMesh("./media/cylinderY.obj")  
$hLOD2Mesh = _IrrGetMesh("./media/cylinderYLow.obj")  
_IrrScaleMesh($hLOD1Mesh, 8.0)  
_IrrScaleMesh($hLOD2Mesh, 8.0)  
_IrrSetMeshHardwareAccelerated($hLOD1Mesh)  
_IrrSetMeshHardwareAccelerated($hLOD2Mesh)  
$hMeshTexture = _IrrGetTexture("./media/Cross.bmp")  
For $i = -(ROWS_AND_COLUMNS / 2) To (ROWS_AND_COLUMNS / 2) - 1  
For $j = -(ROWS_AND_COLUMNS / 2) To (ROWS_AND_COLUMNS / 2) - 1  
  $aSceneNodes[$k] = _IrrAddMeshToScene($hLOD1Mesh)  
  _IrrSetNodePosition($aSceneNodes[$k], $i * 40.0, 0.0, $j * 40.0)  
  _IrrSetNodeMaterialTexture($aSceneNodes[$k], $hMeshTexture, 0)  
  _IrrSetNodeMaterialFlag($aSceneNodes[$k], $IRR_EMF_LIGHTING, $IRR_ON)  
  $hMaterial = _IrrGetMaterial($aSceneNodes[$k], 0)  
  _IrrMaterialVertexColorAffects($hMaterial, $ECM_NONE)  
  _IrrMaterialSetAmbientColor($hMaterial, 255, 255, 255, 255)  
  _IrrMaterialSetDiffuseColor($hMaterial, 255, 255, 255, 255)
$k += 1
Next

$hLODManager = _IrrAddLODManager(2, $IRR_ON)
_IrrSetLODMaterialMap($hLODManager, $IRR_EMT_TRANSPARENT_ADD_COLOR, $IRR_EMT_TRANSPARENT_ADD_COLOR)
_IrrAddLODMesh($hLODManager, 0.0, $hLOD1Mesh)
_IrrAddLODMesh($hLODManager, 300.0, $hLOD2Mesh)
_IrrAddLODMesh($hLODManager, 500.0, $IRR_NO_OBJECT)
_IrrSetNodeMaterialFlag($hLODManager, $IRR_EMF_LIGHTING, $IRR_OFF)
For $i = 0 To $k - 1
    _IrrAddChildToParent($aSceneNodes[$i], $hLODManager)
Next

$hCamera = _IrrAddFPSCamera($IRR_NO_OBJECT, 100.0, 0.05)
_IrrSetNodePosition($hCamera, 0, $ROWS_AND_COLUMNS * 4, $ROWS_AND_COLUMNS * 2)
_IrrSetCameraTarget($hCamera, 0, 50, 0)
_IrrSetCameraClipDistance($hCamera, 2500.0)

_IrrSetAmbientLight(1, 1, 1)

_IrrHideMouse()

While _IrrRunning() And Sleep(10)
    _IrrBeginScene(0, 0, 0)
    _IrrDrawScene()
    _IrrEndScene()
WEnd

_IrrStop()
au3Irr2 Function Reference

_IrrStart

Opens the IrrlichtWrapper.dll, starts Irrlicht interface and opens a window for rendering.

#include <au3Irrlicht2.au3>
_IrrStart($i_DeviceType=$IRR_EDT_DIRECT3D9, $i_ScreenWidth=800, $i_ScreenHeight=600, $i_BitsPerPixel=$IRR_BITS_PER_PIXEL_32, $b_FullScreen=$IRR_WINDOWED, $b_Shadows=$IRR_NO_SHADOWS, $b_InputCapture=$IRR_IGNORE_EVENTS, $b_VSync=$IRR_VERTICAL_SYNC_OFF)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_DeviceType</td>
<td>[optional] specifies the renderer to use when drawing to the display this may be one of the following types: $IRR_EDT_NULL - A NULL device with no display $IRR_EDT_SOFTWARE - Irrlichts default software renderer $IRR_EDT_SOFTWARE2 - An improved quality software renderer $IRR_EDT_OPENGL - Hardware accelerated OpenGL renderer $IRR_EDT_DIRECT3D8 - Hardware accelerated DirectX 8 renderer $IRR_EDT_DIRECT3D9 - Hardware accelerated DirectX 9 renderer</td>
</tr>
<tr>
<td>$i_ScreenWidth</td>
<td>[optional] Screen width specifies the width of the viewport in pixels</td>
</tr>
<tr>
<td>$i_ScreenHeight</td>
<td>[optional] Screen height specifies the height of the viewport in pixels</td>
</tr>
<tr>
<td></td>
<td>[optional] The number of color bits that is used for each pixel 32 bit color gives 24 million different colors</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$i_{\text{BitsPerPixel}}$</td>
<td>whereas 16 bit color gives only 32,000 colors. However the advantage of 16 bit color is that some operations use half the memory and can run at up to twice the speed. This setting can be either of: $\text{IRR_BITS_PER_PIXEL_16}$ $\text{IRR_BITS_PER_PIXEL_32}$</td>
</tr>
<tr>
<td>$b_{\text{FullScreen}}$</td>
<td>[optional] Specifies whether the display is to opened in full screen mode or in a window: $\text{IRR_WINDOWED}$ - For window mode $\text{IRR_FULLSCREEN}$ - For fullscreen mode. When using full screen mode you will need to adjust the window size to the same dimensions as a supported screen resolution on the target display 640x400 for example.</td>
</tr>
<tr>
<td>$b_{\text{Shadows}}$</td>
<td>[optional] Use shadows starts the engine in a mode that supports the rendering of stencil shadows. $\text{IRR_NO_SHADOWS}$ - For a display that does not support shadows. $\text{IRR_SHADOWS}$ - For a display that supports shadows.</td>
</tr>
<tr>
<td>$b_{\text{InputCapture}}$</td>
<td>[optional] Capture mouse and keyboard specified whether you want to capture keyboard and mouse events, if you choose to ignore them they will be handled by Irrlicht for FPS camera control. This parameter should be either of: $\text{IRR_IGNORE_EVENTS}$ $\text{IRR_CAPTURE_EVENTS}$</td>
</tr>
<tr>
<td>$b_{\text{VSync}}$</td>
<td>[optional] Vertical synchronisation specifies whether the display of each new frame is syncronised with vertical refresh of the graphics card. This produces a smoother display and avoids 'tearing' where the viewer can see parts of two different frames at the same time. The setting can be either of: $\text{IRR_VERTICAL_SYNC_OFF}$ $\text{IRR_VERTICAL_SYNC_ON}$</td>
</tr>
</tbody>
</table>
Return Value
Success: True
Failure: False and sets @error:
1 - error occurred on dll call
2 - IrrlichtWrapper.dll not found

Remarks
if .dll cannot be opened, path environment is extended with:
- .\bin (allows an au3Irr2 script to have its binaries in a subdir) and
- .. (allows au3Irr2 examples to be started from their \include subdir).
Nevertheless, ensuring DLL's in a permanent dir reachable via path may
be the better way, as the temporary update of environment can be time-
consuming.

Other needed .dll's (Irrlicht.dll + maybe msvcp71.dll, msvcr71.dll) are
NOT checked but simply expected to be at last in same dir as the
IrrlichtWrapper.dll.

Related
_IrrStartAdvanced, _IrrRunning, _IrrStop

Example

#include "au3Irrlicht2.au3"

_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _
         $IRR_WINDOWED, $IRR_SHADOWS, $IRR_CAPTURE_EVENTS, $IRR_VERTICAL_SYNC_ON )

local $Camera = _IrrAddCamera(10,10,10, 0,0,0 )
local $testNode = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $testNode,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $testNode, $IRR_EMF_LIGHTING, $IRR_OFF )

WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 0)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
Opens the IrrlichtWrapper.dll and starts Irrlicht engine with advanced method.

```au3
#include <au3Irrlicht2.au3>
_IrrStartAdvanced($i_DeviceType=$IRR_EDT_DIRECT3D9,
  $i_ScreenWidth=800, $i_ScreenHeight=600,
  $i_BitsPerPixel=$IRR_BITS_PER_PIXEL_32,
  $b_FullScreen=$IRR_WINDOWED, $b_Shadows=$IRR_NO_SHADOWS,
  $b_InputCapture=$IRR_IGNORE_EVENTS,
  $b_VSync=$IRR_VERTICAL_SYNC_OFF, $i_TypeOfDevice=0,
  $b_DoublebufferEnabled=$IRR_OFF, $i_AntialiasEnabled=0,
  $b_HighPrecisionFpu=$IRR_OFF)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$i_DeviceType</code></td>
<td>[optional] specifies the renderer to use when drawing to the display this may be one of the following types:</td>
</tr>
<tr>
<td></td>
<td>$IRR_EDT_NULL - A NULL device with no display</td>
</tr>
<tr>
<td></td>
<td>$IRR_EDT_SOFTWARE - Irrlichts default software renderer</td>
</tr>
<tr>
<td></td>
<td>$IRR_EDT_SOFTWARE2 - An improved quality software renderer</td>
</tr>
<tr>
<td></td>
<td>$IRR_EDT_OPENGL - Hardware accelerated OpenGL renderer</td>
</tr>
<tr>
<td></td>
<td>$IRR_EDT_DIRECT3D8 - Hardware accelerated DirectX 8 renderer</td>
</tr>
<tr>
<td></td>
<td>$IRR_EDT_DIRECT3D9 - Hardware accelerated DirectX 9 renderer</td>
</tr>
<tr>
<td><code>$i_ScreenWidth</code></td>
<td>[optional] Screen width specifies the width of the viewport in pixels</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$i_{\text{ScreenHeight}}$</td>
<td>[optional] Screen height specifies the height of the viewport in pixels</td>
</tr>
<tr>
<td>$i_{\text{BitsPerPixel}}$</td>
<td>[optional] The number of color bits that is used for each pixel 32 bit color gives 24 million different colors whereas 16 bit color gives only 32,000 colors. However the advantage of 16 bit color is that some operations use half the memory and can run at up to twice the speed. This setting can be either of: $\text{IRR_BITS_PER_PIXEL_16}$, $\text{IRR_BITS_PER_PIXEL_32}$</td>
</tr>
<tr>
<td>$b_{\text{FullScreen}}$</td>
<td>[optional] Specifies whether the display is to opened in full screen mode or in a window: $\text{IRR_WINDOWED}$ - For window mode $\text{IRR_FULLSCREEN}$ - For fullscreen mode. When using full screen mode you will need to adjust the window size to the same dimensions as a supported screen resolution on the target display 640x400 for example.</td>
</tr>
<tr>
<td>$b_{\text{Shadows}}$</td>
<td>[optional] Use shadows starts the engine in a mode that supports the rendering of stencil shadows. $\text{IRR_NO_SHADOWS}$ - For a display that does not support shadows. $\text{IRR_SHADOWS}$ - For a display that supports shadows.</td>
</tr>
<tr>
<td>$b_{\text{InputCapture}}$</td>
<td>[optional] Capture mouse and keyboard specified whether you want to capture keyboard and mouse events, if you choose to ignore them they will be handled by Irrlicht for FPS camera control. This parameter should be either of: $\text{IRR_IGNORE_EVENTS}$, $\text{IRR_CAPTURE_EVENTS}$</td>
</tr>
<tr>
<td></td>
<td>[optional] Vertical syncronisation specifies whether the display of each new frame is</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$b_{\text{VSync}}$</td>
<td>syncronised with vertical refresh of the graphics card. This produces a smoother display and avoids 'tearing' where the viewer can see parts of two different frames at the same time. The setting can be either of: $\text{IRR}<em>{\text{VERTICAL SYNCH OFF}}$  $\text{IRR}</em>{\text{VERTICAL SYNCH ON}}$</td>
</tr>
<tr>
<td>$i_{\text{TypeOfDevice}}$</td>
<td>[optional] Devicetype allows a specific type of device for example a windows screen or a console to be selected. For the time being this should be set to 0 which automatically selects the best device.</td>
</tr>
<tr>
<td>$b_{\text{DoublebufferEnabled}}$</td>
<td>[optional] Doublebufferenabled is used to control whether double buffering is used. When double buffering is used two drawing surfaces are created one for display and the other that is used for drawing too. Double buffering is required for anit-aliasing the options are: $\text{IRR}<em>{\text{ON}}$ or $\text{IRR}</em>{\text{OFF}}$</td>
</tr>
<tr>
<td>$i_{\text{AntialiasEnabled}}$</td>
<td>[optional] Antialiasenabled is used to enable the antialiasing effect, this effect produces a blurring at the edges of object giving their lines a smooth natural appearence. There is usually a big penalty for using this effect though sometimes as high as 30% of the frame rate or more. This is a value for the anti-aliasing and should be a power of 2. (e.g: 2, 4, 8, 16)</td>
</tr>
<tr>
<td>$b_{\text{HighPrecisionFpu}}$</td>
<td>[optional] Highprecisionfpu is used to enable high precision Floating point calculations, that produce more accurate result at the expense of a slower operating speed.</td>
</tr>
</tbody>
</table>
Failure: False and sets @error:
1 - error occurred on dll call
2 - IrrlichtWrapper.dll not found

Remarks
if .dll cannot be opened, path environment is extended with:
- .bin (allows an au3Irr2 script to have its binaries in a subdir) and
- .. (allows au3Irr2 examples to be started from their \include subdir).
Nevertheless, ensuring DLL's in a permanent dir reachable via path may be the better way, as the temporary update of environment can be time-consuming.

Other needed .dll's (Irrlicht.dll + maybe msvcp71.dll, msvcr71.dll) are NOT checked but simply expected to be at last in same dir as the IrrlichtWrapper.dll.

Related
_IrrStart, _IrrRunning, _IrrStop

Example

```
#include "au3Irrlicht2.au3"

_IrrStartAdvanced ( $IRR_EDT_OPENGL, 800, 600,
$IRR_BITS_PER_PIXEL_32, _
   $IRR_WINDOWED, $IRR_SHADOWS, $IRR_CAPTURE_EVENTS,
$IRR_VERTICAL_SYNC_ON, _
   0, $IRR_ON, 4, $IRR_ON)

local $Camera = _IrrAddCamera(10,10,10, 0,0,0 )
local $testNode = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture($testNode,
   _IrrGetTexture(".\media\au3irr2_logo.jpg", 0))
```
_IrrSetNodeMaterialFlag($testNode, $IRR_EMF_LIGHTING, $IRR_OFF)

WHILE _IrrRunning()
   _IrrBeginScene(0, 0, 0)
   _IrrDrawScene()
   _IrrEndScene()
WEND

_IrrStop()
_IrrRunning

Used to determine if the Irrlicht engine is still running.

```
#include <au3Irrlicht2.au3>
_IrrRunning()
```

**Parameters**

<table>
<thead>
<tr>
<th>None.</th>
</tr>
</thead>
</table>

**Return Value**

True if running: False if not.  
Sets @error to true on failures.

**Remarks**

None

**Related**

_ IrrStart, _IrrStartAdvanced, _IrrStop

**Example**

```
#include "au3Irrlicht2.au3"

_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _ 
$IRR_WINDOWED, $IRR_SHADOWS, $IRR_CAPTURE_EVENTS, 
```
$IRR_VERTICAL_SYNC_ON

local $Camera = _IrrAddCamera(10,10,10,0,0,0)
local $testNode = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $testNode,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $testNode, $IRR_EMF_LIGHTING, $IRR_OFF )

WHILE _IrrRunning()
    _IrrBeginScene(0,0,0)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
_IrrSetViewPort

[todo]

#include <au3Irrlicht2.au3>
_IrrSetViewPort($i_TopX, $i_TopY, $i_BottomX, $i_BottomY)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
  moreExplanationIndented

Remarks

[todo]

Related

todo: functionName, functionName
_IrrBeginScene

Starts to draw a frame, erasing the canvas with the specified color.

```
#Include <au3Irrlicht2.au3>
_IrrBeginScene($i_Red, $i_Green, $i_Blue)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Red</td>
<td>Red value from 0 to 255 (full intensity)</td>
</tr>
<tr>
<td>$i_Green</td>
<td>Green value from 0 to 255 (full intensity)</td>
</tr>
<tr>
<td>$i_Blue</td>
<td>Blue value from 0 to 255 (full intensity)</td>
</tr>
</tbody>
</table>

**Return Value**

Success: True
Failure: False

**Remarks**
None

**Related**

_`IrrBeginSceneAdvanced`, _`IrrDrawScene`, _`IrrDrawGUI`, _`IrrEndScene`

**Example**

```
#include "au3Irrlicht2.au3"
```
local $Camera = _IrrAddCamera(10,10,10,0,0)  
local $testNode = _IrrAddTestSceneNode()  
_IrrSetNodeMaterialTexture($testNode, _IrrGetTexture("./media\au3irr2_logo.jpg"), 0)  
_IrrSetNodeMaterialFlag($testNode, $IRR_EMF_LIGHTING, $IRR_OFF)  

WHILE _IrrRunning()  
 _IrrBeginScene(0, 0, 0)  
 _IrrDrawScene()  
 _IrrEndScene()  
WEND  

_IrrStop()
_IrrBeginSceneAdvanced

[todo]

#include <au3Irrlicht2.au3>
_IIrrBeginSceneAdvanced($i_SceneBGColor, $b_ClearBackBuffer = $IRR_ON, $b_ClearZBuffer = $IRR_ON)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

todo

Related

todo: functionName, functionName
_IrrDrawScene

This renders the 3D scene to the canvas, drawing all 3D elements: nodes, particles, billboards, etc.

```c
#include <au3Irrlicht2.au3>
_IrrDrawScene()
```

**Parameters**

None.

**Return Value**

Success: True
Failure: False

**Remarks**

None

**Related**

_IrrBeginScene, _IrrBeginSceneAdvanced, _IrrDrawGUI, _IrrEndScene

**Example**

```c
#include "au3Irrlicht2.au3"

_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _
```
$IRR\_WINDOWED, \$IRR\_SHADOWS, \$IRR\_CAPTURE\_EVENTS, 
\$IRR\_VERTICAL\_SYNC\_ON )

local $Camera = _IrrAddCamera(10, 10, 10, 0, 0, 0 )
local $testNode = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $testNode,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $testNode, \$IRR\_EMF\_LIGHTING, \$IRR\_OFF )

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 0)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
_IrrDrawSceneToTexture

[todo]

#include <au3Irrlicht2.au3>
_IrrDrawSceneToTexture($h_RenderTargetTexture)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetRenderTarget

[todo]

#Include <au3Irrlicht2.au3>
_IrrSetRenderTarget($h_Texture, $i_SceneBGColor = 0, $b_ClearBackBuffer = $IRR_ON, $b_ClearZBuffer = $IRR_ON)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrDrawGUI

This renders the 2D graphical user interface that has been created to the scene.

```
#include <au3Irrlicht2.au3>
_IrrDrawGUI()
```

### Parameters

None.

### Return Value

Success: True
Failure: False

### Remarks

None

### Related

_\_IrrBeginScene, _IrrBeginSceneAdvanced, _IrrDrawScene, _IrrEndScene

### Example

```
#include "au3Irrlicht2.au3"
```
_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _
  $IRR_WINDOWED, $IRR_SHADOWS, $IRR_CAPTURE_EVENTS, _
  $IRR_VERTICAL_SYNC_ON )

_IrrAddStaticText( "Hello AU3-World ;-)", 8, 8, 200, 20, _
  $IRR_GUI_NO_BORDER, $IRR_GUI_NO_WRAP)

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 0)
  _IrrDrawGUI()
  _IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
_IrrEndScene

This renders the 3D scene to the canvas, drawing all 3D elements: nodes, particles, billboards, etc.

#include <au3Irrlicht2.au3>
_IrrEndScene()

Parameters

None.

Return Value

Success: True
Failure: False

Remarks

None

Related

_IrrBeginScene, _IrrBeginSceneAdvanced, _IrrDrawScene, _IrrDrawGUI

Example

#include "au3Irrlicht2.au3"

_IrrStart( $IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32,
local $Camera = _IrrAddCamera(10,10,10, 0,0,0 )
local $testNode = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $testNode,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $testNode, $IRR_EMF_LIGHTING, $IRR_OFF )

WHILE _IrrRunning()
 _IrrBeginScene(0, 0, 0)
 _IrrDrawScene()
 _IrrEndScene()
WEND

_IrrStop()
_IrrStop

Stops the Irrlicht Engine freeing all of the resources, closing the display window and IrrlichtWrapper.dll.

#include <au3Irrlicht2.au3>
_IrrStop()

Parameters

None.

Return Value

Success: True
Failure: False

Remarks

None.

Related

_IrrStart, _IrrStartAdvanced, _IrrRunning

Example

#include "au3Irrlicht2.au3"
_IrrStart($IRR_EDT_OPENGL, 800, 600, $IRR_BITS_PER_PIXEL_32, _
$IRR_WINDOWED, $IRR_SHADOWS, $IRR_CAPTURE_EVENTS,
$IRR_VERTICAL_SYNC_ON )

local $Camera = _IrrAddCamera(10,10,10, 0,0,0 )
local $testNode = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $testNode,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $testNode, $IRR_EMF_LIGHTING, $IRR_OFF )

WHILE _IrrRunning()
    _IrrBeginScene(0, 0, 0)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
au3Irr2 Function Reference

_IrrTransparentZWrite

[todo]

#include <au3Irrlicht2.au3>
_IrrTransparentZWrite()

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
# _IrrGetFPS

Include `<au3Irrlicht2.au3>`

```au3
_IrrGetFPS()
```

## Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

## Return Value

- **success**: explanation
- **failure**: explanation
  
  ```au3
  moreExplanationIndented
  ```

## Remarks

(todo)

## Related

(todo: functionName, functionName)
au3Irr2 Function Reference

_IrrGetPrimitivesDrawn

[todo]

#include <au3Irrlicht2.au3>
_IrrGetPrimitivesDrawn()

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetWindowCaption

Set the caption in the Irrlicht window title bar.

```
#include <au3Irrlicht2.au3>
_IrrSetWindowCaption($s_Caption)
```

Parameters

$s_Caption | String for the title bar.

Return Value

Success: True
Failure: False

Remarks

None.

Related

Example

```
#include "au3Irrlicht2.au3"

_IrrStart()

_IrrSetWindowCaption("Title of the window")
```
local $nodeCamera = _IrrAddCamera(10,10,10,0,0,0)
local $testNode = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $testNode,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $testNode, $IRR_EMF_LIGHTING, $IRR_OFF )

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 0)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
au3Irr2 Function Reference

_IrrGetScreenSize

[todo]

#include <au3Irrlicht2.au3>
_IrrGetScreenSize(ByRef $i_Width, ByRef $i_Height)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>moreTextForParam1</td>
</tr>
</tbody>
</table>

Return Value

[success]: explanation
[failure]: explanation
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrMaximizeWindow

(todo)

#include <au3Irrlicht2.au3>
_IrMaximizeWindow()

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
<th>moreTextForParam1</th>
</tr>
</thead>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

(todo)

Related

(todo: functionName, functionName)
Function Reference

_IrrMinimizeWindow

[TODO]

#include <au3Irrlicht2.au3>
_IrrMinimizeWindow()

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[TODO]

Related

[TODO: functionName, functionName]
au3Irr2 Function Reference

_IrrRestoreWindow

[todo]

#include <au3Irrlicht2.au3>
_IrrRestoreWindow()

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrSetResizableWindow

(todo)

#include <au3Irrlicht2.au3>
_IrrSetResizableWindow($i_Resizable)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
</table>

Return Value

[success]: explanation
[failure]: explanation
    [moreExplanationIndented]

Remarks

(todo)

Related

(todo: functionName, functionName)
_IrrMakeARGB

Computes valid 32bit color value including alpha (translucency) as expected from several functions.

```
#include <au3Irrlicht2.au3>
_IrrMakeARGB($i_Alpha, $i_Red, $i_Green, $i_Blue)
```

**Parameters**

<table>
<thead>
<tr>
<th>$i_Alpha</th>
<th>Alpha component of the colour.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_Red, $i_Green, $i_Blue</td>
<td>Red, green and blue components (0-255).</td>
</tr>
</tbody>
</table>

**Return Value**

success: 32bit unsigned int colour value including alpha.

**Remarks**

None.

**Related**

None.

**Example**

```
#include "au3Irrlicht2.au3"
```
_IrrStart()

local $font = _IrrGetFont( "./media/fonthaettenschweiler.bmp" )

_IrrAddBillboardTextSceneNode( $font, "au3Irrlicht2 ...", 30.0, 15.0, 0.0, 0.0, 20.0, 0, _
    _IrrMakeARGB(0, 255, 0, 0), _IrrMakeARGB(0, 255, 255, 0) )
_IrrAddBillboardTextSceneNode( $font, "... what else?", 30.0, 15.0, 0.0, -5.0, 17.0, 0, _
    _IrrMakeARGB(0, 255, 255, 0), _IrrMakeARGB(0, 0, 255, 0) )

_IrrAddFPSCamera()

WHILE _IrrRunning()
    _IrrBeginScene(0,0,25)
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
au3Irr2 Function Reference

_IrrQueryFeature

[todo]

#include <au3Irrlicht2.au3>
_IrrQueryFeature($i_Feature)

Parameters

[|param1| explanation| [moreTextForParam1|

Return Value

[success]: [explanation]
[failure]: [explanation]
  [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrGetTime

[todo]

#include <au3Irrlicht2.au3>
_ IrrGetTime()

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
Function Reference

__CreateVertStruct

Helper function: creates a structure array of vertices as expected from some functions.

```c
#include <au3Irrlicht2.au3>
__CreateVertStruct($iVert)
```

**Parameters**

| $iVert | Number of vertex elements the struct array shall contain. |

**Return Value**

Success: The created vertex array struct

**Remarks**

None

**Related**

__GetVertStruct, __SetVertStruct, __CreateVectStruct

**Example**

```c
#include "au3Irrlicht2.au3"

_IrrStart()
local $tVertex = __CreateVertStruct(5)
```
; set up five vertices to define the points of a pyramid. the vertices have many properties that need to be set up to properly define the structure
__SetVertStruct__(tVertex, 0, $VERT_X, -20)
__SetVertStruct__(tVertex, 0, $VERT_Y, 0)
__SetVertStruct__(tVertex, 0, $VERT_Z, -20)
__SetVertStruct__(tVertex, 1, $VERT_X, -20)
__SetVertStruct__(tVertex, 1, $VERT_Y, 0)
__SetVertStruct__(tVertex, 1, $VERT_Z, 20)
__SetVertStruct__(tVertex, 2, $VERT_X, 20)
__SetVertStruct__(tVertex, 2, $VERT_Y, 0)
__SetVertStruct__(tVertex, 2, $VERT_Z, 20)
__SetVertStruct__(tVertex, 3, $VERT_X, 20)
__SetVertStruct__(tVertex, 3, $VERT_Y, 0)
__SetVertStruct__(tVertex, 3, $VERT_Z, -20)
__SetVertStruct__(tVertex, 4, $VERT_X, 0)
__SetVertStruct__(tVertex, 4, $VERT_Y, 35)
__SetVertStruct__(tVertex, 4, $VERT_Z, 0)

; co-ordinates across a texture run from 0 to 1 so we place each of the vertices on this texture plane to appear; if the pyramid was painted from its bottom up
__SetVertStruct__(tVertex, 0, $VERT_TEXTUREX, 0)
__SetVertStruct__(tVertex, 0, $VERT_TEXTUREY, 0)
__SetVertStruct__(tVertex, 1, $VERT_TEXTUREX, 0)
__SetVertStruct__(tVertex, 1, $VERT_TEXTUREY, 1)
__SetVertStruct__(tVertex, 2, $VERT_TEXTUREX, 1)
__SetVertStruct__(tVertex, 2, $VERT_TEXTUREY, 1)
__SetVertStruct__(tVertex, 3, $VERT_TEXTUREX, 1)
__SetVertStruct__(tVertex, 3, $VERT_TEXTUREY, 0)
__SetVertStruct__(tVertex, 4, $VERT_TEXTUREX, 0.5)
__SetVertStruct__(tVertex, 4, $VERT_TEXTUREY, 0.5)

; each of the vertices can be assigned a colour to tint the texture:
__SetVertStruct__(tVertex, 0, $VERT_VCOLOR, _IrrMakeARGB(0, 255, 255, 255))
__SetVertStruct__(tVertex, 1, $VERT_VCOLOR, _IrrMakeARGB(0, 255, 255,
__SetVertStruct($tVertex, 2, $VERT_VCOLOR, _IrrMakeARGB(0, 255, 255, 255))
__SetVertStruct($tVertex, 3, $VERT_VCOLOR, _IrrMakeARGB(0, 255, 255, 255))
__SetVertStruct($tVertex, 4, $VERT_VCOLOR, _IrrMakeARGB(0, 255, 255, 255))
;
; create the faces, this is an array of indices referencing the vectors they
; are collected into groups of three each defining a triangle in the mesh
local $aIndices[18] = [0,1,4, 1,2,4, 2,3,4, 3,0,4, 2,1,0, 0,3,2]

; create the mesh from the array of vertices and indices
local $hMesh = _IrrCreateMesh("testMesh", $tVertex, $aIndices)

; add the mesh to the scene
local $SceneNode = _IrrAddMeshToScene( $hMesh )
_IrrAddRotationAnimator($SceneNode, 0, 0.2, 0)
_IrrSetNodeMaterialTexture( $SceneNode, _IrrGetTexture( "/media/au3irr2_logo.jpg" ), 0 )
_IrrSetNodeMaterialFlag( $SceneNode, $IRR_EMF_LIGHTING, $IRR_OFF )

_IrrAddCamera(30, 30, 30, 0, 10, 0)

WHILE _IrrRunning()
  _IrrBeginScene(0, 0, 25)
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
__GetVertStruct

Helper function: Returns a specific value from a structure array of vertices.

```c
#include <au3Irrlicht2.au3>
__GetVertStruct(ByRef $tVertex, $iVertex, $vMember)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$tVertex</td>
<td>Structure array of vertices as created with __CreateVertStruct</td>
</tr>
<tr>
<td>$iVertex</td>
<td>Vertex element from which value shall be returned (0-based!)</td>
</tr>
<tr>
<td>$vMember</td>
<td>One of following values to return: $VERT_X, $VERT_Y, $VERT_Z, $VERT_NORMALX, $VERT_NORMALY, $VERT_NORMALZ, $VERT_VCOLOR, $VERT_TEXTUREX, $VERT_TEXTUREY</td>
</tr>
</tbody>
</table>

### Return Value

Success: Requested $vMember

### Remarks

None
Related

__CreateVertStruct, __SetVertStruct, __CreateVectStruct

Example

```au3
#include "au3Irrlicht2.au3"

​_IrrStart()

local $mesh = _IrrGetMesh( "./media/cube.x" )

local $tVertex; variable for the vertex array struct
; copy the vertex information into the array
local $vertex_count = _IrrGetMeshVertices( $mesh, 0, $tVertex)

local $i
for $i = 0 to $vertex_count - 1; iterate through all of the vertices
    ; shrink vertex X location by half its size, then change vertex colour value
    __SetVertStruct($tVertex, $i, $VERT_X, __GetVertStruct($tVertex, $i, $VERT_X) * 0.5 )
    __SetVertStruct($tVertex, $i, $VERT_VCOLOR, _IrrMakeARGB(0, Random(0,255), Random(0,255), Random(0,255) ) )
next ; $i

; copy the altered vertex information back to the mesh
_IrrSetMeshVertices( $mesh, 0, $tVertex )

; add mesh and camera to the scene:
local $nodeCube = _IrrAddMeshToScene( $mesh )
_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodePosition($nodeCube, -0.5, -0.5, 5)

_IrrAddFPSCamera($IRR_NO_OBJECT, 5, 0.01 )
```
WHILE _IrrRunning()
  _IrrBeginScene( 0, 0, 25 )
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
Helper function: Sets a value into a structure array of vertices.

```csharp
#include <au3Irrlicht2.au3>
__SetVertStruct(ByRef $tVertex, $iVertex, $vMember, $vData)
```

**Parameters**

<table>
<thead>
<tr>
<th>$tVertex</th>
<th>Structure array of vertices as created with __CreateVertStruct</th>
</tr>
</thead>
<tbody>
<tr>
<td>$iVertex</td>
<td>Vertex element where specific value shall be set (0-based!)</td>
</tr>
<tr>
<td>$vMember</td>
<td>One of following values to set:</td>
</tr>
<tr>
<td></td>
<td>$VERT_X</td>
</tr>
<tr>
<td></td>
<td>$VERT_Y</td>
</tr>
<tr>
<td></td>
<td>$VERT_Z</td>
</tr>
<tr>
<td></td>
<td>$VERT_NORMALX</td>
</tr>
<tr>
<td></td>
<td>$VERT_NORMALY</td>
</tr>
<tr>
<td></td>
<td>$VERT_NORMALZ</td>
</tr>
<tr>
<td></td>
<td>$VERT_VCOLOR</td>
</tr>
<tr>
<td></td>
<td>$VERT_TEXTUREX</td>
</tr>
<tr>
<td></td>
<td>$VERT_TEXTUREY</td>
</tr>
</tbody>
</table>

**Return Value**

None.

**Remarks**

None
Related

__CreateVertStruct, __GetVertStruct, __CreateVectStruct

Example

```au3
#include "au3Irrlicht2.au3"

_IrrStart()

local $mesh = _IrrGetMesh( "./media/cube.x" )

local $tVertex; variable for the vertex array struct
; copy the vertex information into the array
local $vertex_count = _IrrGetMeshVertices( $mesh, 0, $tVertex )

local $i
for $i = 0 to $vertex_count - 1 ; iterate through all of the vertices
    ; shrink vertex X location by half its size, then change vertex colour value
    __SetVertStruct($tVertex, $i, $VERT_X, __GetVertStruct($tVertex, $i, $VERT_X) * 0.5 )
    __SetVertStruct($tVertex, $i, $VERT_VCOLOR, _IrrMakeARGB(0, Random(0,255), Random(0,255), Random(0,255) ) )
next ; $i

; copy the altered vertex information back to the mesh
_IrrSetMeshVertices( $mesh, 0, $tVertex )

; add mesh and camera to the scene:
llocal $nodeCube = _IrrAddMeshToScene( $mesh )
_IrrSetNodeMaterialFlag( $nodeCube, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodePosition($nodeCube, -0.5, -0.5, 5)

_IrrAddFPSCamera($IRR_NO_OBJECT, 5, 0.01 )

WHILE _IrrRunning()
```

_IrrBeginScene( 0, 0, 25 )
_IrrDrawScene()
_IrrEndScene()
WEND

_IrrStop()

Copy to Clipboard
__CreateVectStruct

Helper function: creates a structure array of vectors as expected from some functions.

```
#include <au3Irrlicht2.au3>
__CreateVectStruct($iVect)
```

**Parameters**

| $iVect | Number of vector elements the created struct array shall contain. |

**Return Value**

Success: The created vector struct.

**Remarks**

None

**Related**

__SetVectStruct, __GetVectStruct, __CreateVertStruct

**Example**

```
#include "au3Irrlicht2.au3"

_IrrStart()
```
local $nodeCamera = _IrrAddCamera( 150,50,0, 0,50,0 )
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest, _IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

local $tVectors = __CreateVectStruct(4)
__SetVectStruct($tVectors, 0, -100, 50, 0)
__SetVectStruct($tVectors, 1, 0, 100, -100)
__SetVectStruct($tVectors, 2, 100, 50, 0)
__SetVectStruct($tVectors, 3, 0, 100, 100)

_IrrAddSplineAnimator( $nodeTest, $tVectors, 0, 0.5, 1)

WHILE _IrrRunning() BEGIN
  _IrrBeginScene( 0, 0, 25 )
  _IrrDrawScene()
  _IrrEndScene()
END

_IrrStop()

Copy to Clipboard
**au3Irr2 Function Reference**

**__GetVectStruct**

Helper function: Returns a specific value from a structure array of vectors.

```
#include <au3Irrlicht2.au3>
__GetVectStruct(ByRef $tVector, $iVector, $vMember)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$tVector</td>
<td>Structure array of vectors as created with __CreateVectStruct</td>
</tr>
<tr>
<td>$iVector</td>
<td>Vector element from which value shall be returned (0-based!)</td>
</tr>
<tr>
<td>$vMember</td>
<td>One of following values to return: $VECT_X, $VECT_Y, $VECT_Z</td>
</tr>
</tbody>
</table>

### Return Value

Success: Requested $vMember

### Remarks

None

### Related

__CreateVectStruct, __SetVectStruct, __CreateVertStruct
Example

```plaintext
#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeCamera = _IrrAddCamera( 150,50,0, 0,50,0 )
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
_IrrGetTexture("/media/au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

local $tVectors = __CreateVectStruct(4)
__SetVectStruct($tVectors, 0, -100, 50, 0)
__SetVectStruct($tVectors, 1, 0, 100, -100)
__SetVectStruct($tVectors, 2, 100, 50, 0)
__SetVectStruct($tVectors, 3, 0, 100, 100)

_IrrAddSplineAnimator( $nodeTest, $tVectors, 0, 0.5, 1)

_MsgBox(0, "", "Y value of very last vector: " & __GetVectStruct($tVectors, 3, $VECT_Y) )

WHILE _IrrRunning()
    _IrrBeginScene( 0, 0, 25 )
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
```

Copy to Clipboard
__SetVectStruct

Helper function: Sets x, y, z values into a structure array of vectors.

```
#include <au3Irrlicht2.au3>
__SetVectStruct(ByRef $tVector, $iVector, $fX, $fY, $fZ)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$tVector</td>
<td>Structure array of vectors as created with __CreateVectStruct</td>
</tr>
<tr>
<td>$iVector</td>
<td>Vector element to set (0-based!)</td>
</tr>
<tr>
<td>$fX, $fY, $fZ</td>
<td>X, Y, Z values of the vector</td>
</tr>
</tbody>
</table>

### Return Value

None.

### Remarks

None

### Related

__CreateVectStruct, __GetVectStruct, __CreateVertStruct

### Example

```
#include "au3Irrlicht2.au3"
```
local $nodeCamera = _IrrAddCamera( 150,50,0, 0,50,0 )
local $nodeTest = _IrrAddTestSceneNode()
_IrrSetNodeMaterialTexture( $nodeTest,
_IrrGetTexture("\media\au3irr2_logo.jpg"), 0)
_IrrSetNodeMaterialFlag( $nodeTest, $IRR_EMF_LIGHTING, $IRR_OFF )

local $tVectors = __CreateVectStruct(4)
__SetVectStruct($tVectors, 0, -100, 50, 0)
__SetVectStruct($tVectors, 1, 0, 100, -100)
__SetVectStruct($tVectors, 2, 100, 50, 0)
__SetVectStruct($tVectors, 3, 0, 100, 100)

_IrrAddSplineAnimator( $nodeTest, $tVectors, 0, 0.5, 1)

WHILE _IrrRunning()
 _IrrBeginScene( 0, 0, 25 )
 _IrrDrawScene()
 _IrrEndScene()
WEND

_IrrStop()
_IrrAddTerrain

Creates a terrain object from a gray scale bitmap.

Include <au3Irrlicht2.au3>

_IrrAddTerrain($s_Path, $f_PosX = 0.0, $f_PosY = 0.0, $f_PosZ = 0.0, $f_RotX = 0.0, $f_RotY = 0.0, $f_RotZ = 0.0, $f_ScaleX = 1.0, $f_ScaleY = 1.0, $f_ScaleZ = 1.0, $i_VertexAlpha = 255, $i_VertexRed = 255, $i_VertexGreen = 255, $i_VertexBlue = 255, $i_Smoothing = 0, $i_MaxLOD = 5, $i_PatchSize = $ETPS_17)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_Path</td>
<td>Filename of a gray scale image used to define the contours of the surface</td>
</tr>
<tr>
<td>$f_PosX, $f_PosY, $f_PosZ</td>
<td>[optional] Define position of the terrain</td>
</tr>
<tr>
<td>$f_RotX, $f_RotY, $f_RotZ</td>
<td>[optional] Define rotation of the terrain</td>
</tr>
<tr>
<td>$f_ScaleX, $f_ScaleY, $f_ScaleZ</td>
<td>[optional] Define scaling of the terrain</td>
</tr>
<tr>
<td>$i_VertexAlpha</td>
<td>[optional] Alpha value for the vertex colour from 0 to 255.</td>
</tr>
<tr>
<td>$i_VertexRed, $i_VertexGreen, $i_VertexBlue</td>
<td>[optional] Define the vertex colour of all points in the terrain (values from 0 to 255)</td>
</tr>
<tr>
<td>$i_Smoothing</td>
<td>[optional] True or false defines whether the contours of the surface are smoothed over.</td>
</tr>
<tr>
<td></td>
<td>[optional] Control the properties of the level of detail</td>
</tr>
<tr>
<td>$i_{\text{MaxLOD}},$</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>$i_{\text{PatchSize}}$</td>
<td></td>
</tr>
<tr>
<td>calculations applied to the terrain. It is recommended that these are left at default values.</td>
<td></td>
</tr>
</tbody>
</table>

Valid values for $i_{\text{PatchSize}}$:
- $ETPS_9$ (patch size of 9, at most, use 4 levels of detail with this patch size)
- $ETPS_17$ (patch size of 17, at most, use 5 levels of detail with this patch size)
- $ETPS_33$ (patch size of 33, at most, use 6 levels of detail with this patch size)
- $ETPS_65$ (patch size of 65, at most, use 7 levels of detail with this patch size)
- $ETPS_{129}$ (patch size of 129, at most, use 8 levels of detail with this patch size)

**Return Value**

success: Handle of the terrain object
failure: false

**Remarks**

The terrain is created from a gray scale bitmap where bright pixels are high points on the terrain and black pixels are low points. You will inevitably have to rescale the terrain during the call or after it is created.

The Terrain object is a special dynamic mesh whose resolution is reduced in the distance to reduce the number of triangles it consumes.

**Related**

[IrrScaleTexture](#)

**Example**
`#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeTerrain = _IrrAddTerrain( "./media/terrain-heightmap.bmp" )
_IrrSetNodeScale( $nodeTerrain, 40.0, 4.4, 40.0 )

_IrrSetNodeMaterialTexture( $nodeTerrain, _IrrGetTexture( "./media/terrain-texture.jpg" ), 0 )
_IrrSetNodeMaterialTexture( $nodeTerrain, _IrrGetTexture( "./media/detailmap3.jpg" ), 1 )

_IrrScaleTexture( $nodeTerrain, 1.0, 60.0 )

_IrrSetNodeMaterialFlag( $nodeTerrain, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodeMaterialType( $nodeTerrain, $IRR_EMT_DETAIL_MAP )

local $nodeCamera = _IrrAddFPSCamera()
_IrrSetNodePosition( $nodeCamera, 3942.8, 1102.7, 5113.9 )

_IrrSetCameraClipDistance( $nodeCamera, 12000 )

WHILE _IrrRunning()
    _IrrBeginScene( 240, 255, 255 )
    _IrrDrawScene()
    _IrrEndScene()
WEND

_IrrStop()
_IrrAddTerrainTile

[todo]

```
#include <au3Irrlicht2.au3>
_ImmAddTerrainTile($h_Image, $i_TileSize = 256, $i_DataX = 0, $i_DataY = 0, $f_PosX = 0.0, $f_PosY = 0.0, $f_PosZ = 0.0, $f_RotX = 0.0, $f_RotY = 0.0, $f_RotZ = 0.0, $f_ScaleX = 1.0, $f_ScaleY = 1.0, $f_ScaleZ = 1.0, $i_Smoothing = 1, $i_MaxLOD = 5, $i_PatchSize = $ETPS_17)
```

Parameters

| [param1] | [explanation] | [moreTextForParam1] |

Return Value

[success]: [explanation]
[failure]: [explanation]

Remarks

[todo]

Related

[todo: functionName, functionName]
### _IrrAddSphericalTerrain

[todo]

```au3
#Include <au3Irrlicht2.au3>
_IrrAddSphericalTerrain($s_TopPath, $s_FrontPath, $s_BackPath, $s_LeftPath, $RightPath, $s_BottomPath, $f_PosX = 0.0, $f_PosY = 0.0, $f_PosZ = 0.0, $f_RotX = 0.0, $f_RotY = 0.0, $f_RotZ = 0.0, $f_ScaleX = 1.0, $f_ScaleY = 1.0, $f_ScaleZ = 1.0, $i_VertexAlpha = 255, $i_VertexRed = 255, $i_VertexGreen = 255, $i_VertexBlue = 255, $i_Smoothing = 0, $i_Spherical = 0, $i_MaxLOD = 5, $i_PatchSize = $ETPS_17)
```

### Parameters

<table>
<thead>
<tr>
<th>Param</th>
<th>Explanation</th>
<th>More Text For Param</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Return Value

- **[success]**: [explanation]
- **[failure]**: [explanation]
  - [moreExplanationIndented]

### Remarks

[todo]

### Related

- [todo: functionName, functionName]
au3Irr2 Function Reference

_IrrGetTerrainHeight

[todo]

#include <au3Irrlicht2.au3>
_IrrGetTerrainHeight($h_Terrain, $f_X, $f_Y)

Parameters

<table>
<thead>
<tr>
<th>param1</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
    [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrScaleTexture

Specifies the scaling of a terrain object detail texture

```
#include <au3Irrlicht2.au3>
_IrrScaleTexture($h_Terrain, $f_X, $f_Y)
```

**Parameters**

<table>
<thead>
<tr>
<th>$h_Terrain</th>
<th>Handle of a terrain object</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_X, $f_Y</td>
<td>Scaling values for detail texture along x and y axis</td>
</tr>
</tbody>
</table>

**Return Value**

success: True
Failure: False

**Remarks**

As a terrain object is a particularly huge mesh when textures are applied to it they look extremely pixelated.
To get over this effect a terrain object can have two materials applied to it, one to give general surface color and a second that is copied across the surface like tiles to give a rough detailed texture.
This call specifies the scaling of this detail texture.

**Related**

_ IrrAddTerrain

**Example**
#include "au3Irrlicht2.au3"

_IrrStart()

local $nodeTerrain = _IrrAddTerrain( "./media/terrain-heightmap.bmp" )
_IrrSetNodeScale( $nodeTerrain, 40.0, 4.4, 40.0 )

_IrrSetNodeMaterialTexture( $nodeTerrain, _IrrGetTexture( "./media/terrain-texture.jpg" ), 0 )
_IrrSetNodeMaterialTexture( $nodeTerrain, _IrrGetTexture( "./media/detailmap3.jpg" ), 1 )

_IrrScaleTexture( $nodeTerrain, 1.0, 60.0 )

_IrrSetNodeMaterialFlag( $nodeTerrain, $IRR_EMF_LIGHTING, $IRR_OFF )
_IrrSetNodeMaterialType( $nodeTerrain, $IRR_EMT_DETAIL_MAP )

local $nodeCamera = _IrrAddFPSCamera()
_IrrSetNodePosition( $nodeCamera, 3942.8, 1102.7, 5113.9 )

_IrrSetCameraClipDistance( $nodeCamera, 12000 )

WHILE _IrrRunning()
  _IrrBeginScene( 240, 255, 255 )
  _IrrDrawScene()
  _IrrEndScene()
WEND

_IrrStop()
au3Irr2 Function Reference

_IrrGetTerrainTileHeight

[todo]

#Include <au3Irrlicht2.au3>
_IrrGetTerrainTileHeight($h_Terrain, $f_X, $f_Y)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
 [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
_IrrScaleTileTexture

(todo)

#include <au3Irrlicht2.au3>
_IrrScaleTileTexture($h_Terrain, $f_X, $f_Y)

Parameters

<table>
<thead>
<tr>
<th>[param1]</th>
<th>[explanation]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[moreTextForParam1]</td>
</tr>
</tbody>
</table>

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

(todo)

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrSetTileColor

[todo]

#include <au3Irrlicht2.au3>
_IrrSetTileColor($h_Terrain, $h_Image, $i_X=0, $i_Y=0)

Parameters

[|param1| [explanation]
   | [moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
   [moreExplanationIndented]

Remarks

[todo]

Related

[todo: functionName, functionName]
au3Irr2 Function Reference

_IrrScaleSphericalTexture

[todo]

#include <au3Irrlicht2.au3>
_IrrScaleSphericalTexture($h_Terrain, $f_X, $f_Y)

Parameters

[param1] [explanation]
[moreTextForParam1]

Return Value

[success]: [explanation]
[failure]: [explanation]
[moreExplanationIndented]

Remarks

[todo]

Related

todo: functionName, functionName
Credits

Autolt Forum names are mentioned in <brackets> where available.
au3Ir2

au3Ir2 Authors

- J. Rowe <JRowe>
- Andreas Templin <linus>

au3Ir2 Help Files

- One of the above ;)
- Frank Dodd (his FreeBasic Wrapper reference is greatly reusable!)

Additional Contributions

Code and documentation submitted via the forum or by email.

- <ProgAndy>
- <smashly> (also disturbing a sunday morning breakfast coffee is a sin: thanks for excellent work ;)
  )
Last but not least Special Thanks to

- Frank Dodd, his team from the FreeBasic community, and a LOT of others that have supported his Irrlicht Wrapper project with contributions or with technical assistance (see here for detailed acknowledgments).
- Nikolaus Gebhardt and the Irrlicht team.
- All the people that did the used models, textures, ...
- The tool authors around the au3 help - saved a lot of time
- All other not-yet-mentioned people sharing time, tips and work for nuts
- Everyone who uses and support au3Irr2
Wrapper Library for Irrlicht

Introduction
Overview

The first thing I must do is acknowledge the fantastic work that has been carried out by the many different authors whose amazing work makes it possible for this package to enable BASIC programmers to build rich and immersive 3D applications and games. You can find a list of acknowledgements at the end of this document.

This wrapper library provides a simple and partial gateway from non object-oriented languages like FreeBasic into the object oriented 3D graphics Engine Irrlicht. Although the wrapper only exposes part of the functionality of Irrlicht it endeavours to expose the most important parts to give you access to Irrlichts powerful features. It does not directly expose the Irrlicht commands but provides a separate syntax to encapsulate the engine.

In total it provides well over 200 commands that cover Bitmaps, 3D models, Maps, Cameras, Lights and more. At this time there is no practical support for the Irrlicht applications would need to provide any widget controls with.

This document serves as an introduction into the wrapper, from here I suggest you installation instructions below and then try out the examples, these practical demos provide an excellent way to learn how the wrapper functions. If you need specific your first port of call will be the reference manual and your last port of call the IrrlichtWrapper.bi which hopefully you will not need.
Installation

The installation of the IrrlichtWrapper for FreeBasic package is very simple, just extract the zip file to a folder of your choice and your ready to go. You may prefer to move the `IrrlichtWrapper.bi` file to your `FreeBasic\inc` directory however I leave it where my project files within the same directory. As the Irrlicht DLL included with this package is modified to include additional features you cannot use a standard Irrlicht.dll from an external download with this package, however if you like to compile your own executable C++ source is provided to enable you to do this.

Note: You will need FreeBasic version 0.20 or better to use the wrapper
The Scene

When you start Irrlicht with the IrrStart command you create a scene that represents your world, this scene is hidden behind the wrapper and is used to load and store all of your resources: Bitmap Textures, 3D Meshes, Fonts and the Nodes in your world.
The Texture and Image Objects

The texture and the image objects are both 2D bitmaps that are loaded from a bitmap file.

Textures are used to paint the surface of nodes (3D objects) in the scene or can be drawn directly to the screen as an image for counters or logos. Textures are stored in the computers main memory but, where possible, they are also stored in the Graphics Cards video memory to speed up 3D drawing. When you get a new texture the wrapper will return the object to you as an `irr_texture` type.

Images are usually used as a source of information and are not drawn to the screen and are instead used to set the height of terrains or to describe how grass is distributed. Images are stored only in the computer's main memory. When you get a new image file the wrapper will return the object to you as an `irr_image`.

You need to save and manage both of these objects so you can use them later. Once finished with them it is important that you remove them from memory as images can occupy a lot of memory and you can easily run out of video memory in particular.
The Font Object

An Irrlicht font is a monochromatic bitmap font that can be used to draw simple text onto the display. When you get a new font the wrapper will return the object to you as an *irr_font* type.
The Mesh Object

The Mesh can be thought of like a blue print and is not actually displayed in your scene. It is a list of triangular geometry that fits together to describe a 3D model, this can be a static prop, a complex map or a character. The mesh can also store animation information and a description of which parts of a texture are to be painted onto its surface.

Where a mesh describes a map it will also have references to dozens of bitmaps that it uses to paint its surfaces.. When you get a new mesh object the wrapper will return it to you as an irr_mesh type.
The Node Object

A node is a physical object in your world a 3D Model, Camera, Light, Terrain, Billboard or Particle System. It is an element that has a position, rotation and scale, it will usually be rendered as a visible element if it is in front of the active camera when the canvas is drawn.

Nodes can usually be painted with a texture a process that is referred to as applying a material, they can be moved, rotated and scaled, hidden from view and deleted from the scene.

A 3D model is usually created by adding a mesh object to the scene. When you create a new node it will be returned to you as an irr_node type.
The Camera Object

A camera object is a special node type and represent your vantage point into the 3D world. The camera has a viewpoint and a target at which it looks. Cameras have a series of special commands that can alter the appearance of the display however they can also be copied into variables of type `irr_node` and be manipulated with all of the node commands. When a camera is created it is returned to you an `irr_camera` type.
The Terrain Object

A terrain object is a special node type and consists of a large square mesh that is pulled and lowered to represent hills, valleys and mountains. The terrain can be textured to give the appearance of a realistic landmass and with careful design it can even have map objects buried into it. Terrains have some special commands however they can also be copied directly into variables of type `irr_node` and be manipulated with all of the node commands. When a terrain is created it is returned to you an `irr_texture` type.
The Particle System Object

A particle system is a special node type that can be set up to spit out hundreds of tiny little Billboard like objects and simulate effects like, fire, explosions, fountains, waterfalls, rain etc ... The particle system has an item called an emitters attached to spit out particles and items called affectors attached to fade then out and change their direction. Particle systems have some special commands however they can also be copied directly into variables of type irr_node and be manipulated with all of the node commands. When a particle system is created it is returned to you an irr_particle_system type.
The Animator Object

Animator objects are attached to nodes and are used to delete, rotate or move the way without you having to control the animation yourself step by step. When you animator it will be returned to you as an irr_animator type.
The Selector Object

Selector objects are used to gather together groups of triangular geometry in the scene that can be used for special functions, at the moment the wrapper only supports using them for collision detection. When you create a selector object it will be returned to you as a `irr_selector` object.
License

The Irrlicht Engine is based in part on the work of the Independent JPEG Group. Please refer to the Irrlicht documentation and license agreements for further information.

This software is provided 'as-is', without any express or implied warranty. In no event will the author be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.
Contact

If you have any problems or suggestions I would be happy to receive them. Although I cannot promise any action or an individual response I do try to help where possible and look after the babies even when I release them into the wild. Mail: frankdodd@ukonline.co.uk
Acknowledgements

I wish to express my gratitude to the following people whose work the wrapper is built upon or who have helped with contributions to the wrapper or with technical assistance for features and problems.

**Irrlicht**
http://irrlicht.sourceforge.net/author.html

**FreeBasic**
http://www.freebasic.net/index.php/about?section=credits

**Simple Terrain**
With invaluable help from the OpenGL Splat tutorial by Jarno van der Linden (http://www.cs.auckland.ac.nz/~jvan006/multitex/)

**Spattering**
Released under the Irrlicht License by G Davidson

**Grass SceneNode**
Released under the Irrlicht License by G Davidson

**Clouds SceneNode**
Released under the Irrlicht License by G Davidson

**Lens Flare Scene Node**
Placed into the public domain by Paulo Oliveira

**6DOF Camera**
With lots of help from Colin MacDonald and theoretical advice from Aleofjax

**Color and Spline particle affectors**
Released under the Irrlicht License by Dark Kilauea

**Newton Physics Engine support**
Developed by SiskinEDGE and the library provided by the Newton Dynamics team

**Open Dynamics Engine Physics support**
Bindings developed by D.J Peters and the library provided by the Open Dynamics Engine team.

**Many Irrlicht commands and support**
Provided by Agamemnus and The Car

**FreeType truetype font support**
Provided by the FreeType team (www.freetype.org)

**Batching Meshes**
Provided by Gaz Davidson (Bitplane)

**Beam Node**
Provided by Gaz Davidson (Bitplane)

**Bolt Node**
Provided by Sudi and Trivtn under the Irrlicht License

**XEffects - Reloaded**
Provided by Blindside

**Many thanks for the development tools**
Code::Blocks (www.codeblocks.org); GCC Compiler (gcc

**Lots of thanks for**
Eponasoft; Daiwa; Crocodudule, John K, AlecZ, Alvaro V
help in the Forums and everyone that has offered their support. Thanks.

And many thanks to the thousands of mails posts, examples, replies and scattered across the Internet that provided insights into many of the questions to be solved in creating this library.
Table of Contents

- Overview
- System
- Keyboard and Mouse
- Filing System
- 2D
- Materials
- Scene
- Node
- Animation
- Collision
- Camera
- Lighting
- Particle System
- Terrain
- Graphical User Interface
- Wrapper Structure Definitions
Overview

Before using this manual I suggest that you first read the introduction document that will introduce the basic concepts. While I have made great effort to make this document both comprehensive and accurate there are bound to be errors and omissions please be careful when relying on the information when solving problems.

Irrlicht is a highly impressive and powerful rendering engine that has a wide range of operations that can be used for many types of 3D application, it has an impressive, clean and well designed object oriented structure that is linked to through a simple DLL interface.

Unfortunately its object oriented structure makes it extremely difficult to use from imperative languages like FreeBasic that do not support an object model. This is where the wrapper come in. It provides an interface between languages like FreeBasic and the objected oriented model of Irrlicht. Many functions are translated into simple calls that return the objects back to FreeBasic as simple variables that it can then manage, there is some overhead in going through the wrapper although the interface code has been kept simple to reduce to a minimum.
System

These calls deal with starting, running and stopping the Irrlicht engine, it also includes calls that get system metrics and some other miscellaneous tools.

IrrStart

Syntax

IrrStart ( device type, screen width as integer, screen height as integer, bits per pixel, full screen, use shadows, capture mouse and keyboard, vertical synchronisation )

Description

Starts the Irrlicht interface and opens a window for rendering.

device type specifies the renderer to use when drawing to the display this may be one of the following types: -

- IRR_EDT_NULL
  A NULL device with no display
- IRR_EDT_SOFTWARE
  Irrlicht's default software renderer
- IRR_EDT_SOFTWARE2
  An improved quality software renderer
- IRR_EDT_OPENGL
  Hardware accelerated OpenGL renderer
- IRR_EDT_DIRECT3D8
  Hardware accelerated DirectX 8 renderer (not included in the Wrappers 'Irrlicht.dll' distribution)
- IRR_EDT_DIRECT3D9
  Hardware accelerated DirectX 9 renderer (not included in the Wrappers 'Irrlicht.dll' distribution)

screen width specifies the width of the viewport in pixels
screen height specifies the height of the viewport in pixels
The number of color bits that is used for each pixel 32 bit color gives 24 million different colors whereas 16 bit color gives only 32,000 colors. However the advantage of 16 bit color is that some operations use half the memory and can run at up to twice the speed. This setting can be either of: -

IRR_BITS_PER_PIXEL_16
IRR_BITS_PER_PIXEL_32

Full screen specifies whether the display is to opened in full screen mode or in a window

IRR_WINDOWED
For window mode
IRR_FULLSCREEN
For fullscreen mode. When using full screen mode you will need to adjust the window size to the same dimensions as a supported screen resolution on the target display 640x400 for example.

Use shadows starts the engine in a mode that supports the rendering of stencil shadows.

IRR_NO_SHADOWS
For a display that does not support shadows.
IRR_SHADOWS
For a display that supports shadows.

Capture mouse and keyboard specified whether you want to capture keyboard and mouse events, if you choose to ignore them they will be handled by Irrlicht for FPS camera control. This parameter should be either of: -

IRR_IGNORE_EVENTS
IRR_CAPTURE_EVENTS

Vertical syncronisation specifies whether the display of each new frame is syncronised with vertical refresh of the graphics card. This produces a smoother display and avoids 'tearing' where the viewer can see parts of two different frames at the same time. The setting can be either of: -

IRR_VERTICAL_SYNC_OFF
IRR_VERTICAL_SYNC_ON

Example

IrrStart( IRR_EDT_OPENGL, screen_width, screen_height,
**IrrStartAdvanced**

Syntax

```plaintext
integer = IrrStart ( _
    drivertype as IRR_DEVICE_TYPES, _
    scrWidth as integer, _
    scrHeight as integer, _
    bits as uinteger, _
    fullscreen as uinteger, _
    shadows as uinteger, _
    dontignoreinput as uinteger, _
    vsyncenabled as uinteger = IRR_OFF, _
    devicetype as uinteger = 0, _
    doublebufferenabled as uinteger = IRR_ON, _
    antialiasenabled as uinteger = 0, _
    highprecisionfpu as uinteger = IRR_OFF )
```

Description

An advanced call for starting the Irrlicht interface and opens a window for rendering.

device type specifies the renderer to use when drawing to the display this may be one of the following types: -

- **IRR_EDT_NULL**
  A NULL device with no display
- **IRR_EDT_SOFTWARE**
  Irrlicht's default software renderer
- **IRR_EDT_SOFTWARE2**
  An improved quality software renderer
- **IRR_EDT_OPENGL**
  Hardware accelerated OpenGL renderer
IRR_EDT_DIRECT3D8
Hardware accelerated DirectX 8 renderer (not included in the Wrappers 'Irlicht.dll' distribution)

IRR_EDT_DIRECT3D9
Hardware accelerated DirectX 9 renderer (not included in the Wrappers 'Irlicht.dll' distribution)

Screen width specifies the width of the viewport in pixels
Screen height specifies the height of the viewport in pixels

The number of color bits that is used for each pixel 32 bit color gives 24 million different colors whereas 16 bit color gives only 32,000 colors. However the advantage of 16 bit color is that some operations use half the memory and can run at up to twice the speed. This setting can be either of:
- IRR_BITS_PER_PIXEL_16
- IRR_BITS_PER_PIXEL_32

Full screen specifies whether the display is to opened in full screen mode or in a window
- IRR_WINDOWED
  For window mode
- IRR_FULLSCREEN
  For fullscreen mode. When using full screen mode you will need to adjust the window size to the same dimensions as a supported screen resolution on the target display 640x400 for example.

Use shadows starts the engine in a mode that supports the rendering of stencil shadows.
- IRR_NO_SHADOWS
  For a display that does not support shadows.
- IRR_SHADOWS
  For a display that supports shadows.

Capture mouse and keyboard specified whether you want to capture keyboard and mouse events, if you choose to ignore them they will be handled by Irrlicht for FPS camera control. This parameter should be either of:
- IRR_IGNORE_EVENTS
- IRR_CAPTURE_EVENTS
vertical syncronisation specifies whether the display of each new frame is syncronised with vertical refresh of the graphics card. This produces a smoother display and avoids 'tearing' where the viewer can see parts of two different frames at the same time. The setting can be either of:

IRR_VERTICAL_SYNC_OFF
IRR_VERTICAL_SYNC_ON

devicetype allows a specific type of device for example a windows screen or a console to be selected. For the time being this should be set to 0 which automatically selects the best device

doublebufferenabled is used to control whether double buffering is used. When double buffering is used two drawing surfaces are created one for display and the other that is used for drawing too. Double buffering is required for antialiasing the options are: IRR_ON or IRR_OFF

antialiasenabled is used to enable the antialiasing effect, this effect produces a blurring at the edges of object giving their lines a smooth natural appearence. There is usually a big penalty for using this effect though sometimes as high as 30% of the frame rate or more. This is a value for the anti-aliasing and should be a power of 2. (e.g: 2, 4, 8, 16)

highprecisionfpu is used to enable high precision Floating point calculations, that produce more accurate result at the expense of a slower operating speed.

Example

IrrStartAdvanced ( _
    IRR_EDT_OPENGL, _  ' Use OpenGL
    512, 512, _      ' in a window 640x480
    IRR_BITS_PER_PIXEL_32, _ ' using 32 bit true color
    IRR_WINDOWED, _     ' in a window
    IRR_NO_SHADOWS, _    ' without stencil shadows
    IRR_IGNORE_EVENTS, _ ' dont capture keystrokes and mouse
    IRR_ON, _          ' sync to the monitor refresh rate
    0, _               ' 0 = use the most appropriate window device
    IRR_ON, _          ' Switch on double buffering of the display
    4, _              ' Anti-aliasing level 4
    IRR_ON )           ' use high precision floating point math
### IrrRunning

**Syntax**

IrrRunning

**Description**

Used to determine if the Irrlicht engine is still running.

**Example**

```cpp
IrrStart( IRR_EDT_OPENGL, screen_width, screen_height, IRR_WINDOWED, IRR_SHADOWS, IRR_IGNORE_EVENTS )
While IrrRunning
   Wend
IrrStop
```

### IrrSetViewPort

**Syntax**

IrrSetViewPort( topX as integer, topY as integer, bottomX as integer, bottomY as integer )

**Description**

Define the area of the screen into which elements are going to be drawn. This can be used to draw the scene multiple times for split screen effects.

**Example**

```cpp
IrrSetActiveCamera( FirstCamera )
IrrSetViewPort( 0, 0, 200, 200 )
IrrDrawScene
```

### IrrSetRenderTarget

**Syntax**

IrrSetRenderTarget (texture As irr_texture, sceneBackgroundColor As Uinteger = 0, clearBackBuffer As Ubyte = 1, clearZBuffer As Ubyte = 1)

**Description**

Set the target surface for rendering, this allows objects to be rendered to a texture that can then be drawn to the screen or displayed on other objects.
Calling this function with texture set to 0 sets the drawing target back to the screen.

Texture is a texture created with the special `IrrCreateRenderTargetTexture` call. The scene background color is generated with the FreeBasic RGBA call and defines the colour used in any clear operation. Clean back buffer when set to IRR_ON erases the background of the texture. Clear z buffer when set to IRR_ON erases the depth buffer (used by stencil shadows and some shaders).

Example

```
Texture = IrrCreateRenderTargetTexture( 512, 512 )
IrrSetRenderTarget( Texture, RGBA( 0,0,0,0), IRR_ON, IRR_ON )
IrrDrawScene
```

**IrrBeginScene**

Syntax

```
IrrBeginScene( Red as integer, Green as integer, Blue as integer )
```

Description

Starts to draw a frame, erasing the canvas with the specified color. The colors are integer values in the range from 0 (black) to 255 (full intensity).

Example

```
IrrStart( IRR_EDT_OPENGL, screen_width, screen_height, IRR_WINDOWED, IRR_SHADOWS, IRR_IGNORE_EVENTS )
While IrrRunning
   IrrBeginScene( 255, 255, 255 )
   IrrDrawScene
   IrrEndScene
Wend
IrrStop
```

**IrrDrawScene**

Syntax

```
IrrDrawScene
```

Description
This renders the 3D scene to the canvas, drawing all 3D elements: nodes, particles, billboards, etc....

**Example**

```irr
IrrStart( IRR_EDT_OPENGL, screen_width, screen_height, IRR_WINDOWED, IRR_SHADOWS, IRR_IGNORE_EVENTS )
While IrrRunning
   IrrBeginScene( 255, 255, 255 )
   IrrDrawScene
   IrrEndScene
Wend
IrrStop
```

**IrrDrawSceneToTexture**

**Syntax**

```
IrrDrawSceneToTexture( render_texture as irr_texture )
```

**Description**

Draw scene manager objects to a texture surface, the texture must have been created with a call to IrrCreateRenderTargetTexture. This is useful for creating textures from 3D objects in your scene perhaps nameplates in the interface for characters for example. Note the target texture must be **smaller** than the view window as some resources are shared between the two.

**Example**

```irr
IrrSetActiveCamera ( StaticCamera )
IrrDrawSceneToTexture ( RenderTexture )
IrrBeginScene( 240, 255, 255 )
IrrSetActiveCamera ( FPSCamera )
IrrDrawScene
```

**IrrDrawGUI**

**Syntax**

```
IrrDrawGUI
```

**Description**

This renders the 2D graphical user interface that has been created to the scene.
At the moment this wrapper only supports a static text object for experimentation purposes only.

Example

IrrStart( IRR_EDT_OPENGL, screen_width, screen_height, IRR_WINDOWED, IRR_SHADOWS, IRR_IGNORE_EVENTS )
While IrrRunning
  IrrBeginScene( 255, 255, 255 )
  IrrDrawScene
  IrrDrawGUI
  IrrEndScene
Wend
IrrStop

**IrrEndScene**

**Syntax**

IrrEndScene

**Description**

This renders the 3D scene to the canvas, drawing all 3D elements: nodes, particles, billboards, etc ....

Example

IrrStart( IRR_EDT_OPENGL, screen_width, screen_height, IRR_WINDOWED, IRR_SHADOWS, IRR_IGNORE_EVENTS )
While IrrRunning
  IrrBeginScene( 255, 255, 255 )
  IrrDrawScene
  IrrDrawGUI
  IrrEndScene
Wend
IrrStop

**IrrStop**

**Syntax**

IrrStop

**Description**

Stop the Irrlicht Engine freeing all of the resources and closing the display window.
Example

IrrStart( IRR_EDT_OPENGL, screen_width, screen_height, IRR_WINDOWED, IRR_SHADOWS, IRR_IGNORE_EVENTS )
While IrrRunning
  IrrBeginScene( 255, 255, 255 )
  IrrDrawScene
  IrrEndScene
Wend
IrrStop

IrrTransparentZWrite

Syntax
IrrTransparentZWrite

Description
Allow transparency to write to the z buffer, this is necessary sometimes to correct problems with the ordering of transparent objects in the scene, it may also have an effect of performance however.

Example
IrrTransparentZWrite

IrrGetFPS

Syntax
Integer_variable = IrrGetFPS

Description
Get the current frame rate. This is determined by the number of times the IrrEndScene is called per second.

Example
frame_rate = IrrGetFPS
IrrStop()
Print "Frame Rate was "; frame_rate
Sleep

IrrGetScreenSize
### Syntax
IrrGetScreenSize( width as integer, height as integer )

### Description
Gets the screen side into the two supplied variables.

### Example
IrrGetScreenSize( ScreenWidth, ScreenHeight )

---

### `IrrGetPrimitivesDrawn`

**Syntax**
unsigned_Integer_variable = IrrGetPrimitivesDrawn

**Description**
Get the current frame rate. This is determined by the number of times the IrrEndScene is called per second.

**Example**
```
polygons = IrrGetPrimitivesDrawn
IrrStop()
Print "The system drew about ";polygons;" triangles"
Sleep
```

---

### `IrrSetWindowCaption`

**Syntax**
IrrSetWindowCaption( caption text as wide string )

**Description**
Set the caption in the Irrlicht window title bar..

**Example**
```
IrrSetWindowCaption( "Irrlicht in Free Basic" )
```

---

### `IrrMakeARGB`

**Syntax**
unsigned_integer = IrrMakeARGB ( Alpha, Red, Green, Blue )
Description
Takes four values representing a colors Alpha, Red, Green and Blue intensity and returns them as a 32bit unsigned integer. Typically used for working with colors in IRR_VECT structures.

Example
vcolor = IrrMakeARGB( 0, 255, 128, 128 )

IrrQueryFeature

Syntax
uinteger IrrQueryFeature( Feature as IRR_VIDEO_FEATURE_QUERY )

Description
Used to determine if a particular video feature is supported by the graphics card. The function will return (1) if the feature is supported and (0) if it isn't. The feature parameter should be either of the following values:
- EVDF_RENDER_TO_TARGET
  Is driver able to render to a surface?
- EVDF_HARDWARE_TL
  Is hardware transform and lighting supported?
- EVDF_MULTITEXTURE
  Are multiple textures per material possible?
- EVDF_BILINEAR_FILTER
  Is driver able to render with a bilinear filter applied?
- EVDF_MIP_MAP
  Can the driver handle mip maps?
- EVDF_MIP_MAP_AUTO_UPDATE
  Can the driver update mip maps automatically?
- EVDF_STENCIL_BUFFER
  Are stencilbuffers switched on and does the device support stencil buffers?
- EVDF_VERTEX_SHADER_1_1
  Is Vertex Shader 1.1 supported?
- EVDF_VERTEX_SHADER_2_0
  Is Vertex Shader 2.0 supported?
- EVDF_VERTEX_SHADER_3_0
  Is Vertex Shader 3.0 supported?
Is Pixel Shader 1.1 supported?

Is Pixel Shader 1.2 supported?

Is Pixel Shader 1.3 supported?

Is Pixel Shader 1.4 supported?

Is Pixel Shader 2.0 supported?

Is Pixel Shader 3.0 supported?

Are ARB vertex programs v1.0 supported?

Are ARB fragment programs v1.0 supported?

Is GLSL supported?

Is HLSL supported?

Are non-power-of-two textures supported?

Are framebuffer objects supported?

Are vertex buffer objects supported?

Is alpha to coverage supported?

Are color masks supported?

Are multiple render targets supported?

Are separate blend settings for render targets supported?

Are separate color masks for render targets supported?

Are separate blend functions for render targets supported?
Are geometry shaders supported?

Example

```c
if IrrQueryFeature( EVDF_MULTITEXTURE ) = 0 then
  ? "MultiTexture is NOT supported"
End if
```

### IrrDisableFeature

**Syntax**

```c
uinteger IrrDisableFeature( Feature as IRR_VIDEO_FEATURE_QUERY, state as uinteger )
```

**Description**

Used to disable a particular video feature on the graphics card. The feature parameter is identical to IrrQueryFeature.

State should be either IRR_ON or IRR_OFF

**Example**

```
IrrDisableFeature( EVDF_MULTITEXTURE, IRR_OFF )
```

### IrrGetTime

**Syntax**

```c
unsigned_integer = IrrGetTime
```

**Description**

Get the current time in milliseconds.

**Example**

```
time = IrrGetTime
```

### IrrSetTime

**Syntax**

```c
IrrGetTime( time as uinteger )
```

**Description**


Set the current animation time in milliseconds.

Example

IrrSetTime( 2500 )

**IrrIsFullscreen**

**Syntax**

IrrIsFullscreen() as integer

**Description**

Checks if the Irrlicht window is running in fullscreen mode. Returns 0 if the application is windowed any other value indicates full screen mode

Example

if IrrIsFullscreen = IRR_OFF Then Print "Windowed Mode"

**IrrIsWindowActive**

**Syntax**

IrrIsWindowActive() as integer

**Description**

Checks if Irrlicht window is active. Returns 0 if the application is windowed any other value indicates full screen mode

Example

if IrrIsWindowActive > 0 Then Print IrrDrawScene

**IrrIsWindowFocused**

**Syntax**

IrrIsWindowFocused() as integer

**Description**

Checks if the Irrlicht window has focus. Returns 0 if the application is windowed any other value indicates full screen mode
<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>if IrrIsWindowFocused &gt; 0 Then Print IrrDrawScene</td>
</tr>
</tbody>
</table>

**IrrIsWindowMinimized**

**Syntax**
IrrIsWindowMinimized() as integer

**Description**
Checks if the Irrlicht window is minimized. Returns 0 if the application is windowed any other value indicates full screen mode

**Example**
if IrrIsWindowMinimized = 0 Then Print IrrDrawScene

**IrrMaximizeWindow**

**Syntax**
IrrMaximizeWindow()

**Description**
Maximizes the window if possible.

**Example**
IrrMaximizeWindow

**IrrMinimizeWindow**

**Syntax**
IrrMinimizeWindow()

**Description**
Minimizes the window if possible.

**Example**
IrrMinimizeWindow
**IrrRestoreWindow**

Syntax

IrrRestoreWindow()

Description

Restore the window to normal size if possible.

Example

IrrRestoreWindow

**IrrResizableWindow**

Syntax

IrrResizableWindow()

Description

Make the irrlicht window resizable by dragging on the corner of the window.

Example

IrrResizableWindow
**Keyboard and Mouse**

These calls allow you recover keyboard events and mouse actions that the user creates.

<table>
<thead>
<tr>
<th><strong>IrrKeyEventAvailable</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>IrrSetWindowCaption</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Determine if there are any keystrokes waiting to be read..</td>
</tr>
<tr>
<td><strong>Example</strong></td>
</tr>
<tr>
<td>while IrrKeyEventAvailable</td>
</tr>
<tr>
<td>KeyEvent = IrrReadKeyEvent</td>
</tr>
<tr>
<td>Wend</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IrrReadKeyEvent</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>irr_key_event_pointer = IrrReadKeyEvent</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Read a key event from the Irrlicht window the properties of the key event are stored in the returned type.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
</tr>
<tr>
<td>while IrrKeyEventAvailable</td>
</tr>
<tr>
<td>KeyEvent = IrrReadKeyEvent</td>
</tr>
<tr>
<td>If KeyEvent-&gt;key = IRR_KEY_DOWN then</td>
</tr>
<tr>
<td>Movement = DOWN</td>
</tr>
<tr>
<td>End If</td>
</tr>
<tr>
<td>Wend</td>
</tr>
</tbody>
</table>

| **IrrMouseEventAvailable** |
IrrMouseEventAvailable

Syntax
IrrMouseEventAvailable

Description
Determine if there are any mouse actions waiting to be read.

Example
while IrrMouseEventAvailable
    MouseEvent = IrrReadMouseEvent
wend

IrrReadMouseEvent

Syntax
irr_mouse_event_pointer = IrrReadMouseEvent

Description
Read a mouse event from the Irrlicht window the properties of the mouse event are stored in the returned type.

Example
while IrrMouseEventAvailable
    ' read the mouse event out
    MouseEvent = IrrReadMouseEvent
    if MouseEvent->action = IRR_EMIE_MOUSE_MOVED then
        SPIN = MouseEvent->x
    endif
wend

IrrSetMousePosition

Syntax
IrrSetMousePosition( x as single, y as single )

Description
Set the position of the mouse pointer and return the relative change in position.

Example
IrrSetMousePosition( XPosition, YPosition )
### IrrGetAbsoluteMousePosition

**Syntax**

IrrGetAbsoluteMousePosition( x as single, y as single )

**Description**

Gets the absolute position of the mouse pointer.

**Example**

IrrGetAbsoluteMousePosition( XPosition, YPosition )

### IrrHideMouse

**Syntax**

IrrHideMouse

**Description**

Hide the mouse pointer

**Example**

IrrHideMouse

### IrrShowMouse

**Syntax**

IrrShowMouse

**Description**

Shows the mouse pointer

**Example**

IrrShowMouse

### IrrDisplayMouse

**Syntax**

IrrDisplayMouse( hide or show the mouse as integer )
Description
Hide or show the mouse pointer while it is within the Irrlicht display. There are two macro's available for the function IrrHideMouse and IrrShowMouse to simply hide or show the mouse. 1 shows the mouse pointer and 0 hides it.

Example
IrrDisplayMouse( 0 )
Filing System

These calls deal with the way irrlicht operates with the filing system and adds archives to its a virtual filling system allowing you to compress data into zip files that you can access without decompressing them.

### IrrAddZipFile

**Syntax**

IrrAddZipFile( zip file as zstring, ignore case, ignore paths )

**Description**

Adds a zip archive to the filing system allowing you to load files straight out of the zip file. Common pk3 files are simply zip files.

Ignore case should be one of the following values: -
- IRR_USE_CASE
- IRR_IGNORE_CASE

Ignore paths allows you to simply use the filename without the path, the filename should always be unique in the archive when using this option. The value should be one of the following: -
- IRR_USE_PATHS
- IRR_IGNORE_PATHS

**Example**

IrrAddZipFile( "data.pk3", IRR_IGNORE_CASE, IRR_IGNORE_PATHS )

### IrrChangeWorkingDirectory

**Syntax**

IrrChangeWorkingDirectory( New directory as zstring )
<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change the working directory of the Irrlicht Environment.</td>
</tr>
</tbody>
</table>

**Example**

```c
IrrChangeWorkingDirectory( "c:\media" )
```

<table>
<thead>
<tr>
<th><strong>IrrGetWorkingDirectory</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td><code>string = IrrGetWorkingDirectory</code></td>
</tr>
</tbody>
</table>

**Description**

Get the current working directory of the Irrlicht Environment.

**Example**

```c
CurrentDirectory = IrrGetWorkingDirectory
```
2D

Support for 2D operations including the loading of bitmaps that can be used for texturing 3D objects or for display on the screen as 2D sprites.

<table>
<thead>
<tr>
<th><strong>IrrSetTextureCreationFlag</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>IrrSetTextureCreationFlag( flag_to_set as IRR_TEXTURE_CREATION_FLAG, flag_value as uinteger )</td>
</tr>
</tbody>
</table>

**Description**
Sets texture creation flags controlling how textures are handled when they are created. The following flags can be set:

- **ETCF_ALWAYS_16_BIT**
  Forces the driver to always create 16 bit textures, independently of which format the file on disk has. When choosing this you may lose some color detail, but gain speed and save memory. 16 bit textures can be transferred twice as quickly as 32 bit textures and only use half of the memory space. When using this flag, it does not make sense to use the flags ETCF_ALWAYS_32_BIT, ETCF_OPTIMIZED_FOR_QUALITY, or ETCF_OPTIMIZED_FOR_SPEED at the same time.

- **ETCF_ALWAYS_32_BIT**
  Forces the driver to always create 32 bit textures, independently of which format the file on disk has. Please note that some drivers (like the software device) will ignore this, because they are only able to create and use 16 bit textures. When using this flag, it does not make sense to use the flags ETCF_ALWAYS_16_BIT, ETCF_OPTIMIZED_FOR_QUALITY, or ETCF_OPTIMIZED_FOR_SPEED at the same time.

- **ETCF_OPTIMIZED_FOR_QUALITY**
  Lets the driver decide in which format the textures are created and tries to make
the textures look as good as possible. Usually it simply chooses the format in which the texture was stored on disk. When using this flag, it does not make sense to use the flags ETCF_ALWAYS_16_BIT, ETCF_ALWAYS_32_BIT, or ETCF_OPTIMIZED_FOR_SPEED at the same time.

ETCF_OPTIMIZED_FOR_SPEED
Lets the driver decide in which format the textures are created and tries to create them maximizing render speed. When using this flag, it does not make sense to use the flags ETCF_ALWAYS_16_BIT, ETCF_ALWAYS_32_BIT, or ETCF_OPTIMIZED_FOR_QUALITY, at the same time.

ETCF_CREATE_MIP_MAPS
Automatically creates mip map levels for the textures.

ETCF_NO_ALPHA_CHANNEL
Discard any alpha layer and use non-alpha color format.

Example

IrrSetTextureCreationFlag( ETCF_ALWAYS_32_BIT, IRR_ON )

**IrrGetTexture**

Syntax

```
irr_texture = IrrGetTexture( Texture file name as zstring )
```

Description

Load a 2D texture from a bitmap file into video memory that can then be used to texture a model or to draw onto the screen.

Example

```
IrrlichtLogo = IrrGetTexture( "irrlicht.bmp" )
```

**IrrGetImage**

Syntax

```
irr_texture = IrrGetImage( Texture file name as zstring )
```

Description
Load a 2D texture from a bitmap file into main memory that can then be used to supply a heightmap to a terrain or other similar CPU based operations. The images can not be used to texture 3D objects.

Example

```
TerrainMap = IrrGetImage( "heightmap.bmp" )
```

**IrrCreateTexture**

**Syntax**

```
irr_texture = IrrCreateTexture( texture_name as zstring, x_size as integer, y_size as integer, format as IRR_COLOR_FORMAT )
```

**Description**

Creates a blank texture. The format of the texture can be one of the following:

- **ECF_A1R5G5B5**
  16 bit color format used by the software driver, and thus preferred by all other Irrlicht engine video drivers. There are 5 bits for every color component, and a single bit is left for alpha information.

- **ECF_R5G6B5**
  Standard 16 bit color format.

- **ECF_R8G8B8**
  24 bit color, no alpha channel, but 8 bit for red, green and blue.

- **ECF_A8R8G8B8**
  Default 32 bit color format. 8 bits are used for every component: red, green, blue and alpha.

Example

```
IrrlichtLogo = IrrCreateTexture( "mytexture", 128, 128, ECF_A8R8G8B8 )
```

**IrrCreateImage**
**IrrCreateImage**

**Syntax**

```plaintext
irr_image = IrrCreateImage( x_size as integer, y_size as integer, format as IRR_COLOR_FORMAT )
```

**Description**

Creates a blank image, does not use video memory. The format of the image can be one of the following:

- **ECF_A1R5G5B5**
  16 bit color format used by the software driver, and thus preferred by all other irrlicht engine video drivers. There are 5 bits for every color component, and a single bit is left for alpha information.

- **ECF_R5G6B5**
  Standard 16 bit color format.

- **ECF_R8G8B8**
  24 bit color, no alpha channel, but 8 bit for red, green and blue.

- **ECF_A8R8G8B8**
  Default 32 bit color format. 8 bits are used for every component: red, green, blue and alpha.

**Example**

```plaintext
BlankPicture = IrrCreateImage( 128, 128, ECF_A8R8G8B8 )
```

**IrrRemoveTexture**

**Syntax**

```plaintext
IrrRemoveTexture( texture as irr_texture )
```

**Description**

Removes the texture from memory freeing up the space it occupied. You should ensure that the texture is not in use by materials assigned to nodes.

**Example**

```plaintext
DIM MyTexture as irr_texture
```
MyTexture = IrrGetTexture( "irrlicht.bmp" )
IrrRemoveTexture( MyTexture )

**IrrRemoveImage**

Syntax
IrrRemoveImage( image as irr_image )

Description
Removes the image from memory freeing up the space it occupied. You should ensure that the image is not in use by other functions.

Example
DIM MyImage as irr_image
MyImage = IrrGetImage( "irrlicht.bmp" )
IrrRemoveImage( MyImage )

**IrrLockTexture**

Syntax
pixels_ptr = IrrLockTexture( texture as irr_texture )

Description
Locks the texture and returns a pointer to the pixels.

Example
DIM texture_pixels as uinteger ptr
texture_pixels = IrrLockTexture( MyTexture )

**IrrUnlockTexture**

Syntax
IrrUnlockTexture( texture as irr_texture )

Description
Unlock the texture, presumably after it has been modified and recreate the mipmap levels.

Example
**IrrUnlockTexture**

Syntax

```c
IrUnlockTexture( MyTexture )
```

Description

Unlocks the texture and returns a pointer to the pixels.

Example

```c
 DIM image_pixels as uinteger ptr
 image_pixels = IrUnlockTexture( MyTexture )
```

---

**IrrLockImage**

Syntax

```c
 pixels_ptr = IrrLockImage( image as irr_image )
```

Description

Locks the image and returns a pointer to the pixels.

Example

```c
 DIM image_pixels as uinteger ptr
 image_pixels = IrrLockImage( MyImage )
```

---

**IrrUnlockImage**

Syntax

```c
 IrUnlockImage( image as irr_image )
```

Description

Unlock the image, presumably after it has been modified.

Example

```c
 IrUnlockImage( MyImage )
```

---

**IrrCreateRenderTargetTexture**

Syntax

```c
 irr_texture = IrrCreateRenderTargetTexture( x_size as integer, y_size as integer )
```

Description

Create a texture that is suitable for the scene manager to use as a surface to which it can render its 3d object. Each of the dimensions must be of a power of two for example 128x128 or 256x256.

This function is very important when producing texture maps for special effects for example a rendering of a model for a 2D image displayed in the HUD, the rendering of a model for display on a 3D surface for example a video display of
virtual camera, the rendering of the texture for the reflection of a mirror, the rendering of the environment for use in a water or chrome shader. Most cards, even old cards, will support this very important function.

Example

```
RenderTexture = IrrCreateRenderTargetTexture ( 256, 256 )
```

---

**IrrMakeNormalMapTexture**

**Syntax**

```
IrrMakeNormalMapTexture( Texture object as irr_texture, Amplitude as single )
```

**Description**

Create a normal map from a gray-scale height map texture. Normal maps are used to add a high level of surface lighting detail to what are normally low resolution models. They can have a massive effect on the realism of an object, the model you create will have to be created in "tangent" space to support this.

Example

```
IrrMakeNormalMapTexture( WallBumps, 0.9 )
```

---

**IrrColorKeyTexture**

**Syntax**

```
IrrColorKeyTexture( Texture object as irr_texture, Red as integer, Green as integer, Blue as integer )
```

**Description**

Copies any parts of the texture that are the same as the specified color into the textures alpha channel. This can then be used for special effects or to make these regions transparent.

Example

```
IrrColorKeyTexture( IrrlichtLogo, 255, 255, 255 )
```

---

**IrrDraw2DImage**


### Syntax

IrrDraw2DImage( Texture to draw as irr_texture, X position as integer, Y position as integer )

### Description

Draws the texture to the display at the supplied co-ordinates.

### Example

IrrDraw2DImage( IrrlichtLogo, 4, 4 )

### IrrDraw2DImageElement

#### Syntax

IrrDraw2DImageElement( Texture to draw as irr_texture, X position as integer, Y position as integer, Source top X as integer, Source top Y as integer, Source bottom X as integer, Source bottom Y as integer, whether to use alpha as integer )

#### Description

Draws the texture to the display at the supplied co-ordinates, the image is copied from the specified rectangle in the source texture, this enables you to put many images onto a single texture. This function also supports the alpha channel when drawing the image to the display and can draw the image transparently.

The value for whether or not to use the alpha channel should be one of the following values:
- IRR_IGNORE_ALPHA
- IRR_USE_ALPHA

#### Example

IrrDraw2DImageElement( FreeBasicLogo, screen_width - 60 - 4, 4,0,0,60,31, IRR_USE_ALPHA )

### IrrDraw2DImageElementStretch

#### Syntax

IrrDraw2DImageElementStretch (texture as irr_texture, destination top X as integer, destination top Y as integer, destination bottom X as integer,
destination bottom Y as integer, source top X as integer, source top Y as integer, source bottom X as integer, source bottom Y as integer, use Alpha as integer )

Description
Draws the texture to the display into the supplied rectangle, the image is copied from the specified rectangle in the source texture, this enables you to put many images onto a single texture. If the rectangles are different sizes this function will scale the images appropriately. This function also supports the alpha channel when drawing the image to the display and can draw the image transparently.

The value for whether or not to use the alpha channel should be one of the following values:
- IRR_IGNORE_ALPHA
- IRR_USE_ALPHA

Example
IrrDraw2DImageElementStretch( FreeBasicLogo, 16, 16, 80, 80, 0, 0, 32, 32, IRR_USE_ALPHA )

IrrGetFont

Syntax
irr_font = IrrGetFont( Filename of the bitmap font file as zstring )

Description
Loads a bitmap containing a bitmap font.

Example
BitmapFont = IrrGetFont ( "bitmapfont.bmp" )

Irr2DFontDraw

Syntax
Irr2DFontDraw ( Font Object as irr_texture, The text to display as wstring ptr, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer)

Description
Draws the text into the supplied rectangular area using the supplied font object.

Example

Irr2DFontDraw ( BitmapFont, "SIMPLE MONOCHROME FONT", 120, 80, 250, 96 )

---

**IrrSaveScreenShot**

**Syntax**

IrrSaveScreenShot( filename as zstring)

**Description**

Save a screenshot out to a file, the image format is defined by the extension applied to the filename. Irrlicht currently supports: bmp, png, tga, ppm and jpg

Example

IrrSaveScreenShot( "c:\myscreen.bmp" )

---

**IrrGetScreenShot**

**Syntax**

texture = IrrGetScreenShot( x as integer, y as integer, width as integer, height as integer )

**Description**

Return a pointer to a texture containing a rectangular portion of a screenshot.

Example

DIM texture as irr_texture = IrrGetScreenShot( 0,0, 256,256)

---

**IrrGetTextureInformation**

**Syntax**

texture = IrrGetTextureInformation ( texture as irr_texture, textureWidth as unsigned integer, textureHeight as unsigned integer, texturePitch as unsigned integer,textureFormat as IRR_COLOR_FORMAT )

**Description**
Get information on a texture. The width, height, pitch and color format is returned in the supplied variables.

Example

| IrrGetTextureInformation ( selectedTexture, width, height, pitch, col_format ) |

---

### IrrGetImageInformation

**Syntax**

texture = IrrGetImageInformation ( image as irr_image, textureWidth as unsigned integer, textureHeight as unsigned integer, texturePitch as unsigned integer, textureFormat as IRR_COLOR_FORMAT )

**Description**

Get information on an image. The width, height, pitch and color format is returned in the supplied variables.

**Example**

| IrrGetImageInformation ( selectedImage, width, height, pitch, col_format ) |
Materials

Calls for creating and manipulating materials that can be applied to a node to color and texture the object. Basic Materials set common properties like the shininess and reflective color of the objects. Advanced Materials use GPU programs to create sophisticated texturing effects that can greatly add to the realism of the scene but are only supported by modern graphics cards with Pixel and Vertex shader support. Currently Irrlicht supports Vertex Shaders, Pixel Shaders, ARB Vertex programs, ARB Fragment programs, HLSL (DirectX 9) and GLSL (OpenGL).

### IrrSetNodeAmbientColor

**Syntax**

IrrSetNodeAmbientColor ( node As irr_node, uColor As Uinteger)

**Description**

Sets the ambient color of all materials in a node. This color value is created with the FreeBasic RGBA call. The ambient color is a color applied to the whole node as a simulation of ambient lighting reflected from the objects around it.

**Example**

IrrSetNodeAmbientColor ( object_material, RGBA( 128,0,0,0 ) )

### IrrSetNodeDiffuseColor

**Syntax**


### IrrSetNodeDiffuseColor

**Syntax**

```
IrrSetNodeDiffuseColor ( node As irr_node, uColor As Uinteger)
```

**Description**

Sets the diffuse color of all materials in a node. This color value is created with the FreeBasic RGBA call. The diffuse color is the indirectly lit surface colour.

**Example**

```
IrrSetNodeDiffuseColor ( object_material, RGBA( 128,0,0,0 ) )
```

### IrrSetNodeSpecularColor

**Syntax**

```
IrrSetNodeSpecularColor ( node As irr_node, uColor As Uinteger)
```

**Description**

Sets the specular color of all materials in a node. This color value is created with the FreeBasic RGBA call. The specular color is the color of the highlights on the node representing reflections of light sources.

**Example**

```
IrrSetNodeSpecularColor ( object_material, RGBA( 128,0,0,0 ) )
```

### IrrSetNodeEmissiveColor

**Syntax**

```
IrrSetNodeEmissiveColor ( node As irr_node, uColor As Uinteger)
```

**Description**

Sets the emissive color of all materials in a node. This color value is created with the FreeBasic RGBA call. The emissive colour is the light 'generated within' the node. Setting this to 255,255,255,255 will make the node appear as though it has the no lighting effect applied to it.

**Example**

```
IrrSetNodeEmissiveColor ( object_material, RGBA( 128,0,0,0 ) )
```

### IrrSetNodeColorByVertex
Syntax
IrrSetNodeColorByVertex (material as irr_material, affected_property as IRR_COLOR_MATERIAL)

Description
Sets which aspect of all of the materials in a node is affected by the vertex colour.

affected_property can be one of:

- ECM_NONE
  Don't use vertex color for lighting

- ECM_DIFFUSE
  Use vertex color for diffuse light, (the default value)

- ECM_AMBIENT
  Use vertex color for ambient light

- ECM_EMISSIVE
  Use vertex color for emissive light

- ECM_SPECULAR
  Use vertex color for specular light

- ECM_DIFFUSE_AND_AMBIENT
  Use vertex color for both diffuse and ambient light

Example
IrrSetNodeColorByVertex (object_material, ECM_NONE)
Sets which aspect of the material is affected by the vertex colour.

affected_property can be one of:

- ECM_NONE
  
  Don't use vertex color for lighting

- ECM_DIFFUSE
  
  Use vertex color for diffuse light, (the default value)

- ECM_AMBIENT
  
  Use vertex color for ambient light

- ECM_EMISSIVE
  
  Use vertex color for emissive light

- ECM_SPECULAR
  
  Use vertex color for specular light

- ECM_DIFFUSE_AND_AMBIENT
  
  Use vertex color for both diffuse and ambient light

Example

IrrMaterialVertexColorAffects ( object_material, ECM_NONE )

---

IrrSetMaterialBlend

Syntax

IrrSetMaterialBlend ( material as irr_material, source as IRR_BLEND_FACTOR, destination as IRR_BLEND_FACTOR )

Description

Sets the source and destination surface blend factors for the ONETEXTURE_BLEND material. This is mainly useful in multi-pass rendering, where you render the scene to the display and then render the scene a second time with the ONETEXTURE_BLEND material setting which mixes the existing pixels and the new pixels using the blend setting defined here.
IRR_BLEND_FACTOR can be one of the following values:

- **EBF_ZERO**
  A fixed value of zero

- **EBF_ONE**
  A fixed value of one

- **EBF_DST_COLOR**
  The destination color

- **EBF_ONE_MINUS_DST_COLOR**
  The inverted destination color

- **EBF_SRC_COLOR**
  The source color

- **EBF_ONE_MINUS_SRC_COLOR**
  The inverted source color

- **EBF_SRC_ALPHA**
  The source alpha value

- **EBF_ONE_MINUS_SRC_ALPHA**
  The inverted source alpha value

- **EBF_DST_ALPHA**
  The destination alpha value

- **EBF_ONE_MINUS_DST_ALPHA**
  The inverted destination alpha value

- **EBF_SRC_ALPHA_SATURATE**

Example

IrrSetMaterialBlend ( object_material, EBF_SOURCE_COLOR, EFB_DST_COLOR )
### IrrMaterialSetShininess

**Syntax**

```plaintext
IrrMaterialSetShininess ( material as irr_material, shininess as single )
```

**Description**

Set how shiny the material is, the higher the value the more defined the highlights.

**Example**

```plaintext
IrrMaterialSetShininess ( object_material, 20.0 )
```

### IrrMaterialSetSpecularColor

**Syntax**

```plaintext
IrrMaterialSetSpecularColor ( material as irr_material, Alpha as integer, Red as integer, Green as integer, Blue as integer )
```

**Description**

Set the color of specular highlights on objects with this material applied.

**Example**

```plaintext
IrrMaterialSetSpecularColor ( object_material, 0, 255, 128, 128 )
```

### IrrMaterialSetDiffuseColor

**Syntax**

```plaintext
IrrMaterialSetDiffuseColor ( material as irr_material, Alpha as integer, Red as integer, Green as integer, Blue as integer )
```

**Description**

Set the color of diffuse lighting on objects with this material applied.

**Example**

```plaintext
IrrMaterialSetDiffuseColor ( object_material, 0, 255, 128, 255 )
```

### IrrMaterialSetAmbientColor

**Syntax**

```plaintext
IrrMaterialSetAmbientColor
```
IrrMaterialSetAmbientColor (material as irr_material, Alpha as uinteger, Red as uinteger, Green as uinteger, Blue as uinteger)

Description
Set the color of ambient light reflected by objects with this material applied.

Example
IrrMaterialSetAmbientColor(object_material, 0, 64, 128, 255)

IrrMaterialSetEmissiveColor

Syntax
IrrMaterialSetEmissiveColor (material as irr_material, Alpha as uinteger, Red as uinteger, Green as uinteger, Blue as uinteger)

Description
Set the color of light emitted by objects with this material applied.

Example
IrrMaterialSetEmissiveColor(object_material, 0, 64, 128, 255)

IrrMaterialSetMaterialTypeParam

Syntax
IrrMaterialSetMaterialTypeParam(material as irr_material, value as single)

Description
Set material specific parameter. Used in a couple of vertex alpha and normal mapping material types.

Example
IrrMaterialSetMaterialTypeParam(object_material, 0.357)

IrrSetMaterialLineThickness

Syntax
IrrSetMaterialLineThickness(material as irr_material, thickness as single)

Description
Sets the line thickness of none 3D elements associated with this material.
IrrSetMaterialLineThickness( object_material, 2.0 )

**IrrAddHighLevelShaderMaterial**

**Syntax**

```
irr_shader = IrrAddHighLevelShaderMaterial ( vertex_program as zstring ptr, 
vertex_start_function as zstring ptr, vertex_prog_type as integer, 
pixel_program as zstring ptr, pixel_start_function as zstring ptr, 
pixel_prog_type as integer, material_type as integer )
```

**Description**

Creates a new material using a high level shading language.

- **vertex_program**: String containing the source of the vertex shader program. This can be 0 if no vertex program shall be used.
- **vertex_start_function**: Name of the entry function of the vertex shader program.
- **vertex_program_type**: Vertex shader version used to compile the GPU program.
- **pixel_program**: String containing the source of the pixel shader program. This can be 0 if no pixel shader shall be used.
- **pixel_start_function**: Entry name of the function of the pixel shader program.
- **pixel_program_type**: Pixel shader version used to compile the GPU program.
- **baseMaterial**: Base material which renderstates will be used to shade the material.

Returns a type that contains a material_type number that can be used to shade nodes with this new material. If the shader could not be created it will return 0.

**Example**

```
shader = IrrAddHighLevelShaderMaterial ( _
    vertex_program, "main", IRR_EVST_VS_1_1, _
    pixel_program, "main", IRR_EPST_PS_1_1, _
    IRR_EMT_SOLID )
```

**IrrAddHighLevelShaderMaterialFromFiles**

**Syntax**

```
irr_mesh = IrrAddHighLevelShaderMaterialFromFiles ( 
    vertex_program_filename as zstring ptr, vertex_start_function as zstring ptr, 
```

```
vertex_prog_type as integer, pixel_program_filename as zstring ptr, pixel_start_function as zstring ptr, pixel_prog_type as integer, material_type as integer )

Description
Creates a new material using a high level shading language stored in files.

vertex_program_filename: String containing the filename of the vertex shader program. This can be 0 if no vertex program shall be used.
vertex_start_function: Name of the entry function of the vertex shader program
vertex_program_type: Vertex shader version used to compile the GPU program
pixel_program_filename: String containing the filename of the pixel shader program. This can be 0 if no pixel shader shall be used.
pixel_start_function: Entry name of the function of the pixel shader program
pixel_program_type: Pixel shader version used to compile the GPU program
baseMaterial: Base material which renderstates will be used to shade the material.

Returns a type that contains a material_type number that can be used to shade nodes with this new material. If the shader could not be created it will return 0.

Example

```python
code = IrrAddHighLevelShaderMaterialFromFiles ( _
    ".\media\wood.vertex", "main", IRR_EVST_VS_1_1, _
    ".\media\wood.pixel", "main", IRR_EPST_PS_1_1, _
    IRR_EMT_SOLID )
```

IrrAddShaderMaterial

Syntax

```python
code = IrrAddShaderMaterial ( vertex_program as zstring ptr, pixel_program as zstring ptr, material_type as integer )
```

Description

Creates a new material using a shading language program stored in a string.

vertex program: String containing the source of the vertex shader program. This can be 0 if no vertex program shall be used. For DX8 programs, the will always input registers look like this: v0: position, v1: normal, v2: color, v3: texture coordinates, v4: texture coordinates 2 if available. For DX9 programs, you can
**IrrAddShaderMaterial**

**Syntax**

```cpp
def IrrAddShaderMaterial ( vertex_program, pixel_program, material_type )
```

**Description**

Creates a new material using a shading language program. The `material_type` parameter can be used to specify the shader program.

**vertex_program**: String containing the source of the vertex shader program. This can be 0 if no vertex shader shall be used. For DX8 programs, the input registers look like this: v0: position, v1: normal, v2: color, v3: texture coordinates, v4: texture coordinates 2 if available. For DX9 programs, you can manually set the registers using the `dcl_` statements.

**pixel_program**: String containing the source of the pixel shader program. This can be 0 if no pixel shader shall be used.

**baseMaterial**: Base material which renderstates will be used to shade the material.

**Return**: Returns a type that contains a `material_type` number that can be used to shade nodes with this new material. If the shader could not be created it will return 0.

**Example**

```cpp
material = IrrAddShaderMaterial ( "vertex_program", "pixel_program", IRR_EMT_SOLID )
```

---

**IrrAddShaderMaterialFromFiles**

**Syntax**

```cpp
def IrrAddShaderMaterialFromFiles ( vertex_program_filename, pixel_program_filename, material_type )
```

**Description**

Creates a new material using a shading language program stored in files.

**vertex_program_filename**: String containing the source of the vertex shader program. This can be 0 if no vertex program shall be used. For DX8 programs, the input registers look like this: v0: position, v1: normal, v2: color, v3: texture coordinates, v4: texture coordinates 2 if available. For DX9 programs, you can manually set the registers using the `dcl_` statements.

**pixel_program_filename**: String containing the source of the pixel shader program. This can be 0 if no pixel shader shall be used.

**baseMaterial**: Base material which renderstates will be used to shade the material.

**Return**: Returns a type that contains a `material_type` number that can be used to shade nodes with this new material. If the shader could not be created it will return 0.

**Example**

```cpp
material = IrrAddShaderMaterialFromFiles ( "vertex_program_filename", "pixel_program_filename", IRR_EMT_SOLID )
```
### IrrCreateNamedVertexShaderConstant

**Syntax**

```pascal
result = IrrCreateNamedVertexShaderConstant ( shader as IRR_SHADER, const_name as zstring ptr, const_preset as integer, const_data as single, data_count as integer )
```

**Description**

Creates a Vertex shader constant that allows you to change the value of a constant inside a shader during the execution of the program, simply assign one of the preset constants to the constant name or attach the constant to an array of floats and change the constant simply by changing the values in your array.

Returns: 1 if the constant was successfully created.

**Example**

```pascal
IrrCreateNamedVertexShaderConstant ( shader, "Time", byval IRR_NO_PRESET, @time, 1 )
```

### IrrCreateNamedPixelShaderConstant

**Syntax**

```pascal
result = IrrCreateNamedPixelShaderConstant ( shader as IRR_SHADER, const_name as zstring ptr, const_preset as integer, const_data as single, data_count as integer )
```

**Description**

Creates a Pixel shader constant that allows you to change the value of a constant inside a shader during the execution of the program, simply assign one of the preset constants to the constant name or attach the constant to an array of floats and change the constant simply by changing the values in your array.

Returns: 1 if the constant was successfully created.

**Example**

```pascal
dim color(4) as Single => { 1.0, 1.0, 1.0, 1.0 }
IrrCreateNamedPixelShaderConstant ( shader, "color", IRR_NO_PRESET, @color, 4 )
```
### IrrCreateAddressedVertexShaderConstant

**Syntax**

\[
\text{result} = \text{IrrCreateAddressedVertexShaderConstant}( \text{shader as IRR\_SHADER}, \\
\text{const\_address as integer, const\_preset as integer, const\_data as single,} \\
\text{data\_count as integer})
\]

**Description**

Creates a Vertex shader constant that allows you to change the value of a constant inside a shader during the execution of the program, simply assign one of the preset constants to the constant name or attach the constant to an array of floats and change the constant simply by changing the values in your array.

**Returns:** 1 if the constant was successfully created.

**Example**

\[
\text{IrrCreateAddressedVertexShaderConstant ( shader, 4, IRR\_NO\_PRESET,} \\
\text{@time, 1 )}
\]

### IrrCreateAddressedPixelShaderConstant

**Syntax**

\[
\text{result} = \text{IrrCreateAddressedPixelShaderConstant}( \text{shader as IRR\_SHADER}, \\
\text{const\_address as integer, const\_preset as integer, const\_data as single,} \\
\text{data\_count as integer})
\]

**Description**

Creates a Pixel shader constant that allows you to change the value of a constant inside a shader during the execution of the program, simply assign one of the preset constants to the constant name or attach the constant to an array of floats and change the constant simply by changing the values in your array.

**Returns:** 1 if the constant was successfully created.

**Example**

\[
\text{dim position(3) as Single => } \{ \ 0.0, \ 0.0, \ 0.0 \} \\
\text{IrrCreateAddressedPixelShaderConstant ( shader, 2, IRR\_NO\_PRESET,} \\
\text{@position, 3 )}
\]
### IrrXEffectsStart

**Syntax**

```plaintext
IrrXEffectsStart (  
    vsm as integer = IRR_OFF,  
    softShadows as integer = IRR_OFF,  
    bitdepth32 as integer = IRR_OFF )  
```

**Description**

Starts the XEffects advanced shader extension provided by Bitplane from the Irrlicht Forums. This must be called before any other XEffects calls.

The first parameter 'vsm' is used to turn on the 'Variance Shadow Maps' feature. VSM is an advanced form of shading used to avoid aliasing problems that can be seen with the other shadowing function. It can create clear sharp shadowing. Use IRR_ON to enable this feature.

The second parameter 'soft shadows' provides blurred shadows, similar as those cast by a large source. Use IRR_ON to enable this feature.

The last parameter 'bit depth 32' enables 32 bit buffers for the internal processes. While this will use more video memory it can produce improved results.

**Example**

```plaintext
IrrXEffectsStart ( IRR_OFF, IRR_ON )  
```

### IrrXEffectsEnableDepthPass

**Syntax**

```plaintext
IrrXEffectsEnableDepthPass( enable as integer )  
```

**Description**

Enables a depth rendering pass. This is required for shaders that rely on depth information. Use IRR_ON to enable the function.

**Example**

```plaintext
IrrXEffectsEnableDepthPass ( IRR_ON )  
```
IrrXEffectsAddPostProcessingFromFile

Syntax
IrrXEffectsAddPostProcessingFromFile( name as zstring ptr, effectType as integer = 0 )

Description
Adds a shader feature to the display from a GLSL or HLSL program stored in a file. Shaders do need some programming support so only the XEffects Shaders are supported through the XEffects calls.

The first parameter is the path and file name for the shader program. If you are operating in OpenGL you should use the GLSL extension and when operating in DirectX you should use the HLSL extension.

The second parameter can usually be omitted or set to 0. Only when loading the SSAO shader (not the SSAO composite shader) should it be set to 1.

Example
IrrXEffectsAddPostProcessingFromFile ("./media/shaders/ssao.glsl", 1 )

IrrXEffectsSetPostProcessingUserTexture

Syntax
IrrXEffectsSetPostProcessingUserTexture( texture as irr_texture )

Description
Sets the user defined post processing texture. This is used internally for the SSAO shader but is used primarily for the water shader where it defines the specular surface pattern of the water.

You can change the texture through a sequence of images to produce an animated effect.

Example
IrrXEffectsSetPostProcessingUserTexture ( waterTexture(i) )
**IrrXEffectsAddShadowToNode**

**Syntax**

IrrXEffectsAddShadowToNode( node as irr_node,

    filterType as E_FILTER_TYPE = EFTNONE,
    shadowType as E_SHADOW_MODE = ESM_BOTH )

**Description**

Adds the shadowing effect to a node. This controls both receiving and casting shadows.

The filterType defines the amount of sampling that is to be carried out on the node. This can be one of the following settings, increasing the filter increases the quality and also the cost of rendering.

- EFT_NONE
- EFT_4PCF
- EFT_8PCF
- EFT_12PCF
- EFT_16PCF

The shadow type specifies the type of shadowing applied to the node. This can be set to one of the following settings:

- ESM_RECEIVE
- ESM_CAST
- ESM_BOTH
- ESM_EXCLUDE

**Example**

IrrXEffectsAddShadowToNode ( roomNode )

---

**IrrXEffectsRemoveShadowFromNode**

**Syntax**

IrrXEffectsRemoveShadowFromNode( node as irr_node )

**Description**


Removes the shadowing effect from a node.

Example

```c
IrrXEffectsRemoveShadowFromNode ( roomNode )
```

<table>
<thead>
<tr>
<th><strong>IrrXEffectsExcludeNodeFromLightingCalculations</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>IrrXEffectsExcludeNodeFromLightingCalculations( node as irr_node )</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Excludes a node from shadowing calculations.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
</tr>
<tr>
<td>IrrXEffectsExcludeNodeFromLightingCalculations ( particleNode )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IrrXEffectsAddNodeToDepthPass</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>IrrXEffectsAddNodeToDepthPass( node as irr_node )</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Adds a node to the list of nodes used for calculating the depth pass.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
</tr>
<tr>
<td>IrrXEffectsAddNodeToDepthPass ( barrierNode )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IrrXEffectsSetAmbientColor</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>IrrXEffectsSetAmbientColor( R as uinteger, G as uinteger, B as uinteger, Alpha as uinteger )</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Sets the ambient lighting procuded in the scene by the XEffects system.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
</tr>
<tr>
<td>IrrXEffectsSetAmbientColor ( 32,32,32,0 )</td>
</tr>
</tbody>
</table>
### IrrXEffectsSetClearColor

**Syntax**

```
IrrXEffectsSetClearColor( R as uinteger, G as uinteger, B as uinteger, Alpha as uinteger )
```

**Description**

The XEffects system uses a different background color to the one specified in the IrrBeginScene call. Use this call to set this default background color.

**Example**

```
IrrXEffectsSetClearColor( 255,250,32,0 )
```

### IrrXEffectsAddShadowLight

**Syntax**

```
IrrXEffectsAddShadowLight(  
    shadowDimen as uinteger,  
    posX as single, byVal posY as single, byVal posZ as single,  
    targetX as single, byVal targetY as single, byVal targetZ as single,  
    R as single, byval G as single, byval B as single, byval Alpha as single,  
    lightNearDist as single, byval lightFarDist as single,  
    angleDegrees as single )
```

**Description**

Adds a special dynamic shadow casting light to the scene, for each of these lights that you add there is a separate shadow map created and a separate render pass so for each light you add the scene takes more memory and gets slower.

The first parameter specifies the shadow map resolution for the shadow light. The shadow map is always square, so you need only pass 1 dimension, preferably a power of two between 512 and 2048, maybe larger depending on your quality requirements and target hardware.

The pos parameters specify the lights initial position

The target parameters is the (look at) target for the light

The color setting are the floating point color intensity values of the light

The near and far distance of the light are very important values for determining the reach of the light.
The last parameter is the FOV (Field of view), since the light is similar to a spot light, the field of view will determine its area of influence. Anything that is outside of a lights frustum (Too close, too far, or outside of it's field of view) will be unlit by this particular light, similar to how a spot light works.

Example

IrrXEffectsAddShadowLight ( 512, 200,200,0, 0,0,0, _, 
0.7,0.7,0.6,0.0, 1.0, 1200.0, 89.99 )

**IrrXEffectsSetShadowLightPosition**

**Syntax**

IrrXEffectsSetShadowLightPosition( lightIndex as integer, 
   posX as single, byVal posY as single, byVal posZ as single )

**Description**

Set the position of a shadow light. the index refers to the numerical order in which the lights were added.

Example

IrrXEffectsSetShadowLightPosition ( 0, 200,200,0 )

**IrrXEffectsGetShadowLightPosition**

**Syntax**

IrrXEffectsGetShadowLightPosition( lightIndex as integer, 
   posX as single, byVal posY as single, byVal posZ as single )

**Description**

Get the position of a shadow light. the index refers to the numerical order in which the lights were added.

Example

IrrXEffectsGetShadowLightPosition ( 0, x, y, z )

**IrrXEffectsSetShadowLightTarget**

**Syntax**
IrrXEffectsSetShadowLightTarget( lightIndex as integer,
    targetX as single, byVal targetY as single, byVal targetZ as single )

Description
Set the target location of a shadow light. the index refers to the numerical order
in which the lights were added.

Example
IrrXEffectsSetShadowLightTarget ( 0, 25,15,0 )

IrrXEffectsGetShadowLightTarget

Syntax
IrrXEffectsGetShadowLightTarget( lightIndex as integer,
    targetX as single, byVal targetY as single, byVal targetZ as single )

Description
Get the target location of a shadow light. the index refers to the numerical order
in which the lights were added.

Example
IrrXEffectsGetShadowLightTarget ( 0, x, y, z )

IrrXEffectsSetShadowLightColor

Syntax
IrrXEffectsSetShadowLightColor( lightIndex as integer,
    R as single, byval G as single, byval B as single, byval Alpha as single )

Description
Set the target location of a shadow light. the index refers to the numerical order
in which the lights were added.

Example
IrrXEffectsSetShadowLightColor ( 0, 1.0, 0.75, 0.2, 0.0 )

IrrXEffectsGetShadowLightColor
Syntax

IrrXEffectsGetShadowLightColor( lightIndex as uinteger,
    R as single, byval G as single, byval B as single, byval Alpha as single )

Description
Get the target location of a shadow light. The index refers to the numerical order in which the lights were added.

Example

IrrXEffectsGetShadowLightColor( 0, r, g, b, a )
Scene

Calls for managing the scene, loading and creating mesh objects and then adding them to the scene as nodes to be rendered on the screen.

**IrrGetRootSceneNode**

**Syntax**

```
irr_node = IrrGetRootSceneNode()
```

**Description**

Gets the scenes root node, *all* scene nodes are children of this node

**Example**

```
TheScene = IrrGetRootSceneNode()
```

**IrrGetMesh**

**Syntax**

```
irr_mesh = IrrGetMesh( Filename of the mesh object to load as zstring )
```

**Description**

Loads the specified mesh ready to be added to the scene. The Irrlicht engine supports a wide range of mesh types including BSP, MD2, 3DS, Direct X, etc...

**Example**

```
DolphinMesh = IrrGetMesh( "Dolphin.x" )
```

**IrrCreateMesh**

**Syntax**

```
IrrCreateMesh ( mesh_name as zstring ptr, vertex_count as integer, vertices as IRR_VERT, indices_count as integer, indices as ushort) as irr_mesh
```
Create a new mesh. You must supply a list of vertices of type IRR_VECT and an array of indices that refer to these vertices. The indices are taken in groups of three joining up the dots defined by the vertices and forming a collection of triangles.

Example

PyramidMesh = IrrCreateMesh( "Pyramid", 5, vertices(0), 18, indicies(0))

IrrAddHillPlaneMesh

Syntax

IrrAddHillPlaneMesh ( mesh_name As zString Ptr, tileSizeX As Single, tileSizeY As Single, tileCountX As Integer, tileCountY As Integer, material As uInteger Ptr = 0, hillHeight As Single = 0, countHillsX As Single = 0, countHillsY As Single = 0, textureRepeatCountX As Single = 1, textureRepeatCountY As Single = 1 ) as irr_mesh

Description

Creates a hill plane mesh that represents a simple terrain. Many properties have default values allowing a mesh to be created with a simple call.

Example

TerrainMesh = IrrAddHillPlaneMesh( "Terrain", 1.0, 1.0, 10, 10 )

IrrWriteMesh

Syntax

IrrWriteMesh( mesh as irr_mesh, file_format as IRR_MESH_FILE_FORMAT, save_filename as zstring ) as uinteger

Description

Write the first frame of the supplied animated mesh out to a file using the specified file format. The following file formats are supported by Irrlicht:

- Irrlicht Native mesh writer, for static .irrmesh files.
- EMWT_IRR_MESH
COLLADA mesh writer for .dae and .xml files.
EMWT_COLLADA

STL mesh writer for .stl files.
EMWT_STL

The function will return the follow error codes:

- (0) Could not get mesh writer object
- (1) Could not open file
- (2) Unable to write the mesh to the file
- (3) Successfully wrote file

Example

```
if IrrWriteMesh( custom_mesh, EMWT_IRR_MESH, "mymesh.irr" ) = 3
    ? "Wrote the mesh to file successfully"
```

**IrrRemoveMesh**

**Syntax**

```
IrrRemoveMesh( mesh as irr_mesh )
```

**Description**

Removes a mesh from the scene cache, freeing up resources.

**Example**

```
IrrRemoveMesh( my_mesh )
```

**IrrRenameMesh**

**Syntax**

```
IrrRenameMesh( mesh as irr_mesh, name as zstring ptr )
```

**Description**

Rename a loaded mesh through the scene cache, the mesh can then subsequently be loaded again as a different mesh
### IrrClearUnusedMeshes

**Syntax**

IrrClearUnusedMeshes()

**Description**

Clears all meshes that are held in the mesh cache but not used anywhere else. Any references to these meshes will become invalid.

**Example**

IrrClearUnusedMeshes()

### IrrSetMeshHardwareAccelerated

**Syntax**

IrrSetMeshHardwareAccelerated ( mesh as irr_mesh, frame number as integer )

**Description**

Set the supplied mesh as a Hardware Accelerated object, this offloads the verticies and indicies to hardware support on the graphics card, making the process of rendering those meshes much faster. The feature must be supported on the graphics card and the object must contain over 500 vertices for the operation to be successful. This operation is applied to all mesh buffers in the mesh.

**Example**

IrrSetMeshHardwareAccelerated( ShipMesh, 0 )

### IrrCreateBatchingMesh

**Syntax**

IrrCreateBatchingMesh ( )

**Description**

Create a batching mesh that will be a collection of other meshes into a single
source mesh. The function of the batching mesh is to avoid the use of large numbers of nodes that adds an overhead to the rendering process that can significantly slow it down. Where you have a forest with a thousand trees you will see a significant increase in performance by batching all of those trees into a smaller number of node.

Returns: A batching mesh, while this is handled as an irr_mesh it should only be used with batching mesh commands.

Example

```
batchingMesh = IrrCreateBatchingMesh( )
```

### IrrAddToBatchingMesh

**Syntax**

```
IrrAddToBatchingMesh ( 
    meshBatch as irr_mesh,
    mesh as irr_mesh,
    posX as single = 0.0f, posY as single = 0.0f, posZ as single = 0.0f,
    rotX as single = 0.0f, rotY as single = 0.0f, rotZ as single = 0.0f,
    scaleX as single = 1.0f, scaleY as single = 1.0f, scaleZ as single = 1.0f )
```

**Description**

Adds a mesh to the batching mesh at the specified position, rotation and scale. If each of your meshes requires a different texture you should call IrrSetMeshMaterialTexture for the mesh you are about to add prior to adding the mesh to the batch.

Example

```
IrrAddToBatchingMesh( batchingMesh, treeMesh )
```

### IrrFinalizeBatchingMesh

**Syntax**

```
IrrFinalizeBatchingMesh ( mesh as irr_mesh, frame number as integer )
```

**Description**

Finalises the batching mesh, this should be called once all of the meshes have
been added to the batching mesh. The function returns a new mesh object that can be used in all standard mesh calls.

Example

\[
\text{Dim as irr\_mesh newMesh = IrrFinalizeBatchingMesh( BatchingMesh )}
\]

**IrrGetMeshFrameCount**

**Syntax**

\[
\text{integer = IrrGetMeshFrameCount ( mesh as irr\_mesh )}
\]

**Description**

Gets the number of frames in the supplied mesh. You can use this value to traverse the indicies and vertices in a mesh containing a number of frames.

Example

\[
\text{MeshFrameCount = IrrGetMeshFrameCount( WolfMesh )}
\]

**IrrGetMeshBufferCount**

**Syntax**

\[
\text{integer = IrrGetMeshBufferCount ( mesh as irr\_mesh, frame number as integer )}
\]

**Description**

Gets the number of mesh buffers in the supplied mesh. You can use this value to traverse the indicies and vertices in a mesh containing a number of mesh buffers. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0.

Most meshes only contain one mesh buffer however the artist creating the mesh may decide to break the mesh up into a number of groups of meshes, for example a house might have a roof mesh buffer and a walls mesh buffer.

Example

\[
\text{MeshBufferCount = IrrGetMeshBufferCount( TankMesh, 0 )}
\]
**IrrGetMeshIndexCount**

**Syntax**

```plaintext
integer = IrrGetMeshIndexCount ( mesh as irr_mesh, frame number as integer, mesh_buffer as integer )
```

**Description**

Gets the number of Indicies in the supplied mesh. You can use this value to allocate an array for reading out the list of indicies in a mesh. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

**Example**

```plaintext
MeshIndexCount = IrrGetMeshIndexCount( MapMesh, 0,0 )
```

---

**IrrGetMeshIndices**

**Syntax**

```plaintext
IrrGetMeshIndices ( mesh as irr_mesh, frame number as integer, indicies as ushort, mesh_buffer as integer )
```

**Description**

Gets the list of indices in a mesh and copies them into the supplied buffer. Each index references a vertex in the mesh the indices are grouped into three's and together form a triangular surface. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Indices should be the first element of an array or the first integer in a pool of allocated memory, it is passed into the wrapper by reference as a pointer. You must ensure that the array you supply is large enough to contain all of the indices otherwise an overflow will occur and memory will be corrupted.

**Example**

```plaintext
IrrGetMeshIndices( MapMesh, 0, Indicies(0),0 )
```
**IrrSetMeshIndices**

**Syntax**

```plaintext
IrrSetMeshIndices( mesh as irr_mesh, frame number as integer , indicies as ushort, mesh_buffer as integer )
```

**Description**

This sets the value of the list of indicies in a mesh copying them from the supplied buffer. Each index references a vertex in the mesh the indices are grouped into three's and together form a triangular surface. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Indices should be the first element of an array or the first integer in a pool of allocated memory, it is passed into the wrapper by reference as a pointer. You must ensure that the array you supply is large enough to contain all of the indices otherwise an overflow will occur and erroneous values will be written into the mesh causing unpredictable results.

**Example**

```plaintext
IrrSetMeshIndices( MapMesh, 0, Indicies(0),0)
```

**IrrGetMeshVertexCount**

**Syntax**

```plaintext
integer = IrrGetMeshVertexCount ( mesh as irr_mesh, frame number as integer, mesh_buffer as integer )
```

**Description**

Gets the number of Vertices in the supplied mesh. You can use this value to allocate an array for reading out the list of vertices in a mesh. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.
<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeshVertexCount = IrrGetMeshVertexCount( MapMesh, 0 )</td>
</tr>
</tbody>
</table>

### IrrGetMeshVertexMemory

**Syntax**

```plaintext```
IrrGetMeshVertexMemory ( mesh as irr_mesh, frame number as integer , mesh_buffer as integer )
```

**Description**

Get a memory pointer to the vertex memory for the supplied mesh operations can be carried out very quickly on vertices through this function but object sizes and array access needs to be handled by the caller.

**Example**

```plaintext```
Dim as IRR_VERT verts = IrrGetMeshVertexMemory( MapMesh, 0, 0 )
```

### IrrGetMeshVertices

**Syntax**

```plaintext```
IrrGetMeshVertices ( mesh as irr_mesh, frame number as integer , vertices as IRR_VERT, mesh_buffer as integer )
```

**Description**

Gets the list of vertices in a mesh and copies them into the supplied buffer. Each vertex represents a point in the mesh that is the corner of one of the group of triangles that is used to construct the mesh. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Vertices should be the first element of an array or the first IRR_VERT structure in a pool of allocated memory, it is passed into the wrapper by reference as a pointer. You must ensure that the array you supply is large enough to contain all of the vertices otherwise an overflow will occur and memory will be corrupted.

**Example**

```plaintext```
```
**IrrSetMeshVertices**

**Syntax**

IrrSetMeshVertices( mesh as irr_mesh, frame number as integer, indicies as IRR_VERT, mesh_buffer as integer )

**Description**

This sets the value of the list of vertices in a mesh copying them from the supplied buffer. Each vertex represents a point in the mesh that is the corner of one of the group of triangles that is used to construct the mesh. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specify which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Vertices should be the first element of an array or the first IRR_VERT structure in a pool of allocated memory, it is passed into the wrapper by reference as a pointer. You must ensure that the array you supply is large enough to contain all of the vertices otherwise an overflow will occur and erroneous values will be written into the mesh causing unpredictable results.

**Example**

IrrSetMeshVertices( MapMesh, 0, Verticies(0), 0 )

---

**IrrSetNodeMesh**

**Syntax**

IrrSetNodeMesh( node as irr_node, mesh as irr_mesh )

**Description**

Sets the mesh used by a node created from a mesh model.

**Example**

IrrSetNodeMesh( BuildingNode, LowDetailMesh )
### IrrScaleMesh

**Syntax**
```plaintext
IrrScaleMesh( mesh as irr_mesh, scale as single, frame number as integer = 0, mesh_buffer as integer = 0, source mesh as irr_mesh = 0 )
```

**Description**
Scales the vertices in a mesh without affecting the normals, tangents or texture co-ordinates. This is particularly useful for enlarging a mesh without affecting lighting. It should be noted though that scaling the mesh will scale all of the nodes that use it as their source. The scaling is applied uniformly to all axis.

**Example**
```plaintext
IrrScaleMesh( StatueMesh, 2.0 )
```

### IrrSetMeshMaterialTexture

**Syntax**
```plaintext
IrrSetMeshMaterialTexture(  
    mesh as irr_mesh, 
    byval texture as irr_texture, 
    byval material_index as integer, 
    byval buffer as integer = 0 )
```

**Description**
Apply the supplied texture the specified mesh. Upto four textures can be applied to the material by applying them to different material indices, these textures can be used by materials or shader functions. Setting a mesh texture will apply the texture to all nodes that use that mesh it can also used for texturing a mesh before it is added to a batch mesh.

**Example**
```plaintext
IrrSetMeshMaterialTexture( StatueMesh, stoneTexture, 0 )
```

### IrrSetMeshVertexColors

**Syntax**
```plaintext
IrrSetMeshVertexColors( mesh as irr_mesh, frame number as integer, 
vertexColour as uinteger ptr, vertexGroupStartIndicies as uinteger ptr, 
```
Description
This sets the color of groups of vertices in a mesh. You can define any number of groups of vertices and set the color of those group individually. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Example
DIM color(0 to 2) as integer
color(0) = RGBA(255,0,0,0)
color(1) = RGBA(255,0,0,0)
color(2) = RGBA(255,0,0,0)
DIM start as integer = 0
DIM end as integer = 2
IrrSetMeshVertexColors( MapMesh, 0, @color, @start, @end, 1, 0 )

IrrSetMeshVertexCoords

Syntax
IrrSetMeshVertexCoords( mesh as irr_mesh, frame number as integer , vertexCoordinates as IRR_VECTOR Ptr, vertexGroupStartIndicies as integer ptr, vertexGroupEndIndicies as integer ptr, numberOfGroups as integer, mesh_buffer as integer )

Description
This sets the co-ordinates of groups of vertices in a mesh. You can define any number of groups of vertices and set the color of those group individually. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Example
DIM pos(0 to 2) as IRR_VECTOR
pos(0).x = 0 : pos(0).y = 0 : pos(0).z = 0
pos(1).x = 1 : pos(1).y = 0 : pos(1).z = 0
pos(2).x = 0 : pos(2).y = 1 : pos(2).z = 0
DIM start as integer = 0
DIM end as integer = 2
IrrSetMeshVertexCoords( MapMesh, 0, @color, @start, @end, 1, 0 )

**IrrSetMeshVertexSingleColor**

**Syntax**

IrrSetMeshVertexSingleColor( mesh as irr_mesh, frame number as integer, vertexColour as integer ptr, vertexGroupStartIndicies as integer ptr, vertexGroupEndIndicies as integer ptr, numberOfGroups as integer, mesh_buffer as integer )

**Description**

This sets the color of groups of vertices in a mesh. You can define any number of groups of vertices and set the color of those group individually. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

**Example**

DIM start as integer = 0
DIM end as integer = 2
IrrSetMeshVertexSingleColor( MapMesh, 0, RGBA(255,255,255,255), @start, @end, 1, 0 )

**IrrGetMeshBoundingBox**

**Syntax**

IrrGetMeshBoundingBox( mesh as irr_mesh, min X as single, min Y as single, min Z as single, max X as single, max Y as single, max Z as single )

**Description**

Gets the bounding box of a mesh into the supplied variables, the six parameters define the corners of an axis aligned cube that contains the whole mesh.

**Example**

IrrGetMeshBoundingBox( MapMesh, topX, topY, topZ, bottomX, bottomY, bottomZ )
### IrrAddMeshToScene

**Syntax**

```python
irr_node = IrrAddMeshToScene( Mesh object as irr_mesh )
```

**Description**

Adds a mesh to the scene as a new 3D 'node'.

**Example**

```python
DolphinMesh = IrrGetMesh( "Dolphin.x" )
SceneNode = IrrAddMeshToScene( DolphinMesh )
```

### IrrAddMeshToSceneAsOcttree

**Syntax**

```python
irr_node = IrrAddMeshToSceneAsOcttree( Mesh object as irr_mesh )
```

**Description**

Adds a mesh to the scene as a new 3D 'node'. This method optimises the mesh with an Octtree, this is particularly useful for maps where there is a lot of geometry in the mesh but little of it can be seen at any one time. Optimising your node with this function will result in a large increase in performance.

**Example**

```python
MapMesh = IrrGetMesh( "ctfblue.bsp" )
MapNode = IrrAddMeshToSceneAsOcttree( MapMesh )
```

### IrrAddStaticMeshForNormalMappingToScene

**Syntax**

```python
irr_node = IrrAddStaticMeshForNormalMappingToScene( Mesh object as irr_mesh )
```

**Description**

Adds a mesh to the scene as a static object, the mesh is altered so that it is suitable for the application of a Normal or Parallax mapping material, any animation information is lost.
Example

StoneRoomMesh = IrrGetMesh( "StoneRoom.x" )
SceneNode = IrrAddStaticMeshForNormalMappingToScene( StoneRoomMesh )
IrrSetNodeMaterialTexture( SceneNode, colorMap, 0 )
IrrSetNodeMaterialTexture( SceneNode, normalMap, 1 )
IrrMaterialSetSpecularColor( IrrGetMaterial( SceneNode ), 0, 0, 0 )
IrrSetNodeMaterialType( SceneNode, EMT_PARALLAX_MAP_SOLID )
' adjust the height of the parallax effect
IrrMaterialSetMaterialTypeParam( IrrGetMaterial( SceneNode ), 0.035f )

<table>
<thead>
<tr>
<th><strong>IrrLoadScene</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>IrrLoadScene( file_name As zString Ptr )</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Loads all meshes and creates nodes for a scene defined within a file created by IrrEdit.</td>
</tr>
</tbody>
</table>

Example

IrrLoadScene( "Map1.irr" )

<table>
<thead>
<tr>
<th><strong>IrrSaveScene</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>IrrSaveScene( file_name As zString Ptr )</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Saves the current scene into a file that can be loaded by irrEdit.</td>
</tr>
</tbody>
</table>

Example

IrrSaveScene( "MyScene.irr" )

<table>
<thead>
<tr>
<th><strong>IrrGetSceneNodeFromId</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>Irr_node = IrrGetSceneNodeFromId( id as integer )</td>
</tr>
</tbody>
</table>
Description
Get a scene node based on its ID and returns null if no node is found. This is particularly useful for obtaining references to nodes created automatically when using IrrLoadScene.

Example
IrrNode = IrrGetSceneNodeFromId( 15 )

IrrGetSceneNodeFromName

Syntax
Irr_node = IrrGetSceneNodeFromId( id as zstring ptr )

Description
Get a scene node based on its name and returns null if no node is found. This is particularly useful for obtaining references to nodes created automatically when using IrrLoadScene.

Example
IrrNode = IrrGetSceneNodeFromName( "Box" )

IrrAddBillBoardToScene

Syntax
irr_node = IrrAddBillBoardToScene ( X size of the node as integer, Y size of the node as integer, X position as integer, Y position as integer, Z position as integer)

Description
Adds a billboard to the scene of the specified size and at the specified position. A billboard is a flat 3D textured sprite that always faces towards the camera. You need to texture this element with a separate command.

Example
Billboard = IrrAddBillBoardToScene( 10.0, 8.0, 0, 0, 0 )
**IrrSetBillBoardColor**

**Syntax**

```py
irr_node = IrrSetBillBoardColor ( node as irr_node, topColor as uinteger, bottomColor as integer )
```

**Description**

Set the color of the top and bottom vertices in a billboard applying a vertical graduated shade to it. The colors should be generated with the FreeBasic RGBA function.

**Example**

```py
IrrSetBillBoardColor( Billboard, RGBA(255,255,255,255), RGBA(0,0,0,0) )
```

**IrrSetBillBoardSize**

**Syntax**

```py
irr_node = IrrSetBillBoardSize ( node as irr_node, BillWidth as single, BillHeight as single )
```

**Description**

Adds a billboard to the scene of the specified size and at the specified position. A billboard is a flat 3D textured sprite that always faces towards the camera. You need to texture this element with a separate command.

**Example**

```py
IrrSetBillBoardSize( Billboard, 10.0, 8.0 )
```

**IrrAddBillboardTextSceneNode**

**Syntax**

```py
irr_node = IrrAddBillboardTextSceneNode ( font as irr_font, text as wstring, X size of the node as integer, Y size of the node as integer, X position as integer, Y position as integer, Z position as integer, parent as irr_node, topColor as uinteger, bottomColor as uinteger )
```

**Description**

Adds a text billboard to the scene of the specified size and at the specified
position. A text billboard is a flat 3D textured sprite that always faces towards
the camera and has the supplied text written onto it. You should not texture this element.

font defines the font that is used to generate the text.

text is a wide character string containing the text you want to display on the billboard.

X_size and Y_size define the width and height of the billboard

X, Y and Z define the position of the billboard.

Parent defines the object that is the parent to this billboard, if there is no parent this should be set to IRR_NO_OBJECT

topColor is the colour value of the top of the text on the billboard. This can be created with the RGBA command.

bottomColor is the colour value of the bottom of the text on the billboard. This can be created with the RGBA command.

**Example**

```python
Billboard = IrrAddBillboardTextSceneNode( _
    font, "Hello World", _
    64.0, 12.0, _
    0.0, 40.0, 0.0, _
    parentNode, _
    RGBA(255,255,0,0), _
    RGBA(255,0,0,255))
```

---

**IrrAddParticleSystemToScene**

**Syntax**

```python
node as irr_particle_system = IrrAddParticleSystemToScene ( add_emitter )
```

**Description**

Adds a particle system to the scene as a node, a particle system is an object that creates and manages hundreds of small billboard like objects that are used to represent smoke, rain and other natural effects. Once created you then need to
add emitters and affecters to create and control the particles.

Add emitter can be one of the following values:
- IRR_NO_EMITTER
  For no default emitter (this is probably the option you will use and you will then add a specific emitter later)
- IRR_DEFAULT_EMITTER
  To create a default emitter that ejects a thin vertical stream of particles.

Example
Smoke = IrrAddParticleSystemToScene( IRR_NO_EMITTER )

**IrrAddSkyBoxToScene**

**Syntax**
irr_node = IrrSkyBoxToScene ( up_texture as irr_texture, down_texture as irr_texture, left_texture as irr_texture, right_texture as irr_texture, front_texture as irr_texture, back_texture as irr_texture )

**Description**
Adds a skybox node to the scene this is huge hollow cube that encapsulates the entire scene and has a different texture applied to each of its six surfaces to represent a distant sky or matte scene.

Example
SkyBox = IrrAddSkyBoxToScene( 
    IrrGetTexture("./media/irrlicht2_up.jpg"), 
    IrrGetTexture("./media/irrlicht2_dn.jpg"), 
    IrrGetTexture("./media/irrlicht2_lr.jpg"), 
    IrrGetTexture("./media/irrlicht2_rt.jpg"), 
    IrrGetTexture("./media/irrlicht2_ft.jpg"), 
    IrrGetTexture("./media/irrlicht2_bk.jpg"))

**IrrAddSkyDomeToScene (contributed by Eponasoft)**

**Syntax**
irr_node = IrrAddSkyDomeToScene ( texture as irr_texture, horizontal_res as uinteger, vertical_res as uinteger, texture_percentage as double, sphere_percentage as double, sphere_radius as single )
Description
Adds a skydome node to the scene this is huge hollow sphere (or part of a sphere) that encapsulates the entire scene to represent a distant sky or matte scene. The horizontal and vertical resolution define the number of segments in the mesh of the sphere (setting these too high can quickly produce a very costly mesh). Texture percentage defines the amount of the texture that is mapped to the scene, this should be a value between 0 and 1 (0 being non of the texture and 1 being the whole texture). Finally sphere percentage defines how much of a sphere is created and should be a value between 0 and 2 (0 being none of a sphere, 1 being a hemi-sphere and 2 being a full sphere).

Example
```
SkyBox = IrrAddSkyDomeToScene( IrrGetTexture("./media/domesky.jpg"), 8, 8, 1.0, 2.0, 10000.0 );
```

IrrAddEmptySceneNode

Syntax
```
irr_node = IrrAddEmptySceneNode
```

Description
Adds an empty node to the scene. This is required if you wish to add custom OpenGL commands with no Irrlicht Objects.

Example
```
EmptyNode = IrrAddEmptySceneNode
```

IrrAddTestSceneNode

Syntax
```
irr_node = IrrAddTestSceneNode
```

Description
Adds a simple cube object to the scene. This is particularly useful for testing and is a quick and easy way of playing objects into the scene for testing placement.

Example
IrrAddCubeSceneNode

Syntax
irr_node = IrrAddCubeSceneNode( size as single )

Description
Adds a simple cube object to the scene with the specified dimensions.

Example
MyCube = IrrAddCubeSceneNode( 10.0 )

IrrAddSphereSceneNode

Syntax
irr_node = IrrAddSphereSceneNode( radius as single, poly_count as integer )

Description
Adds a simple sphere object to the scene of the specified radius and with the specified level of detail. A reasonable value for poly_count would be 16. Setting this value too high could produce a very high density mesh and affect your frame rate adversely.

Example
MySphere = IrrAddSphereSceneNode( 0.5, 16 )

IrrAddWaterSurfaceSceneNode (contributed by Eponasoft)

Syntax
irr_node = IrrAddWaterSurfaceSceneNode( mesh as irr_mesh, waveHeight as Single = 2.0, waveSpeed as Single = 300.0, waveLength as Single = 10.0, parent as irr_scene_node = 0, id as Integer = -1, positionX as Single = 0, posY as Single = 0, positionZ as Single = 0, rotationX as Single = 0, rotationY as Single = 0, rotationZ as Single = 0, scaleX as Single = 1.0, scaleY as Single = 1.0, scaleZ as Single = 1.0 )

Description
Adds a mesh with a water animator applied to it, the mesh is animated automatically to simulate a water effect across its surface. Many properties are predefined for this node and a convincing water effect can be created simply by supplying the parameter for the mesh, however the node can be positioned, rotated and scaled by this call and the appearance of the waves on its surface can be adjusted.

Example

```java
waterNode = IrrAddWaterSurfaceSceneNode( pond_mesh )
```

### IrrAddZoneManager

**Syntax**

```java
irr_node = IrrAddZoneManager( initialNearDistance as single, initialFarDistance as single )
```

**Description**

Adds a zone/distance management node to the scene. This simple but very powerful object has no visible geometry in the scene, it is used by attaching other nodes to it as children. When the camera is further away than the far distance and closer than the near distance to the zone manager all of the zones child objects are made invisible. This allows you to group objects together and automatically have them hidden from the scene when they are too far away to see. By using the near distance you could have two sets of objects in the scene one with high detail for when you are close and another with low detail for when you are far away.

Another way to use the zone manager would be to test when your camera is inside the zones bounding box and switch its visibility on and off manually.

Example

```java
zone = IrrAddZoneManager(100,300)
```

### IrrAddClouds (Node by G Davidson)

**Syntax**

```java
irr_node = IrrAddClouds( texture as irr_texture, lod as uinteger, depth as uinteger, density as uinteger )
```
Description
Adds a set of clouds to the scene. These clouds are most appropriate to a cloud effect experienced by a vehicle flying through them and could be of particular use in masking the transition of a spacecraft from an orbital vantage point to a flat terrain object. They do make a nice ordernary cloud effect too but can appear unrealistic when they are directly over the observer.

LOD defines the level of detail in the cloud, more detail is added into the cloud depending on the distance of the observer from the object. depth defines the depth of recursion when creating the cloud and finally density defines the number of clouds in the cloud object.

Example
CloudNode = IrrAddClouds( CloudTexture, 3, 1, 500 )

IrrAddLensFlare (Node by Paulo Oliveira with updates from gammaray and torleif)

Syntax
irr_node = IrrAddLensFlare ( texture as irr_texture )

Description
Adds a lens flare patch object to the scene, this object simulates the effect of bright lights on the optics of a camera., the position of the lens flare can be set and changed with the IrrSetNodePosition command. The lens flare object uses a bitmap containing a series of 128x128 images representing stages of the the lens flare effect.

Example
SceneNode = IrrAddLensFlare( LensTexture )

IrrAddGrass (Node by G Davidson)

Syntax
irr_node = IrrAddGrass ( terrain as irr_terrain, x as integer, y as integer, patchSize as integer, fadeDistance as single, crossed as integer, grassScale as single, maxDensity as uinteger, dataPositionX as integer, dataPositionY as
integer, heightMap as irr_image, textureMap as irr_image, grassMap as irr_image, grassTexture as irr_texture )

Description
Adding a grass object to the scene. Grass objects are associated with terrain and tile terrain objects and are used to place small billboard objects into the scene representing foliage, this implementation of grass creates a large number of grass objects already positioned across the terrain and then dynamically shows or hides them depending on where the camera is within the scene. The grass is also affected with a wind modifier that gently moves the grass as if it were caught in the wind (by setting the speed of the wind to zero the grass will become static and you will see an increase in performance).

The position and size of the patch of grass can be set with x, y, patchSize and grassScale.
FadeDistance controls the distance at which the number of displayed grass elements in that patch are reduced. If this is set to 1.0 then when the camera is inside the patch all of grass will be displayed but once outside less and less will be shown. By increasing this to 2.0 then all of the grass is shown until the camera is two patches distant. This gives a better appearance but reduces performance as more grass has to be drawn.

crossed can be set to either IRR_ON or IRR_OFF. When off each piece of grass is a separate entity with its own position and rotation. When On grass is paired up and placed to form a cross. Crossed grass can have a better appearance as you rotate around it. However individual grass can give the impression that there is more of it and you can therefore reduce the number of grass blades and increase performance.
MaxDensity controls the number of individual clumps of foliage that are created.
Data position X and Y can be used with a large bitmap associated with a tiled terrain and allow the color information to be taken from an offset position on the bitmap.
Heightmap is an image that contains the height of the terrain onto which the grass is placed.
TextureMap is the color map used to color the vertices of the grass and allow you to create areas of dark or light grass, you can use the terrain color map here.
GrassMap is an image used to adjust the height and density of the grass. For example you might have a patch where you don't want to see any grass or a
barren patch where you want short stubble.
GrassTexture is the actually texture used for the grass. This RGBA image is automatically broken up into a number of sections that are used to texture different clumps of grass.

Grass usually looks best when it is closely matched to the color of the terrain and to assist with this a new Material Type has been added IRR_EMT_TRANSPARENT_ADD_ALPHA_CHANNEL_REF that adds the color of grass texture to the color of the grass which is automatically set to the color of the terrain that it lies upon.

Example

```c
grassNode = IrrAddGrass ( Terrain, x, y, 1024, 1, 0, 0, terrainHeight, terrainColor, grassMap, grassTexture )
```

---

### IrrSetShadowColor

**Syntax**

```c
IrrSetShadowColor ( Alpha as integer, Red as integer, Green as integer, Blue as integer )
```

**Description**

Sets the color of shadows cast by objects in the scene. If you are observing a bright scene you might use a light grey shadow instead of a heavy black shadow to add to realism.

**Example**

```c
IrrSetShadowColor( 0, 128, 128, 128 )
```

---

### IrrSetFog

**Syntax**

```c
irr_node = IrrSetFog ( Red as integer, Green as integer, Blue as integer, fogtype as integer, fog_start as single, fog_end as single, density as single )
```

**Description**

Set the properties of fog in the scene.
Red, Green and Blue define the fog color, you should set this to the same color as your sky so the scene fogs out nicely into nothing. These are integer values in the range of 0 to 255

Fogtype specifies whether you want the fog to increase in a linear manner or exponentially - exponential fog usually looks more atmospheric while linear looks more like a dense sea fog. This may be specified as either

IRR_LINEAR_FOG
IRR_EXPONENTIAL_FOG

Fog start and end specify the distance at which the fog starts and the distance at which the fog reaches its maximum density. The values here will depend on the size and scale of the scene.

Density is only used with exponential fog and determines how quickly the exponential change takes place, good values for this range from 0 to 1

Example

```
ThinFog = IrrSetFog ( 240, 255, 255, IRR_EXPONENTIAL_FOG, 0.0, 8000.0, 0.5 )
```

**IrrDraw3DLine**

**Syntax**

```
IrrDraw3DLine( x_start as single, y_start as single, z_start as single, x_end as single, y_end as single, z_end as single, Red as integer, Green as integer, Blue as integer )
```

**Description**

Draws a line onto the display using 3D co-ordinates and a specified color.

**Example**

```
IrrBeginScene( 240, 255, 255 )
IrrDraw3DLine( 0.0, 0.0, 0.0, 0.0, 50.0, 0.0, 0, 255, 0 )
IrrDrawScene
```
**IrrSetSkyDomeColor**

**Syntax**

IrrSetSkyDomeColor( dome as irr_node, horizontalRed as integer, horizontalGreen as integer, horizontalBlue as integer, zenithRed as integer, zenithGreen as integer, zenithBlue as integer )

**Description**

Set the color of the verticies in the skydome. Two colors are defined one for the horizon and another for the top of the sky dome, this simulates the type of coloring effects you see in the sky. If you are using a full spherical skydome the horizontal color will be the color at the bottom of the skydome.

**Example**

' color the skydome so that it is brighter at the horizon and a darker blue at the top of the sky
IrrSetSkyDomeColor( SkyDome, 128, 128, 255, 64, 64, 255 )

**IrrSetSkyDomeColorBand**

**Syntax**

IrrSetSkyDomeColorBand( dome as irr_node, horizontalRed as integer, horizontalGreen as integer, horizontalBlue as integer, bandVerticalPosition as integer, bandFade as single, addative as integer )

**Description**

Creates a horizontal band of color in the skydome, this is mainly useful for creating additional bands of color at the horizon, where your sky is a graduation of blues and then in the morning you have a brighter golden band as the sun rises. The vertical position in the vertex at which you wish to create the band, bandFade defines the amount that the band is faded into the existing skydome color, addative can be IRR_ON to add the color of the band to the existing color of the skydome or IRR_OFF to replace it.

**Example**

' add a band of golden color at the horizon
IrrSetSkyDomeColorBand ( SkyDome, 240, 220, 128, 24, 0.25, IRR_ON )

**IrrSetSkyDomeColorPoint**
Syntax
IrrSetSkyDomeColorPoint( dome as irr_node, Red as integer, Green as integer, Blue as integer, pointXPosition as single, pointYPosition as single, pointZPosition as single, pointRadius as single, pointFade as single, addative as integer )

Description
Set the color of the vertices in the skydome radiating out from a point. This is powerful effect that can be used to color parts of the skydome and create effects to represent the glows of the rising sun or the moon in the sky. The radius is used to limit the distance of the coloring, pointFade defines the amount that the band is faded into the existing skydome color and addative can be IRR_ON to add the color of the band to the existing color of the skydome or IRR_OFF to replace it.

Example
' add a bright golden circle of light at the same point as the rising sun
IrrSetSkyDomeColorPoint( SkyDome, 255,220,96, 1000.0, -250.0, 0.0, 1500.0, 0.75, IRR_ON )

IrrSetZoneManagerProperties
Syntax
IrrAddZoneManager( zoneManager as irr_node, newNearDistance as single, newFarDistance as single, accumulateChildBoxes as integer )

Description
Sets the draw distances of nodes in the zone/distance management node and whether or not the zone manager is to accumulate the bounding boxes of its children as they are added.

Example
IrrSetZoneManagerProperties( zone, 0, 600, IRR_ON )

IrrSetZoneManagerBoundingBox
Syntax
**IrrSetZoneManagerBoundingBox**

Syntax

\[
\text{IrrSetZoneManagerBoundingBox ( zoneManager as irr_node, x as single, y as single, z as single, boxWidth as single, boxHeight as single, boxDepth as single )}
\]

Description

Allows the user to manually set the bounding box of a zone manager node.

Example

\[
\text{IrrSetZoneManagerBoundingBox( zone, \theta, \theta, \theta, 100, 100, 100 )}
\]

---

**IrrSetZoneManagerAttachTerrain**

Syntax

\[
\text{IrrSetZoneManagerAttachTerrain ( zoneManager as irr_node, terrain as irr_terrain, structureMapFile as zstring ptr, colorMapFile as zstring ptr, detailMapFile as zstring ptr, ImageX as integer, ImageY as integer, sliceSize as integer )}
\]

Description

A special feature of the zone manager is its ability to manage tiled terrain nodes, a zone does this by taking position of an attached terrain object that it shares with other zone objects whenever the camera starts to come into range. The terrain object is loaded with new height information, color and detail ready for when it becomes visible to the camera.

The structureMapFile is the name of an RGBA bitmap file that is to be used to set the structure of the terrain. The Alpha channel is used to set the height and the RGB channels are used to set the color of the vertex at that position. This can be used to load lighting into the scene or to load detail map blending into the scene for simple terrain spattering (discussed in the tile terrain section).

The optional color and detail maps are loaded to apply new color and detail maps to the terrain. If either is not used they should be replaced with IRR_NO_OBJECT.

The Image X and Y define the X and Y position of this tile on the structure and color images, so you could load a 1024x1024 structure image and a 1024x1024 detail image in and have your zones form a grid across these large bitmaps.
Finally SliceSize allows you to only process a slice of the terrain on each frame, as a tile is swapped loading in bitmaps and then adjusting what could be 65,000 vertices in a single frame will cause a noticeable hiccup in the smooth running of the scene, so by setting the SliceSize you can define how many rows of the terrain are to be processed on each frame. For example if your tile is 128x128 you might process 32 rows, the tile would then be restructured over 4 frames instead of trying to do it all in one.

Note: You can load your images manually to save them with IrrGetImage and IrrGetTexture and let them stay in memory to avoid having to load images while the scene is running however you should stay aware of how much memory you are using especially the graphics card memory used by IrrGetTexture.

Example

IrrSetZoneManagerAttachTerrain ( Zone(X + Y*ROW_SIZE), Terrain(index), "SunnyValley.tga", "SunnyValley.bmp", IRR_NO_OBJECT, X*112, Y*112, 32 )

### IrrSetGrassDensity

**Syntax**

IrrSetGrassDensity ( grass as irr_node, density as integer, distance as single )

**Description**

Set grass density, density being the number of grass nodes visible in the scene and distance being the distance at which they can be seen.

**Example**

IrrSetGrassDensity ( grassNode, 300, 4000 )

### IrrSetGrassWind

**Syntax**

IrrSetGrassWind ( grass as irr_node, strength as single, resolution as single )

**Description**

Set the grass wind effect, the strength being the strength of the wind, the
resolution being how often the effect is calculated. By setting the resolution to zero the wind effect will be stopped and there will be a performance increase however the wind effect adds significantly to the subtle atmosphere of the scene.

Example

| IrrSetGrassWind ( grassNode, 3.0, 1.0 ) |

### IrrGetGrassDrawCount

**Syntax**
`uinteger = IrrGetGrassDrawCount ( grass as irr_node )`

**Description**
Get the number of grass objects drawn.

**Example**
`VisibleGrass = IrrGetGrassDrawCount( Grass )`

### IrrSetFlareScale

**Syntax**
`IrrSetFlareScale ( flare as irr_node, source as single, optics as single )`

**Description**
Sets the scale of optics in the scene. The source is the texture used to surround the light source while the options are the scale of textures in the optics of the camera. Sometimes it is effected to make the scale of the source considerably larger than those of the optics and to scale the effect in the optics down so that their appearance is more subtle.

**Example**
`IrrSetFlareScale ( FlareNode, 2.0, 1.0 )`

### IrrAddLODManager

**Syntax**
`node = IrrAddLODManager ( fadeScale as uinteger = 4, useAlpha as uinteger =`
Description
Adds a level of detail manager to the scene. The primary use for this node is to add other scene nodes to it as children and have their level of detail controlled automatically. If those nodes are made from loaded meshes different meshes containing different amounts of detail can be displayed at different distances.

The other function of the LOD manager is to fade nodes in and out at a specific distance so they gradually fade rather than disappear abruptly. This is achieved by applying a distance without supplying a mesh.

Example
```
LODManager = IrrAddLODManager( 4, IRR_ON, @NodeChangeCallback )
IrrAddLODMesh( LODManager, 0.0, LOD1Mesh )
IrrAddLODMesh( LODManager, 400.0, IRR_NO_OBJECT )
```

**IrrAddLODMesh**

Syntax
```
IrrAddLODMesh( node as irr_node, distance as single, mesh as irr_mesh )
```

Description
Set the distance at which a particular mesh is to be applied to child mesh nodes. if no mesh is supplied it specifies the distance at which the node should be faded in an out.

- node is the LOD manager node
- distance is the distance at which this effect will be applied
- mesh is the mesh used at this distance and beyond or null to specify the limit of visibility for this node.

Example
LODManager = IrrAddLODMesh( 4, IRR_ON, @NodeChangeCallback )
IrrAddLODMesh( LODManager, 0.0, LOD1Mesh )
IrrAddLODMesh( LODManager, 400.0, IRR_NO_OBJECT )

**IrrSetLODMaterialMap**

Syntax

```c
IrrSetLODMaterialMap( node as irr_node, source as IRR_MATERIAL_TYPES, target as IRR_MATERIAL_TYPES )
```

Description

Specifies which material is used to apply the fade effect for another material type. How this is used will depend on the effect that you want to achieve. By default fading is applied with the IRR_EMT_TRANSPARENT_VERTEX_ALPHA material.

node is the LOD manager node
source is the material type your node uses
target is the material type used for the fade effect.

Example

```c
IrrSetLODMaterialMap( LODManager, IRR_EMT_TRANSPARENT_ADD_COLOR, IRR_EMT_TRANSPARENT_ADD_COLOR)
```

**IrrAddBillBoardGroupToScene**

Syntax

```c
node = IrrAddBillBoardGroupToScene( )
```

Description

Adds a billboard group to the scene. This is a special object that can have billboard like objects added and removed from it and rendered in a very quick and efficient manner. They are all treated as a single object rather than as many individual nodes. This is particularly useful for custom particle effects.

Example

```c
BillboardGroup = IrrAddBillBoardGroupToScene
```
**IrrAddBillBoardToGroup**

**Syntax**

```plaintext
BillboardAddress = IrrAddBillBoardToGroup ( 
    group as irr_node, _
    sizex as single, sizey as single, _
    x as single = 0, y as single = 0, z as single = 0, _
    roll as single = 0, _
    A as uinteger = 255, R as uinteger = 255, G as uinteger = 255, B as uinteger = 255 )
```

**Description**

Adds a billboard to a billboard group. There are a number of properties that are used to specify the billboard.

- `group` is the billboard group node
- `sizex` and `sizey` are the x and y sizes of the billboard
- `x`, `y` and `z` define the position of the billboard
- `roll` specifies the number of degrees that the billboard is spun around its center.
- `A`, `R`, `G` and `B` specify the color used for the billboard

**Example**

```plaintext
BillboardAddress = IrrAddBillBoardToGroup( BillboardGroup, _
    200.0, 200.0, _
    0.0, 0.0, 0.0, _
    0.0, _
    0, 255, 255, 255 )
```

---

**IrrAddBillBoardByAxisToGroup**

**Syntax**

```plaintext
BillboardAddress = IrrAddBillBoardByAxisToGroup ( 
    group as irr_node, _
    sizex as single, sizey as single, _
    x as single = 0, y as single = 0, z as single = 0, _
    roll as single = 0, _
    A as uinteger = 255, R as uinteger = 255, G as uinteger = 255, B as uinteger = 255, _
    axis_x as single = 0, axis_y as single = 0, axis_z as single = 0 )
```

---
**Description**

Adds a billboard to a billboard group that is fixed to a particular axis. These billboards are particularly useful for things like grass. There are a number of properties that are used to specify the billboard.

**Example**

```python
BillboardAddress = IrrAddBillBoardByAxisToGroup( BillboardGroup, 200.0, 200.0, 0.0, 0.0, 0.0, 0.0, 0, 255, 255, 255, 0.0, 1.0, 0.0 )
```

---

**IrrRemoveBillBoardFromGroup**

**Syntax**

```python
IrrRemoveBillBoardFromGroup ( group as irr_node, billboardAddress as any ptr )
```

**Description**

Removes the specified billboard from the billboard group

**Example**

```python
IrrRemoveBillBoardFromGroup ( BillboardGroup, BillboardAddress )
```

---

**IrrBillBoardGroupShadows**

**Syntax**

```python
IrrBillBoardGroupShadows ( group as irr_node, _
    x as single = 1.0, y as single = 0, z as single = 0, _
    intensity as single = 1.0, ambient as single = 0.0 )
```
Description
Applies lighting to the billboards in a cluster of billboards. This can be used for example to shade the particles in a group of billboards representing a cloud.

group is the group of billboards to which the lighting is to be applied.
x, y and z is the direction from which the light is arriving
intensity is the strength of the light
ambient is the strength of ambient light in the billboard group

Example
IrrBillBoardGroupShadows( BillboardGroup, 1.0, 0.0, 0.0, 1.0, 0.5

---

**IrrGetBillBoardGroupCount**

Syntax
uinteger = IrrGetBillBoardGroupCount( group as irr_node )

Description
Get the number of billboards in the billboard group.

Example
count = IrrGetBillBoardGroupCount( BillboardGroup )

---

**IrrBillBoardForceUpdate**

Syntax
IrrBillBoardForceUpdate( group as irr_node )

Description
Unlike regular billboards the billboard group does not always update the orientation of the billboards every frame. If you are a long distance away from the billboard group the camera needs to travel a significant distance before the angle has changed enough to warrent an update of all of the billboards verticies to make them point to the camera once more. You may want to force a refresh at some point with this call.

Example
IrrAddBoltSceneNode (Scene Node created by Sudi with extensions by Trivtn)

Syntax
irr_node = IrrAddBoltSceneNode ( )

Description
The bolt is a special scene node that can be used to replicate electrical effects. This command simply adds the bolt you should then make a call to set the bolts properties. This node can be used to simulate lightning and other electrical effects.

Example

ElectricNode = IrrAddBoltSceneNode( )
IrrSetBoltProperties ( SceneNode, _
    0,90,0, _ ' the start point for the bolt
    0,0,0, _  ' the end point for the bolt
    50, _     ' the bolt updates every 50 milliseconds
    10, _     ' the bolt is 10 units wide
    5, _      ' the bolt is 5 units thick
    10, _     ' there are 10 sub parts in each bolt
    4, _      ' there are 4 individual bolts
    IRR_ON, _ ' the end is not connected to an exact point
    RGBA( 255, 255, 255, 0 ) ) ' Lighting color

IrrSetBoltProperties

Syntax
IrrSetBoltProperties ( 
    bolt as irr_node, _
    startX as single, startY as single, startZ as single, _
    endX as single, endY as single, endZ as single, _
    updateTime as uinteger = 50, _
    radius as uinteger = 10, _
    thickness as single = 5.0, _
    parts as uinteger = 10, _
)
bolts as uinteger = 6, _
steadyend as uinteger = IRR_OFF, _
boltColor as uinteger = RGBA(0,0,255,255))

Description
This sets the properties of a bolt node that simulates an electrical effect. There are a number of properties that control many aspects of the bolt to produce a wide range of appearances..

Start X, Y and Z define the point that the bolt originates from.

End X,Y and Z| define the terminating point for the bolt.

Update time specifies the number of milliseconds between updates to the appearance of the bolt.

Radius is the radius of the entire bolt effect.

Thickness is the thickness of a single electrical element in the bolt.

Parts defines the number of segments the bolt is divided into.

Bolts represents the number of individual electrical arcs that are rendered.

SteadyEnd when set to IRR_ON ends in a tight point, when set to IRR_OFF it ends with the same width as the rest of the bolt.

Color specifies the diffuse color that is applied to the bolt.

Example

```
ElectricNode = IrrAddBoltSceneNode( )
IrrSetBoltProperties ( SceneNode, _
    0,90,0, _       ' the start point for the bolt
    0,0,0, _        ' the end point for the bolt
    50, _           ' the bolt updates every 50
miliseconds
    10, _           ' the bolt is 10 units wide
    5, _            ' the bolt is 5 units thick
    10, _           ' there are 10 sub parts in each
bolt
    4, _            ' there are 4 individual bolts
```
IRR_ON, _  ' the end is not connected to an
exact point
RGBA( 255, 255, 255, 0 ))  ' Lighting color

**IrrAddBeamSceneNode** *(Scene Node by Gaz Davidson (Blindside))*

**Syntax**

```python
irr_node = IrrAddBeamSceneNode()
```

**Description**
The beam is a special scene node that can be used to replicate beam effects like lasers and tracer gun fire. This command simply adds the beam you should then make calls to set the beams properties.

**Example**

```python
BeamNode = IrrAddBeamSceneNode()
IrrSetBeamSize(BeamNode, 5.0)
IrrSetBeamPosition(BeamNode, X,Y,Z, X+100,Y,Z)
```

**IrrSetBeamSize**

**Syntax**

```python
IrrSetBeamSize(beam as irr_node, size as single)
```

**Description**
This call sets the width of a beam node.

**Example**

```python
IrrSetBeamSize(BeamNode, 5.0)
```

**IrrSetBeamPosition**

**Syntax**

```python
IrrSetBeamPosition(beam as irr_node, _
    startX as single, startY as single, startZ as single, _
    endX as single, endY as single, endZ as single)
```

**Description**
This call sets the start and end positions of a beam node. The beam will stretch between the two nodes.
Start X, Y and Z define the point that the bolt originates from.

End X,Y and Z define the terminating point for the bolt.

Example

```cpp
IrrSetBeamPosition ( BeamNode, X,Y,Z, X+100,Y,Z )
```
Scene Nodes

Calls for manipulating, texturing and getting information from nodes in the scene.

### IrrGetNodeName

**Syntax**

```c
const zstring ptr = IrrGetNodeName ( node as irr_node )
```

**Description**

Get the name of the node.

**Example**

```c
NodeName = IrrGetNodeName( StatueNode )
```

### IrrSetNodeName

**Syntax**

```c
IrrSetNodeName ( node as irr_node, name as zstring ptr )
```

**Description**

Set the name of a node

**Example**

```c
IrrSetNodeName( StatueNode, "HeroStatue" )
```

### IrrGetNodeMesh

**Syntax**

```c
irr_mesh = IrrGetNodeMesh ( node as irr_node )
```

**Description**

Get the mesh that is associated with a node
Example

```c
myMesh = IrrGetNodeMesh( StatueNode )
```

### IrrGetMaterialCount

**Syntax**

```c
integer = IrrGetMaterialCount( node as irr_node )
```

**Description**

Get the number of materials associated with a node.

**Example**

```c
nummaterials = IrrGetMaterialCount( StatueNode )
```

### IrrGetMaterial

**Syntax**

```c
irr_material = IrrGetMaterial( node as irr_node, material_index as integer )
```

**Description**

Get the material associated with the node at the particular index.

**Example**

```c
current_material = IrrGetMaterial( StatueNode, index )
```

### IrrSetNodeMaterialTexture

**Syntax**

```c
IrrSetNodeMaterialTexture( node as irr_node, texture as irr_texture, material_index as integer )
```

**Description**

Applys a texture to a node in the scene, how the texture is applied across the surface of the node will depend on the texturing co-ordinates in each of the vectors of the mesh and how they are plotted across the surface of the texture. Some nodes can have several textures applied to them to create special material effects.
Node refers to a node you have added to the scene.

Texture refers to a texture you have loaded from an image file.

Material is the index number of the material layer, this will usually be 0 or 1.

Example

```
IrrSetNodeMaterialTexture(DolphinNode, DolphinTexture, 0)
```

### IrrSetNodeMaterialFlag

**Syntax**

```
IrrSetNodeMaterialFlag( node as irr_node, flag as IRR_MATERIAL_TYPES, value as uinteger )
```

**Description**

Sets material properties of a node that will effect its appearance on the screen, each of these properties can be either switched on or off.

Node refers to a node that has been added to the scene.

Flag is one of the following properties:

- **IRR_EMF_WIREFRAME**
  Render as wireframe outline
- **IRR_EMF_GOURAUD_SHADING**
  Render smoothly across polygons
- **IRR_EMF_LIGHTING**
  Material is effecte by lighting
- **IRR_EMF_ZBUFFER**
  Enable z buffer
- **IRR_EMF_ZWRITE_ENABLE**
  Can write as well as read z buffer
- **IRR_EMF_BACK_FACE_CULLING**
  Cull polygons facing away
- **IRR_EMF_BILINEAR_FILTER**
  Enable bilinear filtering
- **IRR_EMF_TRILINEAR_FILTER**
Enable trilinear filtering
IRR_EMF_ANISOTROPIC_FILTER
Reduce blur in distant textures
IRR_EMF_FOG_ENABLE
Enable fogging in the distance
IRR_EMF_NORMALIZE_NORMALS
Use when scaling dynamically lighted models

The value should be one of the following to switch the property on or off:
- IRR_ON
- IRR_OFF

Example
IrrSetNodeMaterialFlag( CharacterNode, IRR_EMF_GOURAUD_SHADING, IRR_ON )

**IrrSetNodeMaterialType**

**Syntax**
IrrSetNodeMaterialType( node as irr_node, mat_type as IRR_MATERIAL_FLAGS )

**Description**
Set the way that materials are applied to the node.

Node refers to a node that has been added to the scene.

Mat_type is one of the following properties that is applied to the node:
- IRR_EMT_SOLID
  Standard solid rendering uses one texture
- IRR_EMT_SOLID_2_LAYER
  2 blended textures using vertex alpha value
- IRR_EMT_LIGHTMAP
  2 textures: 0=color 1=lighting level and ignores vertex lighting
- IRR_EMT_LIGHTMAP_ADD
... as above but adds levels instead of modulating between them

IRR_EMT_LIGHTMAP_M2
... as above but color levels are multiplied by 2 for brightening

IRR_EMT_LIGHTMAP_M4
... as above but color levels are multiplied by 4 for brightening

IRR_EMT_LIGHTMAP_LIGHTING
2 textures: 0=color 1=lighting level but supports dynamic lighting

IRR_EMT_LIGHTMAP_LIGHTING_M2
... as above but color levels are multiplied by 2 for brightening

IRR_EMT_LIGHTMAP_LIGHTING_M4
... as above but color levels are multiplied by 4 for brightening

IRR_EMT_DETAIL_MAP
2 blended textures: the first is a color map the second at a different scale adds and subtracts from the color to add detail

IRR_EMT_SPHERE_MAP
makes the material appear reflective

IRR_EMT_REFLECTION_2_LAYER
a reflective material blended with a color texture

IRR_EMT_TRANSPARENT_ADD_COLOR
a transparency effect that simply adds a color texture to the background. the darker the color the more transparent it is.

IRR_EMT_TRANSPARENT_ALPHA_CHANNEL
a transparency effect that uses the color textures alpha as a transparency level

IRR_EMT_TRANSPARENT_ALPHA_CHANNEL_REF
a transparency effect that uses the color textures alpha, the pixel is only drawn if the alpha is > 127. this is a fast effect that does not blur edges and is ideal for leaves & grass etc.
IRR_EMT_TRANSPARENT_VERTEX_ALPHA
a transparency effect that uses the vertex alpha value

IRR_EMT_TRANSPARENT_REFLECTION_2_LAYER
a transparent & reflecting effect. the first texture is a reflection map, the second a color map. transparency is from vertex alpha

IRR_EMT_NORMAL_MAP_SOLID
A solid normal map renderer. First texture is color, second is normal map. Only use nodes added with IrrAddStaticMeshForNormalMappingToScene. Only supports nearest two lights. Requires vertex and pixel shaders 1.1

IRR_EMT_NORMAL_MAP_TRANSPARENT_ADD_COLOR
... as above only with a transparency effect that simply adds the color to the background. the darker the color the more transparent it is.

IRR_EMT_NORMAL_MAP_TRANSPARENT_VERTEX_ALPHA
... as above only with a transparency effect that uses the vertex alpha value

IRR_EMT_PARALLAX_MAP_SOLID
similar to the solid normal map but more realistic providing virtual displacement of the surface. Uses the alpha channel of the normal map for height field displacement. Requires vertex shader 1.1 and pixel shader 1.4.

IRR_EMT_PARALLAX_MAP_TRANSPARENT_ADD_COLOR
... as above only with a transparency effect that simply adds the color to the background. the darker the color the more transparent it is.

IRR_EMT_PARALLAX_MAP_TRANSPARENT_VERTEX_ALPHA
... as above only wiht a transparency effect that uses the vertex alpha value

Example
IrrSetNodeMaterialType( WaterNode, IRR_EMT_LIGHTMAP )

**IrrSetNodePosition**

**Syntax**
IrrSetNodePosition( node as irr_node, X as single, Y as single, Z as single )
Description
Moves the node to the new position.

Example
IrrSetNodePosition( CharacterNode, 500.0, 100.7, -192.6 )

IrrSetNodeRotation

Syntax
IrrSetNodeRotation( node as irr_node, X as single, Y as single, Z as single )

Description
Rotate a node to the specified orientation through its X, Y and Z axis

Example
IrrSetNodeRotation( CharacterNode, 34.5 0.76, -67.3 )

IrrSetNodeScale

Syntax
IrrSetNodeScale( node as irr_node, X as single, Y as single, Z as single )

Description
Change the scale of a node in the scene making it bigger or smaller in the X, Y and Z axis

Example
IrrSetNodeScale( CharacterNode, 1.2,1.5,1.2 )

IrrSetNodeRotationPositionChange

Syntax
IrrSetNodeRotationPositionChange( node as irr_node, roll as single, pitch as single, yaw as single, drive as single, strafe as single, elevate as single, forwardVector as IRR_VECTOR ptr, upVector as IRR_VECTOR ptr, offsetVectorCount as integer, offsetVectors as IRR_VECTOR ptr )
Apply a change in rotation and a directional force. We can also optionally recover pointers to a series of vectors. The first is a pointer to a vector pointing forwards the second is a pointer a vector pointing upwards following this are any number of points that will also be rotated (the effect on these points is NOT accumulative so the points should be initialised with their original values each time this is called)

Example

**IrrSetNodeRotationPositionChange( SceneNode, roll, pitch, yaw, drive, strafe, elevate, @forwardVector, @upVector, 2, @cameraVector(0))**

---

**IrrDebugDataVisible**

**Syntax**

IrrDebugDataVisible ( node as irr_node, visible as integer )

**Description**

Displays debugging data around a node, this typically means drawing the bounding box around the edges of the node.

There are a series of values for displaying different types of debugging information and not all of them are supported on all node types

0 No Debugging
1 Bounding Box
2 Normals
4 Skeleton
8 Wireframe
16 Transparency
32 Bounding Box Buffers
&hffffffff Everything

Example

**IrrDebugDataVisible ( PyramidNode, 1 )**

---

**IrrGetNodePosition**
### Syntax

**IrrGetNodePosition**

```
IrrGetNodePosition( node as irr_node, X as single, Y as single, Z as single )
```

**Description**

Gets the position of a node in the scene and stores its X, Y and Z co-ordinates into the supplied variables.

**Example**

```
IrrGetNodePosition( CharacterNode, XPosition, YPosition, ZPosition )
```

### Irr getNodeAbsolutePosition

**Syntax**

```
IrrGetNodeAbsolutePosition( node as irr_node, X as single, Y as single, Z as single )
```

**Description**

Get the absolute position of the node in the scene this position includes the position changes of all of the nodes parents too.

**Example**

```
IrrGetNodeAbsolutePosition( CharacterNode, XPosition, YPosition, ZPosition )
```

### IrrGetNodeRotation

**Syntax**

```
IrrGetNodeRotation( node as irr_node, X as single, Y as single, Z as single )
```

**Description**

Get the rotation of a node in the scene and stores the X, Y and Z rotation values in the supplied variables.

**Example**

```
IrrGetNodeRotation( CharacterNode, XRotation, YRotation, ZRotation )
```
### IrrGetNodeScale

**Syntax**

\[
\text{IrrGetNodeScale( node as irr_node, X as single, Y as single, Z as single )}
\]

**Description**

Get the scale of a node in the scene and stores the X, Y and Z scale values in the supplied variables..

**Example**

\[
\text{IrrGetNodeScale( CharacterNode, XScale, YScale, ZScale )}
\]

### IrrGetJointNode

**Syntax**

\[
\text{irr_node = IrrGetJointNode ( node as irr_node, joint_name as zstring ptr )}
\]

**Description**

This supplies you with an invisible node that follows the motion of a particular joint in an animated models skeleton. You can use this to attach child nodes that represent objects a person is carrying for example. (This call now replaces IrrGetMS3DJointNode and IrrGetDirectXJointNode which are only supplied for backwards compatibility). It can now also be used to manually move the joint.

The name should refer to the name of a joint in the model.

**Example**

\[
\text{HandNode = IrrGetJointNode( CharacterNode, "LeftHand" )}
\]

### IrrAddChildToParent

**Syntax**

\[
\text{IrrAddChildToParent ( child as irr_node, parent as irr_node )}
\]

**Description**

Attaches the child node to the parent node, whenever you change the parent node the child node changes too. This is useful for putting a cup in a characters hand for example. You can move and rotate the child node to move the object.
into position against its parent.

Example
IrrAddChildToParent( CupNode, CharacterNode)

**IrrGetNodeFirstChild**

Syntax
irr_node = IrrGetNodeFirstChild ( node as irr_node, searchPosition as any ptr )

Description
Get the first child node of this node, returns 0 if there is no child.

Example
ChildNode = IrrGetNodeFirstChild ( SectorNode, position )

**IrrGetNodeNextChild**

Syntax
irr_node = IrrGetNodeNextChild ( node as irr_node, searchPosition as any ptr )

Description
Get the next child node of this node, returns 0 if there is no child.

Example
ChildNode = IrrGetNodeNextChild( SectorNode, position )

**IrrIsNodeLastChild**

Syntax
integer = IrrIsNodeLastChild ( child as irr_node, parent as irr_node )

Description
Attaches the child node to the parent node, whenever you change the parent node the child node changes too. This is useful for putting a cup in a characters hand for example. You can move and rotate the child node to move the object into position against its parent.
Example

```c
if IrrIsNodeLastChild( SectorNode, position ) = 0 then
    LastNode = IRR_YES
end if
```

### IrrAddNodeShadow

Syntax

```
irr_node = IrrAddNodeShadow ( node as irr_node, mesh as irr_mesh = 0 )
```

Description

Adds shadows to a node that are cast across other nodes in the scene, shadowing need to be turned on when you call IrrStart. You should analyse the performance of your scene carefully when using this function as it can have a significant effect on your frame rate. You can supply a different mesh to the one used to display the node, this shadow mesh could be a much lower resolution than that used for your model thereby improving performance.

Example

```
IrrAddNodeShadow ( CharacterNode )
```

### IrrSetNodeVisibility

Syntax

```
IrrSetNodeVisibility ( node as irr_node, visible as integer )
```

Description

This allows you to hide nodes from the display so you can quickly and easily switch objects out to improve performance or create effects like one node transforming into another node (perhaps in a puff of particle smoke).

Visible can be one of the following values: -
- IRR_INVISIBLE
- IRR_VISIBLE

Example

```
IrrSetNodeVisibility( CharacterNode, IRR_VISIBLE )
```
**IrrRemoveNode**

Syntax
IrrRemoveNode( node as irr_node )

Description
Removes a node from the scene deleting it.

Example
IrrRemoveNode( CharacterNode )

---

**IrrRemoveAllNodes**

Syntax
IrrRemoveAllNodes()

Description
Clears the entire scene, any references to nodes in the scene will become invalid.

Example
IrrRemoveAllNodes()

---

**IrrSetNodeParent**

Syntax
IrrSetNodeParent( node as irr_node, parent as irr_node )

Description
Sets the parent of the specified node.

Example
ParentNode = IrrSetNodeParent( ChildNode, ParentNode )

---

**IrrGetNodeParent**

Syntax
irr_node = IrrGetNodeParent( node as irr_node )
Description
Gets the parent of the specified node.

Example

```
ParentNode = IrrGetNodeParent( ChildNode )
```

---

**IrrGetNodeID**

Syntax

```
integer = IrrGetNodeID( node as irr_node )
```

Description

Each node can have a 32 bit signed identification number assigned to them this can be used in collision operations to filter out particular classes of object.

Example

```
NodeID = IrrGetNodeID( TreeNode )
```

---

**IrrSetNodeID**

Syntax

```
IrrSetNodeID( node as irr_node, id as integer )
```

Description

Adds a simple cube object to the scene. This is particularly useful for testing and is a quick and easy way of playing objects into the scene for testing placement.

Example

```
IrrSetNodeID( TreeNode, 8 )
```

---

**IrrGetNodeBoundingBox**

Syntax

```
integer = IrrGetNodeBoundingBox( node as irr_node, x1 as single, y1 as single, z1 as single, x2 as single, y2 as single, z2 as single, )
```

Description
Gets the coordinates describing the bounding box of the node into the six supplied variables.

Example

\[
\text{NodeID} = \text{IrrGetNodeBoundingBox}( \text{BuildingNode}, Xa, Ya, Za, Xb, Yb, Zb )
\]

\[\text{IrrGetNodeTransformedBoundingBox}\]

Syntax

integer = IrrGetNodeTransformedBoundingBox ( node as irr_node, x1 as single, y1 as single, z1 as single, x2 as single, y2 as single, z2 as single, )

Description

Gets the transformed (absolute value) bounding box of a node into the six supplied variables. So if your node has been moved hundreds of units away from the origin the co-ordinates of its bounding box will also be hundreds of units away corresponding to its real location in the scene.

Example

\[
\text{NodeID} = \text{IrrGetNodeTransformedBoundingBox}( \text{BuildingNode}, Xa, Ya, Za, Xb, Yb, Zb )
\]
Animation

Calls that control the animation of nodes in the scene either by playing animation that is embedded in the mesh or applying animator controls to automatically effect the nodes.

**IrrSetNodeAnimationRange**

Syntax

```cpp
IrrSetNodeAnimationRange( node as irr_node, Start Frame as integer, End Frame as integer )
```

Description

Sets the range of animation that is to be played in the node. An animation sequences might run from 0 to 200 frames and a sequence where your character is running might only occupy a portion of this.

Example

```cpp
IrrSetNodeAnimationRange( CharacterNode, 50, 75 )
```

**IrrPlayNodeMD2Animation**

Syntax

```cpp
IrrPlayNodeMD2Animation( node as irr_node, sequence as uinteger )
```

Description

MD2 format models have specific animation sequences contained within them that can be played back with a simple call.

sequence should be one of the following values: -
- IRR_EMAT_STAND
- IRR_EMAT_RUN
- IRR_EMAT_ATTACK
Example

IrrPlayNodeMD2Animation( CharacterNode, IRR_EMAT_STAND )

**IrrSetNodeAnimationSpeed**

**Syntax**

IrrSetNodeAnimationSpeed( node as irr_node, speed as integer )

**Description**

Change the speed at which an animation is played for a node. You could use this to make a character run slowly or quickly and still keep its feet on the ground.

**Example**

IrrSetNodeAnimationSpeed( CharacterNode, 25 )

**IrrGetNodeAnimationFrame**

**Syntax**
uinteger = IrrGetNodeAnimationFrame( node as irr_node )

Description
Get the frame number that is currently being played by the node.

Example
CurrentFrame = IrrGetNodeAnimationFrame( AnimNode )

**IrrSetNodeAnimationFrame**

Syntax
IrrSetNodeAnimationFrame( node as irr_node, frame as integer )

Description
Set the current frame number being played in the animation

Example
IrrSetNodeAnimationFrame( CharacterNode, 75 )

**IrrSetTransitionTime**

Syntax
IrrSetTransitionTime ( node as irr_node, speed as single )

Description
Sets the transition time across which two poses of an animated mesh are blended. For example a character in a sitting pose can be switched into a lying down pose by blending the two frames, this will provide a more convincing smooth transition instead of a snap change in position. IrrAnimateJoints must be called before IrrDrawScene if blending is used.

Example
IrrSetTransitionTime( CharacterNode, 0.75 )

**IrrAnimateJoints**

Syntax
IrrAnimateJoints( node as irr_node )
Animations the mesh based on the position of the joints, this should be used at the end of any manual joint operations including blending and joints animated using IRR_JOINT_MODE_CONTROL and IrrSetNodeRotation on a bone node.

Example

IrrAnimateJoints( CharacterNode )

**IrrSetJointMode**

**Syntax**

IrrSetJointMode ( node as irr_node, mode as integer )

**Description**

Sets the animation mode of joints in a node. When using the control mode IrrAnimateJoints must be called before IrrDrawScene.

IRR_JOINT_MODE_NONE will result in no animation of the model based on bones
IRR_JOINT_MODE_READ will result in automatic animation based upon the animation defined with calls like IrrSetNodeAnimationRange
IRR_JOINT_MODE_CONTROL will allow the position of the bones to be set through code

Example

IrrSetJointMode( CharacterNode, IRR_JOINT_MODE_CONTROL )

**IrrAddCollisionAnimator**

**Syntax**

irr_animator = IrrAddCollisionAnimator ( selector as irr_selector, node as irr_node, radius x as single, radius y as single, radius z as single, gravity x as single, gravity y as single, gravity z as single, offset x as single, offset y as single, offset z as single )

**Description**
This adds a collision animator to a node that applies collision detection and gravity to the object. The collision detection will stop the object penetrating through a surface in the objects it is colliding against and will also press it against the surface using gravity.

Selector represents a selection of triangles in the scene, this is usually all of the triangles in a map for instance. Please refer to the section on collision for further details of how to obtain this object.

Radius X, Radius Y and Radius Z define an ellipsoid that defines the area of collision this elliptical shape allows the collision detection to slide the object up steps and even ladders. If you make it too big you might be too large to get through a doorway but if you make it too small you may not be able to climb steps. You should play with these values and find the best ones for your scene.

Gravity X, Gravity Y and Gravity Z work together to specify the force that is applied to the node to make it drop to the ground. Other values could be used to simulate wind effects.

Offset X, Offset Y and Offset Z are used to offset the node by a specific distance from the center of the collision, as the center of the object and the size of your collision ellipsoid vary you can use this to adjust the position of the node and to bring it into contact with the ground.

Example

| CollisionAnimator = IrrAddCollisionAnimator( MapCollision, CameraNode, 30.0,30.0,30.0, 0.0,-3.0,0.0, 0.0,50.0,0.0 ) |

**IrrAddDeleteAnimator**

Syntax

```java
irr_animator = IrrAddDeleteAnimator ( node as irr_node, milliseconds to deletion as integer )
```

Description

This animator deletes the node it is attached to after the specified number of milliseconds (1/1000ths of a second). You could use this animator to delete a falling rock for example, all you would need to do is attach the delete animator, a movement animator and then forget about it.
Example
DeleteAnimator = IrrAddDeleteAnimator( RockNode, 3000 )

IrrAddFadeAnimator
Syntax
irr_animator = IrrAddFadeAnimator( node as irr_node, milliseconds to deletion as integer, scale as single )

Description
This animator deletes the node it is attached to after the specified number of milliseconds (1/1000ths of a second). During the time while it is waiting to delete it the node is slowly faded to invisibility and is also scaled by the specified amount. You could use this animator to fade and delete an object from a scene that was no longer required like a used medical pack, all you would need to do is attach the fade animator and forget about it.

Example
FadeAnimator = IrrAddFadeAnimator( MedicalNode, 3000, 0.0 )

IrrAddFlyCircleAnimator
Syntax
irr_animator = IrrAddFlyCircleAnimator( node as irr_node, center x as single, center y as single, center z as single, radius as single, speed as single )

Description
This animator moves the node it is attached to in a circular path.

Center X, Center Y and Center Z define the center of the circular path.

Radius defines the radius of the path.

Speed defines the rate the node moves around the circular path.

Example
CircleAnimator = IrrAddFlyCircleAnimator( PowerNode, 0,0,0 50, 20 )
**IrrAddFlyStraightAnimator**

Syntax

```c
irr_animator = IrrAddFlyStraightAnimator ( node as irr_node, start x as single, start y as single, start z as single, end x as single, end y as single, end z as single, time to complete as uinteger, loop path as integer )
```

Description

This animator makes the node it is attached to move in a straight line from the start to the end end point. It would be useful for objects moving on a conveyor belt for example.

Start X, Start Y and Start Z specify the start point of the path.

End X, End Y and End Z specify the end point of the path.

Time to complete specifies the number of milliseconds the animator will take to move the node from the start to the end point.

Loop path determines if the node will be moved from the start to the end and then stopped or whether the animation will be looped this parameter should be either:

- **IRR_ONE_SHOT**
  For a single animation and then stop
- **IRR_LOOP**
  To continuously repeat the animation

Example

```c
FlyAnimator = IrrAddFlyStraightAnimator( AnimatedBox, 0,50,-300, 0,50,300, 3000, IRR_LOOP )
```

**IrrAddRotationAnimator**

Syntax

```c
irr_animator = IrrAddRotationAnimator ( node as irr_node, x as single, y as single, z as single )
```
This animator makes the node it is attached to spin around.

X, Y and Z specify the number of radians the object is spun around each axis.

**Example**

```c
RotationAnimator = IrrAddRotationAnimator( DisplayCaseNode, 0, 0.1, 0 )
```

---

**IrrAddSplineAnimator**

**Syntax**

```c
irr_animator = IrrAddSplineAnimator( node as irr_node, array size as integer, x as single, y as single, z as single, time to start as integer, speed as single, tightness as single )
```

**Description**

This is one of the more difficult to set up of the animators but is very natural looking and powerful. A spline is a curved line that passes through or close to a list of co-ordinates, creating a smooth flight. This animator needs a list of coordinates stored in three arrays, one array each for the X, Y and Z locations of all the points. A good way to get co-ordinates for these arrays is to load in the camera position example program and move your camera to a point and write down its co-ordinates.

Array size specifies how many points there are in your spline motion.

The three arrays X, Y and Z containing co-ordinates are passed by reference as a pointer therefore you should ensure that the array is the correct size otherwise unpredictable results will be obtained.

Time to start specifies the number of milliseconds that must pass before the animation starts.

Speed defines the rate the node moves along the spline curve.

Tightness specifies how tightly the curve is tied to the points (0 is angular and 1 is very loose)
Example

SplineX(0) = -100 : SplineY((0) = 50 : SplineZ((0) = 0
SplineX(1) = 0 : SplineY((1) = 100 : SplineZ((1) = -100
SplineX(2) = 100 : SplineY((2) = 50 : SplineZ((2) = 0
SplineX(3) = 0 : SplineY((3) = 100 : SplineZ((3) = 100
SplineAnimator = IrrAddSplineAnimator( CameraNode, 4,
SplineX(0), SplineY(0), SplineZ(0), 0, 0.5, 1)

**IrrRemoveAnimator**

Syntax

IrrRemoveAnimator ( node as irr_node, node as irr_animator )

Description

This removes an animator from a node. Stopping the animation or cleaning an animator up so you can apply a new one.

Example

IrrRemoveAnimator( DoorNode, FlyAnimator )
Collision

Calls for creating collision groups and for calculating collisions in the scene.

**IrrGetCollisionGroupFromMesh**

Syntax

```python
irr_selector = IrrGetCollisionGroupFromMesh ( mesh as irr_mesh, node as irr_node )
```

Description

Creates a collision object from the triangles contained within the specified mesh as applied to the position, rotation and scale of the supplied node.

Example

```python
ObjectSelector = IrrGetCollisionGroupFromMesh( SimpleBuildingMesh, MyBuilding )
```

**IrrGetCollisionGroupFromComplexMesh**

Syntax

```python
irr_selector = IrrGetCollisionGroupFromComplexMesh ( mesh as irr_mesh, node as irr_node)
```

Description

Creates an optimized triangle selection group from a large complex mesh like a map. This group can then be used in collision functions to collide objects against this node. You need to supply both the mesh the node was created from and the node itself.

Example

```python
MapSelector = IrrGetCollisionGroupFromComplexMesh( MapMesh, MapNode )
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IrrGetCollisionGroupFromBox</strong></td>
<td><code>irr_selector = IrrGetCollisionGroupFromBox ( node as irr_node )</code></td>
<td>Creates a collision object from the bounding box of a node.</td>
<td><code>ObjectSelector = IrrGetCollisionGroupFromBox( CharacterNode )</code></td>
</tr>
<tr>
<td><strong>IrrGetCollisionGroupFromTerrain</strong></td>
<td><code>irr_selector = IrrGetCollisionGroupFromTerrain ( node as irr_node, level of detail as integer )</code></td>
<td>Creates a collision object from a terrain node. A higher level of detail improves the collision detection but consumes more resources and can effect the speed of the process.</td>
<td><code>TerrainSelector = IrrGetCollisionGroupFromTerrain( TerrainNode, 1 )</code></td>
</tr>
<tr>
<td><strong>IrrRemoveCollisionGroup</strong></td>
<td><code>IrrRemoveCollisionGroup ( collisionGroup as irr_selector, node as irr_node )</code></td>
<td>Remove the collision selector from memory. This collision selector must not be attached to another collision group when it is removed, the collision group is first removed from the node you supply.</td>
<td><code>IrrRemoveCollisionGroup( buildingCollision, buildingNode )</code></td>
</tr>
</tbody>
</table>
### IrrAttachCollisionGroupToNode

**Syntax**

```cpp
IrrAttachCollisionGroupToNode ( collisionGroup as irr_selector, node as irr_node )
```

**Description**

Attaches a collision group that you have already created from a mesh and a node to another node without duplicating the collision geometry.

**Example**

```cpp
IrrAttachCollisionGroupToNode( boxCollision, anotherBoxNode )
```

### IrrSetNodeTriangleSelector

**Syntax**

```cpp
IrrSetNodeTriangleSelector ( node as irr_node, collisionGroup as irr_selector )
```

**Description**

Assigns a collision group to a specific node.

**Example**

```cpp
IrrSetNodeTriangleSelector( newBuilding, buildingCollision )
```

### IrrCreateCombinedCollisionGroup

**Syntax**

```cpp
irr_selector = IrrCreateCombinedCollisionGroup
```

**Description**

Creates a collision object that can be used to combine several collision objects together so you could add a couple of maps and a terrain for example. Initially the combined collision object is empty.

**Example**

```cpp
SelectorGroup = IrrCreateCombinedCollisionGroup
```
### IrrAddCollisionGroupToCombination

**Syntax**
IrrAddCollisionGroupToCombination ( combined collision group as irr_selector, collision group as irr_selector )

**Description**
Adds a collision object to group of collision objects.

**Example**
IrrAddCollisionGroupToCombination( SelectorGroup, MapSelector )
IrrAddCollisionGroupToCombination( SelectorGroup, TerrainSelector )

### IrrRemoveAllCollisionGroupsFromCombination

**Syntax**
IrrRemoveAllCollisionGroupsFromCombination ( combined collision group as irr_selector )

**Description**
Empty a collision group object so that you can add different collision groups to it.

**Example**
IrrRemoveAllCollisionGroupsFromCombination( SelectorGroup )

### IrrRemoveCollisionGroupFromCombination

**Syntax**
IrrRemoveCollisionGroupFromCombination ( combined collision group as irr_selector, collision group as irr_selector )

**Description**
Remove a single specified collision object from a group of collision objects.

**Example**
IrrRemoveCollisionGroupFromCombination( SelectorGroup, TerrainSelector )
**IrrGetCollisionPoint**

Syntax
integer = IrrGetCollisionPoint ( start as IRR_VECTOR, line_end as IRR_VECTOR, collision group as irr_selector, collision point as IRR_VECTOR )

Description
Detect the collision point of a ray in the scene with a collision object if a collision was detected 1 is returned and vector collision contains the co-ordinates of the point of collision

Start defines the start point of the ray and End defines the endpoint

Collision group is a selector object created with one of the above functions.

Collision point is the co-ordinates in 3D space of the collision object the ray and the selector object.

Example
collided = IrrGetCollisionPoint ( StartVector, EndVector, CharacterSelector, CollisionVector )

**IrrGetRayFromScreenCoordinates**

Syntax
IrrGetRayFromScreenCoordinates ( screen x as integer, screen y as integer, camera as irr_camera, ray start as IRR_VECTOR, ray end as IRR_VECTOR )

Description
Gets a ray that goes from the specified camera and through the screen coordinates the information is copied into the supplied start and end vectors. You can then use this ray in other collision operations.

Example
IrrGetRayFromScreenCoordinates ( screen_x, screen_y, CameraNode, StartVector, EndVector )
### IrrGetCollisionNodeFromCamera

**Syntax**

```plaintext
irr_node = IrrGetCollisionNodeFromCamera ( camera as irr_camera )
```

**Description**

A ray is cast through the camera and the nearest node that is hit by the ray is returned. If no node is hit zero is returned for the object.

**Example**

```plaintext
TargetedNode = IrrGetCollisionNodeFromCamera ( CameraNode )
```

### IrrGetCollisionNodeFromRay

**Syntax**

```plaintext
irr_node = IrrGetCollisionNodeFromRay ( start as IRR_VECTOR, line_end as IRR_VECTOR )
```

**Description**

A ray is cast through the supplied coordinates and the nearest node that is hit by the ray is returned. If no node is hit zero is returned for the object.

**Example**

```plaintext
TargetedNode = IrrGetCollisionNodeFromRay( RayStartVector, RayEndVector )
```

### IrrGetCollisionNodeFromScreenCoordinates

**Syntax**

```plaintext
irr_node = IrrGetCollisionNodeFromScreenCoordinates ( screen x as integer, screen y as integer )
```

**Description**

A ray is cast through the screen at the specified co-ordinates and the nearest node that is hit by the ray is returned. If no node is hit zero is returned for the object.

**Example**
**IrrGetScreenCoordinatesFrom3DPosition**

**Syntax**

IrrGetScreenCoordinatesFrom3DPosition ( screen x as integer, screen y as integer, at position as IRR_VECTOR )

**Description**

Screen co-ordinates are returned for the position of the specified 3D co-ordinates as if an object were drawn at them on the screen, this is ideal for drawing 2D bitmaps or text around or on your 3D object on the screen for example in the HUD of an aircraft. After the call Screen X and Screen Y will contain the co-ordinates.

**Example**

IrrGetScreenCoordinatesFrom3DPosition ( XPosition, YPosition, RocketVector )

**IrrGet2DPositionFromScreenCoordinates (contributed by agamemnus)**

**Syntax**

IrrGet2DPositionFromScreenCoordinates ( screenx As integer, screeny As integer, x As Single, y As Single, camera As irr_camera )

**Description**

Calculates the intersection between a ray projected through the specified screen co-ordinates and a plane at the world origin.

The Parameters X, Y and Z will receive the 2D position where the line through the screen intersects with the plane.

**Example**

IrrGet2DPositionFromScreenCoordinates ( 256, 256, x, y, OurCamera )

IrrSetNodePosition( MyCursor, XPosition, YPosition, ZPosition )
**IrrGet3DPositionFromScreenCoordinates (contributed by agamemnus)**

**Syntax**
IrrGet3DPositionFromScreenCoordinates ( screenx as integer, screeny as integer, x as single, y as single, z as single, camera as irr_camera, normalX as single = 0.0, normalY as single = 0.0, normalZ as single = 1.0, distanceFromOrigin as single = 0.0 )

**Description**
Calculates the intersection between a ray projected through the specified screen co-ordinates and a plane defined from a normal and the distance of that plane from the world origin.

The Parameters X, Y and Z will receive the 3D position where the line through the screen intersects with the plane.

**Example**

```c
IrrGet3DPositionFromScreenCoordinates ( ScreenX, ScreenY, XPosition, YPosition, ZPosition, MyCamera )
IrrSetNodePosition( MyModel, XPosition, YPosition, ZPosition )
```

---

**IrrGetChildCollisionNodeFromRay**

**Syntax**
irr_node = IrrGetChildCollisionNodeFromRay ( node as irr_node, idMask as integer, recurse as uinteger, start as IRR_VECTOR, line_end as IRR_VECTOR )

**Description**
A ray is cast through the supplied coordinates and the nearest node that is hit by the ray is returned. If no node is hit zero is returned for the object, only a subset of objects are tested, i.e. the children of the supplied node that match the supplied id. If the recurse option is enabled the entire tree of child objects connected to this node are tested.

**Example**

```c
IrrGetChildCollisionNodeFromRay ( SectorNode, 100, IRR_OFF, StartPoint, EndPoint )
```
### IrrGetChildCollisionNodeFromPoint

**Syntax**

```
irr_node = IrrGetChildCollisionNodeFromPoint ( node as irr_node, idMask as integer, recurse as uinteger, point as IRR_VECTOR )
```

**Description**

The node and its children are recursively tested and the first node that contains the matched point is returned. If no node is hit, zero is returned for the object, only a subset of objects are tested, i.e. the children of the supplied node that match the supplied id. If the recurse option is enabled the entire tree of child objects connected to this node are tested.

**Example**

```
IrrGetChildCollisionNodeFromPoint ( SectorNode, 100, IRR_ON, TestPoint )
```

### IrrGetNodeAndCollisionPointFromRay

**Syntax**

```
irr_node = IrrGetNodeAndCollisionPointFromRay ( vectorStart as IRR_VECTOR, vectorEnd as IRR_VECTOR, node as irr_node, posX as single, posY as single, posZ as single, normalX as single, normalY as single, normalZ as single, id as integer = 0, rootNode as irr_node = IRR_NO_OBJECT )
```

**Description**

A ray is cast through the specified co-ordinates and the nearest node that has a collision selector object that is hit by the ray is returned along with the coordinate of the collision and the normal of the triangle that is hit. If no node is hit, zero is returned for the object. If a node is supplied for the rootNode that tests for collision start from that node and are only tested against that node and its children.

**Example**

```
IrrGetRayFromScreenCoordinates ( screen_x, screen_y, CameraNode, StartVector, EndVector )
IrrGetNodeAndCollisionPointFromRay ( StartVector, EndVector, 
```
```plaintext
collidedNode, hitX, hitY, hitZ, normalX, normalY, normalZ, 0, myRoom 
if NOT collidedNode = IRR_NO_OBJECT then
    Print "We hit something"
end if
```

### IrrGetDistanceBetweenNodes

**Syntax**

```plaintext
distance = IrrGetDistanceBetweenNodes ( nodeA as IRR_NODE, nodeB as IRR_NODE )
```

**Description**

The distance between two nodes is measured using fast maths functions that will show inaccuracies. Useful for when it is necessary to test distances between many nodes.

**Example**

```plaintext
Dim As Single Distance = IrrGetDistanceBetweenNodes( nodeA, nodeB )
```

### IrrAreNodesIntersecting

**Syntax**

```plaintext
test = IrrAreNodesIntersecting ( nodeA as IRR_NODE, nodeB as IRR_NODE )
```

**Description**

Tests whether the bounding boxes are two nodes are intersecting. Bounding boxes are axis aligned and do not rotate when you rotate the nodes. This should be kept in mind when testing for collisions.

**Example**

```plaintext
If NOT IrrAreNodesIntersecting ( nodeA, nodeB ) = 0 Then
    Print "Collision"
End If
```

### IrrIsPointInsideNode

**Syntax**

```plaintext
```
irr_node = IrrIsPointInsideNode ( node as IRR_NODE, X as Single, Y as Single, Z as Single )

Description
Determine if the specified point is inside the bounding box of the node.

Example
If NOT IrrIsPointInsideNode ( node, X, Y, Z ) = 0 Then
    Print "Point is inside Node"
End If

IrrGetCollisionResultPosition (contributed by The Car)

Syntax
IrrGetCollisionResultPosition ( selector As irr_selector, ellipsoidPosition As IRR_VECTOR, ellipsoidRadius As IRR_VECTOR, velocity As IRR_VECTOR, gravity As IRR_VECTOR, slidingSpeed as single, outPosition As IRR_VECTOR, outHitPosition As IRR_VECTOR, outFalling As Integer )

Description
Collides a moving ellipsoid with a 3d world with gravity and returns the resulting new position of the ellipsoid, the point at which the ellipsoid collided with the surface and whether the ellipsoid is falling through the air.

This can be used for moving a character in a 3d world: The character will slide at walls and is able to walk up stairs. The method used how to calculate the collision result position is based on the paper "Improved Collision detection and Response" by Kasper Fauerby.

Example
IrrGetCollisionResultPosition ( _
    collisionGroup, _
    vectPosition, _
    vectRadius, _
    vectVelocity, _
    vectGravity, _
    0.00005, _
    vectResultPosition, _
    vectHitPosition
    areFalling )
IrrSetNodePosition( rockNode, vectPosition.X,
\[ \text{vectPosition.Y, vectPosition.Z } \]
**Cameras**

Calls for creating and controlling cameras in the scene. The camera objects are used for defining a view point and a target point which is used to render the scene.

<table>
<thead>
<tr>
<th>IrrAddFPSCamera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
</tbody>
</table>
irr_camera = IrrAddFPSCamera |
| Description     |
Adds a 'first person shooter' style camera into the scene that will be used to define the view point and target point and other attributes of the view into the 3D scene. If you haven't captured mouse and keyboard events this camera can be controlled with the cursor keys and the mouse. |
| Example         |
FPSCamera = IrrAddFPSCamera |

<table>
<thead>
<tr>
<th>IrrAddCamera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
</tbody>
</table>
irr_camera = IrrAddCamera ( camera X as single, camera Y as single, camera Z as single, target X as single, target Y as single, target Z as single ) |
| Description  |
Adds a camera to into the scene that will be used to define the view point and target point and other attributes of the view into the 3D scene. Animators and other node functions can be applied to this node. |
| Camera X, Camera Y and Camera Z define the view point of the camera. |
| Target X, Target Y and Target Z define the target of the camera, |
**IrrAddMayaCamera**

Syntax

```c
irr_camera = IrrAddMayaCamera ( parent as irr_node, rotateSpeed as single, zoomSpeed as single, moveSpeed as single )
```

Description

Adds a Maya style camera to into the scene the user can click with the left, middle and right mouse buttons to move, zoom and rotate the camera.

- **rotateSpeed** the speed at which the camera revolves
- **zoomSpeed** the speed at which the camera zooms in and out
- **moveSpeed** the speed at which the camera moves

Example

```c
CameraObject = IrrAddMayaCamera( IRR_NO_OBJECT, 100.0, 100.0, 100.0 )
```

**IrrSetCameraTarget**

Syntax

```c
IrrSetCameraTarget ( camera as irr_camera, X as single, Y as single, Z as single )
```

Description

The camera view point can be moved by simply using the `IrrSetNodePosition` function but this operation will change the point that the camera is pointing at.

Example

```c
IrrSetCameraTarget ( CameraObject, 0, 50, 0 )
```

**IrrGetCameraTarget**

Syntax

```c
IrrGetCameraTarget ( camera as irr_camera, X as single, Y as single, Z as single )
```
### IrrGetCameraTarget

**Syntax**

IrrGetCameraTarget ( CameraObject, LookAtX, LookAtY, LookAtZ )

**Description**

Get the point in space that the camera is looking at. The point is copied into the supplied X, Y and Z variables.

**Example**

IrrGetCameraTarget ( CameraObject, LookAtX, LookAtY, LookAtZ )

### IrrGetCameraUpDirection

**Syntax**

IrrGetCameraUpDirection ( camera as irr_camera, X as single, Y as single, Z as single )

**Description**

Get the up vector of a camera object into the supplied variables, this controls the upward direction of the camera and allows you to roll it for free flight action. This specifies a point in space at which the top of the camera points.

**Example**

IrrGetCameraUpDirection ( CameraObject, TopOfCamPointsAtX, TopOfCamPointsAtY, TopOfCamPointsAtZ )

### IrrSetCameraUpDirection

**Syntax**

IrrSetCameraUpDirection ( camera as irr_camera, X as single, Y as single, Z as single )

**Description**

Set the up vector of a camera object, this controls the upward direction of the camera and allows you to roll it for free flight action. This specifies a point in space at which the top of the camera points.

**Example**

IrrSetCameraUpDirection ( CameraObject, TopOfCamPointsAtX, TopOfCamPointsAtY, TopOfCamPointsAtZ )
### IrrGetCameraOrientation

**Syntax**

```plaintext
IrrGetCameraOrientation ( camera as irr_camera, X as IRR_VECTOR, Y as IRR_VECTOR, Z as IRR_VECTOR )
```

**Description**

Gets the vectors describing the camera direction useful after the camera has been revolved.

**Example**

```plaintext
IrrGetCameraOrientation ( CameraObject, VectorX, VectorY, VectorZ )
```

### IrrRevolveCamera

**Syntax**

```plaintext
IrrRevolveCamera ( camera as irr_camera, yaw as single, pitch as single, roll as single, drive as single, strafe as single, elevate as single )
```

**Description**

Revolve the camera using quaternion calculations, this will help avoid gimbal lock associated with normal Rotations and is ideal for spacecraft and aircraft.

The command takes six parameters that control yaw (turning left and right), pitch (tilting up and down), roll (rolling left and right), drive (moving forwards and backward), strafe (moving left and right) and finally elevate (moving up and down)

Many thanks to RogerBorg for this.

**Example**

```plaintext
IrrRevolveCamera ( CameraObject, CameraYaw, CameraPitch, CameraRoll, CameraDrive, CameraDrive, CameraStrafe, CameraElevate )
```

### IrrSetCameraUpAtRightAngle
### Syntax
```c
IrrSetCameraUpAtRightAngle ( camera as irr_camera )
```

### Description
Set the camera up at a right angle to the camera vector.

### Example
```c
IrrSetCameraUpAtRightAngle ( CameraObject )
```

---

**IrrSetCameraOrthogonal**

### Syntax
```c
IrrSetCameraOrthogonal ( camera as irr_camera,
distanceX as single,
distanceY as single,
distanceZ as single )
```

### Description
Set the projection of the camera to an orthagonal view, where there is no sense of perspective. The distance to the target adjusts the width and height of the camera view, essentially the smaller it is the larger the object will appear.

### Example
```c
IrrGetNodePosition( MyTarget, tarX, tarY, tarZ )
IrrGetNodePosition( MyCamera, camX, camY, camZ )
IrrSetCameraOrthogonal ( MyCamera, camX-tarX, camY-tarY, camZ-tarZ )
```

---

**IrrSetCameraClipDistance**

### Syntax
```c
IrrSetCameraClipDistance ( camera as irr_camera,
distance as single )
```

### Description
A camera clips objects in the distance that may be a part of the scene to increase rendering performance without requiring you to manage adding and deleting the objects from the view. This defines the distance beyond which no polygons will be drawn.

### Example
```c
IrrSetCameraClipDistance ( CameraObject, 12000 )
```
**IrrSetActiveCamera**

Syntax

IrrSetActiveCamera ( camera as irr_camera )

Description

When you have several camera objects in the scene you can use this call to define which of them is to be used to look through when drawing the scene.

Example

IrrSetActiveCamera( CameraObject )

---

**IrrSetCameraFOV**

Syntax

IrrSetCameraFOV ( camera as irr_camera, fov as single )

Description

Sets the field of vision of the camera a wide field of vision will give a distorted perspective, if the angle is too narrow the display will feel restricted. The value is in radians and has a default value of PI / 2.5

Example

IrrSetCameraFOV( CameraObject, PI / 2 )

---

**IrrSetCameraAspectRatio**

Syntax

IrrSetCameraAspectRatio ( camera as irr_camera, aspectRatio as single )

Description

Sets the aspect ratio of the camera in the same way you think of standard screens and widescreens. A widescreen usually has an aspect ratio of 16:9 or 16/9 = 1.78. The camera aspect ratio is set up automatically however if you are using split screen effects you may need to change the camera aspect ratio.

Example
IrrSetCameraAspectRatio( CameraObject, 1.78 )
# Lighting

Calls to create and effect lighting in the scene.

<table>
<thead>
<tr>
<th><strong>IrrAddLight</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td><code>irr_node = IrrAddLight ( x as single, y as singlez as single, red as single, green as single, blue as single, size as single )</code></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Adds a light into scene to naturally illuminate your scene.</td>
</tr>
<tr>
<td>X, Y and Z defines the coordinates of the light in the scene.</td>
</tr>
<tr>
<td>Red, Green and Blue define the intensities of the lighting for those colors. This is a fractional number ranging from 0 upwards the higher the value the brighter the light.</td>
</tr>
<tr>
<td>Size specifies the radius of effect of the light</td>
</tr>
<tr>
<td><strong>Example</strong></td>
</tr>
<tr>
<td><code>WarningLight = IrrAddLight ( 0, 100, 50, 0.5, 0.5, 0.5, 50 )</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IrrSetAmbientLight</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td><code>IrrSetAmbientLight ( Red as single, Green as single, Blue as single )</code></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Sets the ambient lighting level in the scene, ambient light casts light evenly across the entire scene and can be used to increase the overall lighting level. If should never be greater that the brightness of the darkest area of your scene, it can however reduce the number of lights you need in the scene.</td>
</tr>
</tbody>
</table>
| The Red, Green and Blue components of this lighting is supplied as integers in
the range or 0 to 255

Example

```
IrrSetAmbientLight( 72, 64, 64 )
```

### IrrSetLightAmbientColor

**Syntax**

```
IrrSetLightAmbientColor( Light as irr_node, Red as single, Green as single, Blue as single )
```

**Description**

Ambient color emitted by the light, ambient light casts light evenly across the entire scene and can be used to increase the overall lighting level. If should never be greater that the brightness of the darkest area of your scene, it can however reduce the number of lights you need in the scene.

The Red, Green and Blue components of this lighting is supplied as singles specifying the brightness in each color channel.

**Example**

```
IrrSetLightAmbientColor( SceneLight, 1.0, 0.1, 0.7 )
```

### IrrSetLightAttenuation

**Syntax**

```
IrrSetLightAttenuation( Light as irr_node, Red as single, Green as single, Blue as single )
```

**Description**

Changes the light strength fading over distance. Good values for distance effects use (1.0, 0.0, 0.0) and simply add small values to the second and third element.

**Example**

```
IrrSetLightAttenuation( SceneLight, 1.0, 0.08, 0.07 )
```
### IrrSetLightCastShadows

**Syntax**

```c
IrrSetLightCastShadows( Light as irr_node, cast_shadows as uinteger )
```

**Description**

Specifies whether the light casts shadows in the scene or not. Shadowing must be enabled in the IrrStart call and also on the nodes in the scene.

**Example**

```c
IrrSetLightCastShadows( SceneLight, IRR_ON )
```

### IrrSetLightDiffuseColor

**Syntax**

```c
IrrSetLightDiffuseColor( Light as irr_node, Red as single, Green as single, Blue as single )
```

**Description**

IrrSetLightDiffuseColor

The Red, Green and Blue components of this lighting is supplied as singles specifying the brightness in each color channel.

**Example**

```c
IrrSetLightDiffuseColor( SceneLight, 1.0, 1.0, 0.8 )
```

### IrrSetLightFalloff

**Syntax**

```c
IrrSetLightFalloff( Light as irr_node, Falloff as single )
```

**Description**

The light strength's decrease between Outer and Inner cone.

**Example**

```c
IrrSetLightFalloff( SceneLight, 0.8 )
```
**IrrSetLightInnerCone**

Syntax

IrrSetLightInnerCone( Light as irr_node, InnerCone as single )

Description
The angle of the spot's inner cone. Ignored for other lights.

Example

IrrSetLightInnerCone( SceneLight, 0.4 )

---

**IrrSetLightOuterCone**

Syntax

IrrSetLightOuterCone( Light as irr_node, OuterCone as single )

Description
The angle of the spot's outer cone. Ignored for other lights.

Example

IrrSetLightOuterCone( SceneLight, 0.9 )

---

**IrrSetLightRadius**

Syntax

IrrSetLightRadius( Light as irr_node, Radius as single )

Description
Radius of light. Everything within this radius be be lighted. If some artefacts can be seen when the radius is changed in this instance simply make the radius a little large.

Example

IrrSetLightRadius( SceneLight, 50.2 )

---

**IrrSetLightSpecularColor**

Syntax

IrrSetLightSpecularColor( Light as irr_node, Red as single, Green as single,
Blue as single

Description
Sets the ambient lighting level in the scene, ambient light casts light evenly across the entire scene and can be used to increase the overall lighting level. If should never be greater that the brightness of the darkest area of your scene, it can however reduce the number of lights you need in the scene.

The Red, Green and Blue components of this lighting is supplied as singles specifying the brightness in each color channel

Example

| IrrSetLightSpecularColor( SceneLight, 1.0, 1.0, 1.0 ) |

### IrrSetLightType

**Syntax**

IrrSetLightType( Light as irr_node, Light_type as E_LIGHT_TYPE )

**Description**

The type of the light. All lights default to a point light but can be changed with this setting to one of the following values:

- ELT_POINT
- ELT_SPOT
- ELT_DIRECTIONAL

**Example**

| IrrSetLightType( SceneLight, ELT_SPOT ) |
**Terrain**

Calls to create and alter the properties of terrain meshes, special nodes that are used to create large expansive landscapes.

### IrrAddTerrain

**Syntax**

```c
irr_terrain = IrrAddTerrain ( path as zstring ptr, xPosition as single = 0.0, yPosition as single = 0.0, zPosition as single = 0.0, xRotation as single = 0.0, yRotation as single = 0.0, zRotation as single = 0.0, xScale as single = 1.0, yScale as single = 1.0, zScale as single = 1.0, vertexAlpha as integer = 255, vertexRed as integer = 255, vertexGreen as integer = 255, vertexBlue as integer = 255, smoothing as integer = 0, maxLOD as integer = 5, patchSize as IRR_TERRAIN_PATCH_SIZE = ETPS_17 )
```

**Description**

Creates a terrain object from a gray scale bitmap where bright pixels are high points on the terrain and black pixels are low points. You will inevitably have to rescale the terrain during the call or after it is created. The Terrain object is a special dynamic mesh whose resolution is reduced in the distance to reduce the number of triangles it consumes.

Path is the filename of a gray scale image used to define the contours of the surface.

xPosition, yPosition and zPosition define the position of the terrain.

xRotation, yRotation and zRotation define the rotation of the terrain.

xScale, xScale and xScale define the scale of the terrain.

vertexAlpha, vertexRed, vertexGreen, vertexBlue, define the vertex color of all points in the terrain.

smoothing allows you to define whether the contours of the surface of the terrain are smoothed over.

maxLOD and patchsize control the properties of the level of detail calculations applied to the terrain, it is recommended that these are left at default values.
Example
TerrainNode = IrrAddTerrain( "CanyonsHeightField.bmp" )

**IrrAddTerrainTile**

**Syntax**

```plaintext
irr_terrain = IrrAddTerrainTile ( image as irr_image, tileSize as integer = 256, 
dataX as integer = 0, dataY as integer = 0, xPosition as single = 0.0, yPosition 
as single = 0.0, zPosition as single = 0.0, xRotation as single = 0.0, yRotation 
as single = 0.0, zRotation as single = 0.0, xScale as single = 1.0, yScale as 
single = 1.0, zScale as single = 1.0, smoothing as integer = 1, maxLOD as 
integer = 5, patchSize as IRR_TERRAIN_PATCH_SIZE = ETPS_17 )
```

**Description**

Creates a tilable terrain object from a gray scale bitmap where bright pixels are high points on the terrain and black pixels are low points. You will inevitably have to rescale the terrain during the call or after it is created. The Terrain object is a special dynamic mesh whose resolution is reduced in the distance to reduce the number of triangles it consumes.

Unlike the origonal terrain object the tileable terrain object can be attached to other terrain tile objects without being affected by cracks between tiles caused by the level of detail mechanism. When working with tile terrains it should be noted that the terrain is internally divided up into patches that are patchSize - 1 and there is always one invisible row of patches at the top and left of the terrain. Essentially this means that if your tileSize is 128 x 128 the visible size of your terrain will be 112 x 112 (with a patchSize of ETPS_17)

**Note:** Tiled Terrain object can be automatically control with the Zone Manager objects please refer to them for further details.

Image is an image file loaded with IrrGetImage and containing a gray scale image used to define the contours of the surface.
TileSize defines the size of the terrain independantly of the size of the image used to create it
xPosition, yPosition and zPosition define the position of the terrain
xRotation, yRotation and zRotation define the rotation of the terrain
xScale, xScale and xScale define the scale of the terrain
smoothing allows you to define whether the contours of the surface of the
terrain are smoothed over.
maxLOD and patchsize control the properties of the level of detail calculations
applied to the terrain, it is recommended that these are left at default values.

Example

TerrainNode = IrrAddTerrainTile( EasterIslandImage, 128 )

**IrrAddSphericalTerrain**

Syntax

irr_terrain = IrrAddSphericalTerrain( topPath as zstring ptr, frontPath as
zstring ptr, backPath as zstring ptr, leftPath as zstring ptr, rightPath as zstring
ptr, bottomPath as zstring ptr, xPosition as single = 0.0, yPosition as single =
0.0, zPosition as single = 0.0, xRotation as single = 0.0, yRotation as single =
0.0, zRotation as single = 0.0, xScale as single = 1.0, yScale as single = 1.0,
zScale as single = 1.0, vertexAlpha as integer = 255, vertexRed as integer =
255, vertexGreen as integer = 255, vertexBlue as integer = 255, smoothing as
integer = 0, spherical as integer = 0, maxLOD as integer = 5, patchSize as
IRR_TERRAIN_PATCH_SIZE = ETPS_17 )

Description

Creates a spherical terrain that represents a planetary body. When using this
terrain it is better to think of it as a cube rather than a sphere, in fact it is a cube
that is distorted so that its surface becomes spherical, like a cube it has a top,
bottom, left, right, front and back and co-ordinates are thought of as being at
position X,Y on cube face N. In someways this makes working with placing
things on the object simpler as you can think of it as six flat surfaces.

The first six paths are the path of six gray scale bitmaps where bright pixels are
high points on the terrain and black pixels are low points.
The position, rotation and scale of the terrain are specified with the next series
of parameters.
Four parameters are used to set the vertex color of all the verticies in the
terrain.
Smoothing is used to smooth out the contours of the hills in the terrain.
maxLOD and patchSize allow you to adjust the level of detail within the terrain.
although it is usually best to leave these to default values.

When creating heightmaps for the faces of the terrain you will need to ensure that the height of pixels at the edge of adjoining sides of the terrain are the same otherwise large visible cracks will appear at the edges of the faces, the easiest way to do this is to create terrain texture and then copy and/or rotate it onto its adjacent face. You can get some surprisingly effective planets and asteroids with textures as small as 32x32 but the object also runs well with a terrain size at the maximum 256 x 256.

Example

```python
Terrain = IrrAddSphericalTerrain( _
    "moonbase_top.bmp",_
    "moonbase_front.bmp",_
    "moonbase_back.bmp",_
    "moonbase_left.bmp",_
```
IrrGetTerrainHeight

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>single = IrrGetTerrainHeight ( terrain as irr_terrain, X as single, Y as single )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the height of a point on a terrain. This can be a particularly fast and accurate way to move an object over a terrain.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y = IrrGetTerrainHeight ( TerrainNode, X, Z )</td>
</tr>
</tbody>
</table>

IrrScaleTexture

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>IrrScaleTexture ( terrain as irr_terrain, X as single, Y as single )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As a terrain object is a particularly huge mesh when textured are applied to it they look extremely pixelated. To get over this effect a terrain object can have two materials applied to it, one to give general surface color and a second that is copied across the surface like tiles to give a rough detailed texture. This call specifies the scaling of this detail texture.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>IrrScaleTexture ( TerrainNode, 20, 20 )</td>
</tr>
</tbody>
</table>

IrrGetTerrainTileHeight

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>single = IrrGetTerrainTileHeight ( terrain as irr_terrain, X as single, Y as single )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Get the height of a point on a terrain tile. This can be a particularly fast and accurate way to move an object over a terrain.

Example

\[ Y = \text{IrrGetTerrainTileHeight} \left( \text{TerrainNode, X, Z} \right) \]

### IrrScaleTileTexture

**Syntax**

\[ \text{IrrScaleTileTexture} \left( \text{terrain as irr\_terrain, X as single, Y as single} \right) \]

**Description**

As a tile terrain object is a particularly huge mesh when textured are applied to it they look extremely pixelated. To get over this effect a terrain object can have two materials applied to it, one to give general surface color and a second that is copied across the surface like tiles to give a rough detailed texture. This call specifies the scaling of this detail texture.

Example

\[ \text{IrrScaleTileTexture} \left( \text{TerrainNode, 20, 20} \right) \]

### IrrAttachTile

**Syntax**

\[ \text{IrrAttachTile} \left( \text{terrain as irr\_terrain, neighbouring\_terrain as irr\_terrain, edge as integer} \right) \]

**Description**

Set the adjacent tile to this tile node. To avoid cracks appearing between tiles, tiles need to know which tiles are their neighbours and which edges they are attached too.

Example

\[ \text{IrrAttachTile( TerrainNorth, TerrainSouth, TOP\_EDGE )} \]
\[ \text{IrrAttachTile( TerrainSouth, TerrainNorth, BOTTOM\_EDGE )} \]

### IrrSetTileStructure
**IrrSetTileStructure**

**Syntax**

```pascal
IrrSetTileStructure ( terrain as irr_terrain, image as irr_image, x as integer, y as integer )
```

**Description**

Loads the tile structure from the supplied image file. Unlike the image in the original call to create a terrain tile this image has a different structure. The image should be in RGBA format, the alpha value is used to set the height of the terrain and the RGB values are used to set the color of the vertices. This can either be for loading precalculated lighting into the scene or it can be used with the new IRR_EMT_FOUR_DETAIL_MAP material type to define the weight of each of the greyscale detail maps in the RGB channels of the detail map. The x and y values can be used to load the structure from a specific point on the bitmap.

**Example**

```pascal
IrrSetTileStructure( TerrainCove, CoveStructure, 0, 0 )
```

**IrrSetTileColor**

**Syntax**

```pascal
IrrSetTileColor( terrain as irr_terrain, image as irr_image, x as integer, y as integer )
```

**Description**

Loads the tile vertex colors from the supplied image file. The RGB values are used to set the color of the vertices. This can either be for loading precalculated lighting into the scene or it can be used with the new IRR_EMT_FOUR_DETAIL_MAP material type to define the weight of each of the greyscale detail maps in the RGB channels of the detail map. The x and y values can be used to load the structure from a specific point on the bitmap.

**Example**

```pascal
IrrSetTileColor( TerrainCove, CoveStructure, 0, 0 )
```
IrrScaleSphericalTexture ( terrain as irr_terrain, X as single, Y as single )

Description
As the surfaces of a spherical terrain object are a particularly huge mesh when textures are applied to them they look extremely pixelated. To get over this effect a spherical terrain object can have two materials applied to it, one to give general surface color and a second that is copied across the surface like tiles to give a rough detailed texture. This call specifies the scaling of this detail texture.

Example
IrrScaleSphericalTexture ( SphericalTerrainNode, 20, 20 )

IrrSetSphericalTerrainTexture

Syntax
IrrSetSphericalTerrainTexture ( terrain as irr_terrain, topTexture as irr_texture, frontTexture as irr_texture, backTexture as irr_texture, leftTexture as irr_texture, rightTexture as irr_texture, bottomTexture as irr_texture, materialIndex as integer )

Description
Apply six textures to the surface of a spherical terrain. By using the material index you can set the color or the detail maps

Example
IrrSetSphericalTerrainTexture ( TerrainNode, _"moobbase_col_top.bmp", _"moobbase_col_front.bmp", _"moobbase_col_back.bmp", _"moobbase_col_left.bmp", _"moobbase_col_right.bmp", _"moobbase_col_bottom.bmp", _0 )

IrrLoadSphericalTerrainVertexColor

Syntax
IrrLoadSphericalTerrainVertexColor ( terrain as irr_terrain, topMap as irr_image, frontMap as irr_image, backMap as irr_image, leftMap as irr_image, rightMap as irr_image,
rightMap as irr_image, bottomMap as irr_image )

Description
Apply six images to the vertex colors of the faces, this is useful for setting the vertices so that they can be used with simple terrain spattering described in the section on tiled terrains above.

Example

IrrLoadSphericalTerrainVertexColor ( TerrainNode, _
  "moobbase_vert_top.bmp", _
  "moobbase_vert_front.bmp", _
  "moobbase_vert_back.bmp", _
  "moobbase_vert_left.bmp", _
  "moobbase_vert_right.bmp", _
  "moobbase_vert_bottom.bmp" )

---

**IrrGetSphericalTerrainSurfacePosition**

Syntax
IrrGetSphericalTerrainSurfacePosition ( terrain as irr_terrain, face as integer, logicalX as single, logicalZ as single, X as single, Y as single, Z as single )

Description
Get the surface position of a logical point on the terrain. You supply a face number and a logical X, Y position on that face and this call will return the height of that point on the terrain sphere inside the X, Y, Z parameters.

Note: By subtracting the center of the sphere from this co-ordinate and converting this vector to angles you can find the upward direction of the point on the surface.

Example

IrrGetSphericalTerrainSurfacePosition ( TerrainNode, IRR_TOP_FACE, buggyX, buggyZ, X, Y, Z )

---

**IrrGetSphericalTerrainSurfacePositionAndAngle**

Syntax
IrrGetSphericalTerrainSurfacePosition ( terrain as irr_terrain, face as integer, logicalX as single, logicalZ as single, X as single, Y as single, Z as single,
RotationX as single, RotationY as single, RotationZ as single

Description
Get the surface position and angle of a logical point on the terrain. This is not the normal of the surface but essentially the angles to the gravitational center.

Example
IrrGetSphericalTerrainSurfacePositionAndAngle ( Terrain, F, I, J, PX, PY, PZ, RX, RY, RZ )

IrrGetSphericalTerrainLogicalSurfacePosition

Syntax
IrrGetSphericalTerrainSurfacePosition ( terrain as irr_terrain, face as integer, logicalX as single, logicalZ as single, X as single, Y as single, Z as single )

Description
Convert a co-ordinate into a logical Spherical terrain position. Thanks for the example from "David" posted on Infinity-Universe forum

Please note that this calculation is not 100% accurate, it is advised that the translation is done at altitude and the difference either ignored or blended as the observer descends.

Note: The height above the surface can be calculated simply by calculating the length of the center of the planet to the surface and then the center of the planet to the space coordinate and subtracting the two

Note: The momentum could be calculated by converting two samples and then measuring the difference in height and X and Z on the face

Example
IrrGetSphericalTerrainLogicalSurfacePosition ( Terrain, X, Y, Z, face, LX, LZ )
Particles

Calls to control the appearance and follow of particles in particle systems.

IrrSetMinParticleSize

Syntax
IrrSetMinParticleSize ( particle emitter as irr_particle_emitter, X as single, Y as single )

Description
Particles in a particle system are simple 2 dimensional billboard like objects, this sets the size of these particles, larger particles can be effective and use less resources however they can look blocky if taken too far.

Example
IrrSetMinParticleSize ( SmokeEmitter, 5, 5 )

IrrSetMaxParticleSize

Syntax
IrrSetMaxParticleSize ( particle emitter as irr_particle_emitter, X as single, Y as single )

Description
Particles in a particle system are simple 2 dimensional billboard like objects, this sets the size of these particles, larger particles can be effective and use less resources however they can look blocky if taken too far.

Example
IrrSetMaxParticleSize ( SmokeEmitter, 15, 15 )

IrrAddParticleEmitter
Syntax
irr_emitter = IrrAddParticleEmitter ( particle system as irr_particle_system, settings as IRR_PARTICLE_EMITTER )

Description
Adds a particle emitter to the particle system, this creates particles and controls how they move and when they are to be removed. It requires a very large number of parameters to define this flexible effect and as such these parameters are stores in a special IRR_PARTICLE_EMITTER structure.

Example
MyEmitter = IrrAddParticleEmitter( SmokeParticles, SmokeEmitter )

IrrAddAnimatedMeshSceneNodeEmitter
Syntax
irr_emitter = IrrAddAnimatedMeshSceneNodeEmitter( particle_system as irr_particle_system, node as irr_node, use_normal_direction as uinteger, normal_direction_modifier as single, emit_from_every_vertex as integer, settings as IRR_PARTICLE_EMITTER )

Description
Creates a particle emitter for an animated mesh scene node

Parameters:
 node - Pointer to the animated mesh scene node to emit particles from
 useNormalDirection - If true, the direction of each particle created will be the normal of the vertex that it's emitting from. The normal is divided by the normalDirectionModifier parameter, which defaults to 100.0f.
 normalDirectionModifier - If the emitter is using the normal direction then the normal of the vertex that is being emitted from is divided by this number.
 everyMeshVertex - If true, the emitter will emit between min/max particles every second, for every vertex in the mesh, if false, it will emit between min/max particles from random vertices in the mesh.

A large number of additional parameters are also required to define this flexible effect and as such these parameters are stores in a special IRR_PARTICLE_EMITTER structure. The box size properties of this structure
## are unused in this call

**Example**

```plaintext
MyEmitter = IrrAddAnimatedMeshSceneNodeEmitter ( SmokeParticles, SceneNode, 1, 0.25, 0, SmokeEmitter )
```

### IrrAddFadeOutParticleAffector

**Syntax**

```plaintext
irr_affector = IrrAddFadeOutParticleAffector ( particle_system as irr_particle_system )
```

**Description**

Adds a fade out affector to the particle system, this fades the particles out as they come to the end of their lifespan and stops them 'popping' out of existance. This creates a convincing effect for fire and smoke in particular.

**Example**

```plaintext
MyAffector = IrrAddFadeOutParticleAffector( SmokeParticles )
```

### IrrAddGravityParticleAffector

**Syntax**

```plaintext
irr_affector = IrrAddGravityParticleAffector ( particle_system as irr_particle_system, x as single, y as single, z as single )
```

**Description**

Adds a gravity affector to the particle system, this gradually pulls the particles in the direction of the effect, although it is called a gravity effector it can be used to make a wind effect and have the particles drift off to the side.

X, Y and Z define the force that is applied to the particles over time.

**Example**

```plaintext
MyAffector = IrrAddGravityParticleAffector( SmokeParticles, -0.1, 0, 0 )
```

### IrrAddParticleAttractionAffector
Syntax
irr_affecter = IrrAddParticleAttractionAffector( particle_system as
irr_particle_system, x as Single, y as Single, z as Single, speed as Single = 1.0,
attract as uinteger = 1, affectX as uinteger = 1, affectY as uinteger = 1, affectZ
as uinteger = 1 )

Description
Creates a point attraction affecter. This affecter modifies the positions of the
particles and attracts them to a specified point at a specified speed per second.

Parameters:
x,y,z - Point to attract particles to.
speed - Speed in units per second, to attract to the specified point.
attract - Whether the particles attract or detract from this point use the constants
IRR_ATTRACT or IRR_REPEL (defaults to IRR_ATTRACT)
affectX - Whether or not this will affect the X position of the particle, use 1 to
effect the position and 0 to leave it unaffected ( defaults to true ).
affectY - Whether or not this will affect the Y position of the particle, use 1 to
effect the position and 0 to leave it unaffected ( defaults to true ).
affectZ - Whether or not this will affect the Z position of the particle, use 1 to
effect the position and 0 to leave it unaffected ( defaults to true ).

Example
MyAffecter = IrrAddParticleAttractionAffector( SmokeParticles,
0.0, 10.0, 0.0, 20.0, IRR_ATTRACT, 1, 1, 1 )

IrrCreateRotationAffector

Syntax
irr_affecter = IrrCreateRotationAffector ( particle_system as
irr_particle_system, Speed_X as Single, Speed_Y as Single, Speed_Z as Single,
pivot_X as Single, pivot_Y as Single, pivot_Z as Single )

Description
Creates a rotation affecter. This affecter modifies the positions of the particles
and attracts them to a specified point at a specified speed per second.

Parameters:
speed x,y,z - Rotation in degrees per second
pivot x,y,z - Point to rotate the particles around

Example
MyAffector = IrrCreateRotationAffector( SmokeParticles, -120.0, 0.0, 0.0, 0.0, 0.0, 0.0 )

IrrAddStopParticleAffector

Syntax
irr_affecter = IrrAddStopParticleAffector ( particle_system as irr_particle_system, time as integer, emitter as irr_emitter )

Description
The stop particle affecter waits for the specified period of time to elapse and then stops the specified emitter emitting particles by setting its minimum and maximum particle emission rate to zero. The emitter can easily be started up again by changing its emission rate.

Parameters:
Time - The number of milliseconds to elapse before the particles are stopped
Emitter - The particle generating object to stop

Example
MyAffector = IrrAddStopParticleAffector( SmokeSystem, 1000, smoke_emitter )

IrrAddParticlePushAffector

Syntax
irr_affecter = IrrAddParticlePushAffector ( particle_system as irr_particle_system, x as single, y as single, z as single, speedX as single, speedY as single, speedZ as single, far as single, near as single, column as single, distant as integer )

Description
Creates a point push affector. This affector modifies the positions of the particles and pushes them toward or away from a specified point at a specified speed per second. The strength of this effect is adjusted by a near and a far distance. Beyond the far distance the particle is not effected at all and the closer
you get to the center of the effect the stronger the force is.

If a near distance is defined (a value greater than 0.0) the effect is somewhat different, particles closer to the center than the near distance are not effected at all, and the strongest point of the effect is always halfway between the near and far limits, for example if your near distance was 25.0 and your far distance was 75.0 the strongest force would be applied to particles at a distance of 50.0

If a column width is defined the effect will only take place in a vertical column that is that wide, this is useful for fountains of water where as the water spreads out of the column a gravity affector can take over.

By adjusting these parameters and the strength you can create columns, spheres, shells and rings of effect that can, in combination, push particles in complex motions

Parameters:
- x, y, z - Point to attract particles to or repel particles away from
- speedX, speedY, speedZ - A vector describing the strength of the effect
- Far - Furthest distance of effect
- Near - Closest distance of effect
- Column - The width of a vertical column in which the push affector has influence, somewhat like the column of water in a fountain
- Distant - Use IRR_ON to apply the same force in the same direction to all particles and use IRR_OFF to apply a force that radiates away from the center of the effect

Example

MyAffector = IrrAddParticlePushAffector ( ColumnOfSmoke, 0, 0, 0, 0, 100, 0, 100, 0.0, 0.0, IRR_OFF )

**IrrAddColorMorphAffector**

**Syntax**

```
irr_affector = IrrAddColorMorphAffector ( particle_system as irr_particle_system, numberOfParticles as uinteger, particlecolors as uinteger ptr, particletimes as uinteger ptr, smooth as uinteger )
```
Description
This clever effect by Dark Kilauea that allows you to provide an array of colors and an optional array of times that effect the color of the particle over its lifetime, the particle could start off bright orange and fade away into grey and then black for example.

Parameters:
numEntries - the number of entries in the supplied table
colors - the table of colors
time - the table of times at which each color becomes active
smooth - Use IRR_ON to smoothly blend between colors and use IRR_OFF to sharply switch between colors

Example
DIM colors(0 to 4) as integer = {
    IrrMakeARGB(0,255,255,128), _
    ' yellow white
    IrrMakeARGB(0,255,128,0), _
    ' yellow
    IrrMakeARGB(0,128,64,0), _
    ' orange
    IrrMakeARGB(0,0,0,128), _
    ' slight blue
    IrrMakeARGB(255,0,0,0) _
    ' black and faded
} '  
DIM times(0 to 4) as integer = {500, 800, 1250, 1500, 2000 } '  
MyAffector = IrrAddColorMorphAffector( Fire.particles, 5, @colors(0), @times(0), IRR_ON )

### IrrAddSplineAffecter

**Syntax**
```
irr_affecter = IrrAddSplineAffecter ( particle_system as
irr_particle_system, VertexCount as uinteger, verticies as IRR_VERT ptr, speed
as single, tightness as single, attraction as single, deleteAtEnd as uinteger )
```

**Description**
This clever effect by Dark Kilauea that allows you to create an affecter that moves the particles along the path of a spline for very controled and complex particle motion.
Parameters:
VertexCount - Is the number of points in your spline
Verticies - Is an array of IRR_VERT objects defining the X,Y and Z position of points
Speed - is the speed with which particles are moved along the spline
tightness - is the tightness of the curve of the spline
attraction - is how closely the particles are attracted to the curve of the spline
deleteAtEnd - Use IRR_ON to delete the particles when they reach the end of the spline and use IRR_OFF to allow the particles to be deleted naturally.

Example

\[
\begin{align*}
\text{DIM splineverticies(0 to 3) as IRR_VERT} \\
&\text{splineverticies(0).x = 0.0 : splineverticies(0).y = 0.0 :} \\
&\text{splineverticies(0).z = 0.0} \\
&\text{splineverticies(1).x = 0.0 : splineverticies(1).y = 20.0 :} \\
&\text{splineverticies(1).z = 25.0} \\
&\text{splineverticies(2).x = 0.0 : splineverticies(2).y = 40.0 :} \\
&\text{splineverticies(2).z = -25.0} \\
&\text{splineverticies(3).x = 0.0 : splineverticies(3).y = 60.0 :} \\
&\text{splineverticies(3).z = 0.0} \\
\text{IrrAddSplineAffecttor ( NeonLight.particles, 4,} \\
&@\text{splineverticies(0), 2.0, 1.0, 5.0, IRR_ON })
\end{align*}
\]

\textbf{IrrRemoveAffecttors}

\textbf{Syntax}
\text{IrrRemoveAffecttors ( particle system as irr\_particle\_system )}

\textbf{Description}
Removes all affectors from a particle system, you might use this if you wanted to change the direction or strength of the wind for example.

\textbf{Example}
\text{IrrRemoveAffecttors( SmokeParticles )}

\textbf{IrrSetParticleEmitterDirection}

\textbf{Syntax}
\text{IrrSetParticleEmitterDirection( particle\_emitter as irr\_emitter, x as single, y as
single, z as single)

Description
Set direction the emitter emits particles.

Example
IrrSetParticleEmitterDirection( MyEmitter, 0.0, 0.4, 0.0 )

**IrrSetParticleEmitterMinParticlesPerSecond**

Syntax
IrrSetParticleEmitterMinParticlesPerSecond( particle_emitter as irr_emitter, min_particles_per_second as integer )

Description
Set minimum number of particles the emitter emits per second.

Example
IrrSetParticleEmitterMinParticlesPerSecond( MyEmitter, 32 )

**IrrSetParticleEmitterMaxParticlesPerSecond**

Syntax
IrrSetParticleEmitterMaxParticlesPerSecond( particle_emitter as irr_emitter, max_particles_per_second as integer )

Description
Set maximum number of particles the emitter emits per second.

Example
IrrSetParticleEmitterMaxParticlesPerSecond( MyEmitter, 100 )

**IrrSetParticleEmitterMinStartColor**

Syntax
IrrSetParticleEmitterMinStartColor( particle_emitter as irr_emitter, Red as integer, Green as integer, Blue as integer )
<table>
<thead>
<tr>
<th>Description</th>
<th>Set minimum starting color for particles.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>IrrSetParticleEmitterMinStartColor( MyEmitter, 255, 192, 128 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IrrSetParticleEmitterMaxStartColor</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>IrrSetParticleEmitterMaxStartColor( particle_emitter as irr_emitter, Red as uinteger, Green as uinteger, Blue as uinteger )</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Set maximum starting color for particles.</td>
</tr>
<tr>
<td>Example</td>
</tr>
<tr>
<td>IrrSetParticleEmitterMaxStartColor( MyEmitter, 255, 192, 128 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IrrSetParticleAffectorEnable</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>IrrSetParticleAffectorEnable( particle_affector as irr_affector, Enable as uinteger )</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Enable or disable an affector. Setting the value to 1 enables the affector, setting it to 0 disables it. IRR_ON and IRR_OFF can be used also.</td>
</tr>
<tr>
<td>Example</td>
</tr>
<tr>
<td>IrrSetParticleAffectorEnable( MyAffector, IRR_OFF )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IrrSetFadeOutParticleAffectorTime</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>IrrSetFadeOutParticleAffectorTime( particle_affector as irr_affector, FadeFactor as float )</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Alter the fadeout affector changing the fade out time.

Example

IrrSetFadeOutParticleAffectorTime( MyAffector, 2000.0 )

<table>
<thead>
<tr>
<th><strong>IrrSetFadeOutParticleAffectorTargetColor</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>IrrSetFadeOutParticleAffectorTargetColor( particle_affector as irr_affector, Red as uinteger, Green as uinteger, Blue as uinteger )</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Alter the fadeout affector changing the target color to the affector fades to over time.</td>
</tr>
<tr>
<td>Example</td>
</tr>
<tr>
<td>IrrSetFadeOutParticleAffectorTargetColor( MyAffector, 16, 8, 0 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IrrSetGravityParticleAffectorDirection</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>IrrSetGravityParticleAffectorDirection( particle_affector as irr_affector, x as single, y as single, z as single)</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Alter the direction and force of gravity for a gravity affector.</td>
</tr>
<tr>
<td>Example</td>
</tr>
<tr>
<td>IrrSetGravityParticleAffectorDirection( MyAffector, 0.2, 0.1, 0.0 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>IrrSetGravityParticleAffectorTimeForceLost</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>IrrSetGravityParticleAffectorTimeForceLost( particle_affector as irr_affector, time_force_lost as single )</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Set the time in milliseconds when the gravity force is totally lost and the particle does not move any more.

Example
IrrSetGravityParticleAffectorTimeForceLost( MyAffector, 800.0 )

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>IrrSetGravityParticleAffectorTimeForceLost( particle_affector as irr_affector, affect as float )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set whether or not an attraction affector will affect particles in the X direction. Setting the value to 1 enables the effect, setting it to 0 disables it. IRR_ON and IRR_OFF can be used also.</td>
</tr>
</tbody>
</table>

Example
IrrSetParticleAttractionAffectorAffectX( MyAffector, IRR_ON )

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>IrrSetParticleAttractionAffectorAffectX( particle_affector as irr_affector, affect as integer )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set whether or not an attraction affector will affect particles in the Y direction. Setting the value to 1 enables the effect, setting it to 0 disables it. IRR_ON and IRR_OFF can be used also.</td>
</tr>
</tbody>
</table>

Example
IrrSetParticleAttractionAffectorAffectY( MyAffector, IRR_ON )

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>IrrSetParticleAttractionAffectorAffectY( particle_affector as irr_affector, affect as integer )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set whether or not an attraction affector will affect particles in the Z direction. Setting the value to 1 enables the effect, setting it to 0 disables it. IRR_ON and IRR_OFF can be used also.</td>
</tr>
</tbody>
</table>

Example
IrrSetParticleAttractionAffectorAffectZ( MyAffector, IRR_ON )

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>IrrSetParticleAttractionAffectorAffectZ( particle_affector as irr_affector, affect as integer )</td>
</tr>
</tbody>
</table>
as uinteger )

Description
Set whether or not an attraction affector will affect particles in the Z direction. Setting the value to 1 enables the effect, setting it to 0 disables it. IRR_ON and IRR_OFF can be used also.

Example
IrrSetParticleAttractionAffectorAffectZ( MyAffector, IRR_ON )

**IrrSetParticleAttractionAffectorAttract**

Syntax
IrrSetParticleAttractionAffectorAttract( particle_affecter as irr_affecter, affect as uinteger )

Description
Set whether or not the particles are attracted or repelled from an attractor effector. Use the values IRR_ATTRACT and IRR_REPEL for convenience.

Example
IrrSetParticleAttractionAffectorAttract( MyAffector, IRR_ATTRACT )

**IrrSetParticleAttractionAffectorPoint**

Syntax
IrrSetParticleAttractionAffectorPoint( particle_affecter as irr_affecter, x as single, y as single, z as single )

Description
Set the point that particles will attract to when affected by this attractor affector.

Example
IrrSetParticleAttractionAffectorPoint( MyAffector, IRR_ATTRACT )

**IrrSetRotationAffectorPivotPoint**

Syntax

IrrSetRotationAffectorPivotPoint( particle_affector as irr_affector, x as single, y as single, z as single )

Description
Set the point that particles will rotate about when affected by this rotation affector.

Example
IrrSetRotationAffectorPivotPoint( MyAffector, IRR_ATTRACTION )

IrrSetFurthestDistanceOfEffect

Syntax
IrrSetFurthestDistanceOfEffect( particle_affector as irr_affector, newDistance as single )

Description
Set the furthest distance of effect on particles affected by the push affector.

Example
IrrSetFurthestDistanceOfEffect( MyAffector, 100.0 )

IrrSetNearestDistanceOfEffect

Syntax
IrrSetNearestDistanceOfEffect( particle_affector as irr_affector, newDistance as single )

Description
Set the nearest distance of effect on particles affected by the push affector.

Example
IrrSetNearestDistanceOfEffect( MyAffector, 10.0 )

IrrSetColumnDistanceOfEffect

Syntax
IrrSetColumnDistanceOfEffect( particle_affector as irr_affector, newDistance as single )
as single )

Description
Set the column distance of effect on particles affected by the push affector.

Example
IrrSetColumnDistanceOfEffect( MyAffector, 20.0 )

---

**IrrSetCenterOfEffect**

Syntax
IrrSetCenterOfEffect( particle_affector as irr_affector, x as single, y as single, z as single )

Description
Set the center of the effect of particles affected by the push affector.

Example
IrrSetCenterOfEffect( MyAffector, 0.0, PushHeight, 0.0 )

---

**IrrSetStrengthOfEffect**

Syntax
IrrSetStrengthOfEffect( particle_affector as irr_affector, x as single, y as single, z as single )

Description
Set the strength of the effect of particles affected by the push affector.

Example
IrrSetStrengthOfEffect( MyAffector, PipeVelocity, 0.0, 0.0 )
Irrlicht Graphical User Interface

Calls to add graphical user interface objects to the screen that can be drawn with a single call. At the moment this section is awaiting further development in the wrapper.

**IrrGUIClear**

Syntax
IrrGUIClear()

Description
Clears all GUI objects from the display.

Example
IrrGUIClear()

**IrrGUIRemove**

Syntax
IrrGUIRemove ( object as IRR_GUI_OBJECT )

Description
Removes the specified GUI object from the display.

Example
IrrGUIRemove( myButton )

**IrrGUIGetText**

Syntax
IrrGUIGetText ( object as IRR_GUI_OBJECT )

Description
Gets the text associated with a GUI object.
### Example
```
DIM myString as wstring = IrrGUIGetText( myEditBox )
```

### IrrGUISetText
**Syntax**
IrrGUISetText ( object as IRR_GUI_OBJECT, text as wstring )

**Description**
Sets the text of a GUI object..

**Example**
```
DIM fpsString as wstring * 256
fpsString = "FPS: " + Str(IrrGetFPS)
IrrGUISetText( myButton, fpsString )
```

### IrrAddWindow
**Syntax**
IrrAddWindow ( title as wstring ptr, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, modal as uinteger, parent as IRR_GUI_OBJECT ) as IRR_GUI_OBJECT

**Description**
Creates an empty window that can form the frame to contain other controls.

**Title** is a wide string that contains the title of the window.

**Top X, Top Y, Bottom X and Bottom Y** define a box in which the window is drawn

**Modal** determines if the window locks out the rest of the interface until it is closed: -
- IRR_GUI_MODAL
- IRR_GUI_NOT_MODAL

**Parent** defines the parent object of this window. This can be omitted if the window has no parent.
**IrrAddWindow**

```c
windowObject = IrrAddWindow("A Window", 4,0,200,64, IRR_GUI_MODAL)
```

**IrrAddStaticText**

**Syntax**

```c
IrrAddStaticText( text as wstring ptr, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, border as uinteger, wordwrap as uinteger, parent as IRR_GUI_OBJECT ) as IRR_GUI_OBJECT
```

**Description**

Creates a static text object on the Graphical User Interface, this simply displays the specified text in the specified box.

Text is a wide string that contains the text you want to display.

Top X, Top Y, Bottom X and Bottom Y define a box in which the text is drawn.

Border is used to draw a visible box around the text, its value should be either of:
- IRR_GUI_NO_BORDER
- IRR_GUI_BORDER

Word wrap is used to define whether text is to be wrapped around into a second line when it fills the width of the text box, its value should be either of:
- IRR_GUI_NO_WRAP
- IRR_GUI_WRAP

Parent defines the parent object of this window. This can be omitted if the object has no parent.

**Example**

```c
statictextObject = IrrAddStaticText( "Hello World", 4,0,200,16, NO_BORDER, NO_WRAP, windowObject )
```
**IrrAddButton**

Syntax

```
IrrAddButton ( Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, text as wstring ptr, parent as IRR_GUI_OBJECT ) as IRR_GUI_OBJECT
```

Description

Add a clickable button object to the gui display.

Top X, Top Y, Bottom X and Bottom Y define a box in which the button is drawn.

Id specifies a unique numerical reference for the button so events can be identified as coming from this object.

Text specifies the label assigned to the button.

Parent defines the parent object of this window. This can be omitted if the object has no parent.

Example

```
buttonObject = IrrAddButton( 16,16,96,32, 120, "My Button", windowObject )
```

**IrrAddScrollBar**

Syntax

```
IrrAddScrollBar ( horizontal as integer, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, currentValue as integer, maxValue as integer, parent as IRR_GUI_OBJECT ) as IRR_GUI_OBJECT
```

Description

Add a scrollbar object to the GUI display.

Horizontal defines if the scrollbar is horizontal or vertical, acceptable values for this field are: -
IRR_GUI_HORIZONTAL
IRR_GUI_VERTICAL

Top X, Top Y, Bottom X and Bottom Y define a box in which the scrollbar is drawn

id specifies a unique numerical reference for the scrollbar so events can be identified as coming from this object

currentValue specified the current setting of the scrollbar

maxValue specifies the maximum setting of the scrollbar

Parent defines the parent object of this window. This can be omitted if the object has no parent.

Example
scrollbarObject = IrrAddScrollBar( IRR_GUI_HORIZONTAL, 16,16,96,32, 120, 128, 156, windowObject )

IrrAddListBox

Syntax
IrrAddListBox( horizontal as integer, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, background as integer, parent as IRR_GUI_OBJECT ) as IRR_GUI_OBJECT

Description
Add a listbox object containing a list of items to the gui display.

horizontal specifies whether the scrollbar is oriented horizontally or vertically.
acceptable values are: -

IRR_GUI_HORIZONTAL
IRR_GUI_VERTICAL

Top X, Top Y, Bottom X and Bottom Y define a box in which the listbox is drawn
id specifies a unique numerical reference for the listbox so events can be identified as coming from this object.

background specifies whether the background of the listbox should be drawn. Acceptable values are:
- IRR_GUI_DRAW_BACKGROUND
- IRR_GUI_EMPTY_BACKGROUND

Parent defines the parent object of this window. This can be omitted if the object has no parent.

Example
```
listboxObject = IrrAddListBox(16,16,96,32,120,
IRR_GUI_DRAW_BACKGROUND, windowObject)
```

---

**IrrAddListBoxItem**

Syntax
```
IrrAddListBoxItem (listbox as IRR_GUI_OBJECT, text as wstring )
```

Description
Add a text element to a list box.

listbox defines the listbox gui object to add the string too.

text is the string containing the new item

Example
```
IrrAddListBoxItem(listboxObject, "Apples")
```

---

**IrrInsertListBoxItem**

Syntax
```
IrrInsertListBoxItem (parent as IRR_GUI_OBJECT, text as wstring, index as integer )
```

Description
Insert a text element to a list box.
listbox defines the listbox gui object to insert the string into.

text is the string containing the new item

index is the position at which to insert the item

Example
IrrInsertListBoxItem( listboxObject, "Pears", 3 )

IrrRemoveListBoxItem

Syntax
IrrRemoveListBoxItem ( parent as IRR_GUI_OBJECT, index as integer )

Description
Remove a text element from a list box.

listbox defines the listbox gui object to remove the string from.

index is the position of the item to be removed

Example
IrrRemoveListBoxItem( listboxObject, 2 )

IrrSelectListBoxItem

Syntax
IrrSelectListBoxItem ( parent as IRR_GUI_OBJECT, index as integer )

Description
Select a text element in a list box.

listbox defines the listbox gui object to select the item within.

index is the position of the item to be removed

Example
IrrAddEditBox

Syntax
IrrAddEditBox (text as wstring, horizontal as integer, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, border as integer, password as integer, parent as IRR_GUI_OBJECT ) as IRR_GUI_OBJECT

Description
Add a editbox object containing a list of items to the GUI display.

text is the string that is inserted into the editbox

Top X, Top Y, Bottom X and Bottom Y define a box in which the editbox is drawn

id specifies a unique numerical reference for the editbox so events can be identified as coming from this object

border specifies whether the object has a border drawn around it. acceptable values are: -

IRR_GUI_NO_BORDER
IRR_GUI_BORDER

password specifies whether the editbox is a password field that hides the text typed into it. acceptable values are: -

IRR_GUI_PASSWORD
IRR_GUI_NOT_PASSWORD

Parent defines the parent object of this window. This can be omitted if the object has no parent.

Example
editboxObject = IrrAddEditBox( "My String", 16,16,96,32, 120, IRR_GUI_BORDER, IRR_GUI_NOT_PASSWORD, windowObject )
**IrrAddCheckBox**

Syntax

IrrAddCheckBox (text as wstring, horizontal as integer, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, checked as integer, parent as IRR_GUI_OBJECT ) as IRR_GUI_OBJECT

Description

Add a checkbox object to the GUI display.

text is the string that is used to label the checkbox

Top X, Top Y, Bottom X and Bottom Y define a box in which the checkbox is drawn

id specifies a unique numerical reference for the checkbox so events can be identified as coming from this object

checked specifies whether the object starts in the checked state. acceptable values are: -

IRR_OFF
IRR_ON

Parent defines the parent object of this window. This can be ommited if the object has no parent.

Example

checkboxObject = IrrAddCheckBox( "My Checkbox", 16,16,96,32, 120, IRR_OFF, windowObject )

---

**IrrCheckCheckBox**

Syntax

IrrCheckCheckBox ( checkbox as IRR_GUI_OBJECT, checked as integer )

Description

Set the checked state of a checkbox.
checkbox defines the checkbox GUI object to check or uncheck.

checked specifies whether the object starts in the checked state. acceptable values are: -

IRR_OFF
IRR_ON

Example

IrrCheckCheckBox( checkboxObject, IRR_ON )

---

**IrrAddImage**

Syntax

IrrAddImage ( texture as IRR_TEXTURE, horizontal as integer, X as integer, Y as integer, useAlpha as integer, id as integer, parent as IRR_GUI_OBJECT ) as IRR_GUI_OBJECT

Description

Add an image object to the GUI display.

texture is a loaded texture object that is to be displayed

X, Y define a position at which the image is drawn

useAlpha specifies whether the alpha channel of the texture is to be used. acceptable values are: -

IRR_IGNORE_ALPHA
IRR_USE_ALPHA

id specifies a unique numerical reference for the image so events can be identified as coming from this object

Parent defines the parent object of this window. This can be omitted if the object has no parent.
Example
imageObject = IrrAddImage( texture, 16,16, IRR_IGNORE_ALPHA, 120, windowObject )

<table>
<thead>
<tr>
<th>IrrAddFileOpen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>IrrAddFileOpen (title as wstring, id as integer, checked as integer, modal as integer, parent as IRR_GUI_OBJECT ) as IRR_GUI_OBJECT</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Open a modal file open dialog so that a file can be selected.</td>
</tr>
<tr>
<td>title is the string that is displayed in the titlebar of the file selector window.</td>
</tr>
<tr>
<td>id specifies a unique numerical reference for the button so events can be identified as coming from this object</td>
</tr>
<tr>
<td>Modal determines if the window locks out the rest of the interface until it is closed: -</td>
</tr>
<tr>
<td>IRR_GUI_MODAL</td>
</tr>
<tr>
<td>IRR_GUI_NOT_MODAL</td>
</tr>
<tr>
<td>Parent defines the parent object of this window. This can be omitted if the window has no parent.</td>
</tr>
<tr>
<td>Example</td>
</tr>
<tr>
<td>fileOpenObject = IrrAddFileOpen( &quot;Select a bitmap&quot;, 120, IRR_GUI_MODAL, windowObject )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IrrGetLastSelectedFile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>IrrGetLastSelectedFile ( fileopenobject as IRR_GUI_OBJECT, checked as integer )</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Get the last file name selected from a file selection dialog.</td>
</tr>
<tr>
<td>Example</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td><code>fileName = IrrGetLastSelectedFile()</code></td>
</tr>
</tbody>
</table>
## Wrapper Structure Definitions

### IRR_MOUSE_EVENT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>as uinteger</td>
<td>Action determines which mouse action took place and can be one of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>following values: -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_EMIE_LMOUSE_PRESSED_DOWN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_EMIE_RMOUSE_PRESSED_DOWN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_EMIE_MMOUSE_PRESSED_DOWN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_EMIE_LMOUSE_LEFT_UP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_EMIE_RMOUSE_LEFT_UP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_EMIE_MMOUSE_LEFT_UP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_EMIE_MOUSE_MOVED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_EMIE_MOUSE_WHEEL</td>
</tr>
<tr>
<td>delta</td>
<td>as single</td>
<td>This defines the amount of movement of the mouse wheel.</td>
</tr>
<tr>
<td>x</td>
<td>as integer</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>as integer</td>
<td>These define the screen co-ordinates at which the event took place.</td>
</tr>
</tbody>
</table>

### IRR_KEY_EVENT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>as uinteger</td>
<td>The scan code for the key</td>
</tr>
<tr>
<td>direction</td>
<td>as uinteger</td>
<td>Whether the key moved up or down, this can be either of: -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_KEY_UP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- IRR_KEY_DOWN</td>
</tr>
<tr>
<td>flags</td>
<td>as uinteger</td>
<td>Bits are set in this parameter to specify whether the shift or control key was</td>
</tr>
</tbody>
</table>
keydown at the time the key action occurred

**IRR_PARTICLE_EMITTER**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_box_x</td>
<td>single</td>
</tr>
<tr>
<td>min_box_y</td>
<td>single</td>
</tr>
<tr>
<td>min_box_z</td>
<td>single</td>
</tr>
<tr>
<td>max_box_x</td>
<td>single</td>
</tr>
<tr>
<td>max_box_y</td>
<td>single</td>
</tr>
<tr>
<td>max_box_z</td>
<td>single</td>
</tr>
</tbody>
</table>

These six parameters define a box in space inside which the position of a particle is randomly created.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>direction_x</td>
<td>single</td>
</tr>
<tr>
<td>direction_y</td>
<td>single</td>
</tr>
<tr>
<td>direction_z</td>
<td>single</td>
</tr>
</tbody>
</table>

These three parameters define a direction into which the particles will be ejected as the animation plays.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_particles_per_second</td>
<td>integer</td>
</tr>
<tr>
<td>max_particles_per_second</td>
<td>integer</td>
</tr>
</tbody>
</table>

A range defining the minimum and maximum number of particles that will be created each second.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_start_color_red</td>
<td>integer</td>
</tr>
<tr>
<td>min_start_color_green</td>
<td>integer</td>
</tr>
<tr>
<td>min_start_color_blue</td>
<td>integer</td>
</tr>
<tr>
<td>max_start_color_red</td>
<td>integer</td>
</tr>
<tr>
<td>max_start_color_green</td>
<td>integer</td>
</tr>
<tr>
<td>max_start_color_blue</td>
<td>integer</td>
</tr>
</tbody>
</table>

Although particles can be textured by texturing the particle system node, these can be used to apply a range that tints the color of the particles.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_lifetime</td>
<td>integer</td>
</tr>
<tr>
<td>max_lifetime</td>
<td>integer</td>
</tr>
</tbody>
</table>

How long the particle will live, long lifespans can create very large numbers of particles.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_start_sizeX</td>
<td>single</td>
</tr>
</tbody>
</table>
**min_start_sizeY** as single  
**max_start_sizeX** as single  
**max_start_sizeY** as single  
The minimum and maximum start sizes for the particles.

**max_angle_degrees** as integer  
The maximum number of degrees that the ejected particles will deviate from the defined direction

---

**IRR_VERT**  
A vertex is a point in space that also defines a number of properties that can be applied to the corner of a triangle.

x as single  
y as single  
z as single  
The 3D position of the vertex

**normal_x** as single  
**normal_y** as single  
**normal_z** as single  
The normal direction of the vertex

**vcolor** as uinteger  
The 32bit ARGB color of the vertex

**texture_x** as single  
**texture_y** as single  
The 2 dimensional co-ordinate of the vertex when it is mapped to an applied texture (0 to 1)

---

**IRR_VECTOR**  
x as single  
y as single  
z as single  
A point that can be use for co-ordinates, directions or speeds.
Known Issues

- Currently the material type `IRR_EMT_TRANSPARENT_ADD_ALPHA_CHANNEL_REF` when applied to grass will only work when using the material type `IRR_EMT_FOUR_DETAIL_MAP` on the terrain.
- Newton Physics only uses convex hulls for collisions at the moment, this needs to be optimised so that simple boxes, spheres and cylinders can also be used to improve the performance.
- The surface of Tiled Terrains are now inverted in the X axis, this has been inherited from the Irrlicht Terrain node it has been derived from. For compatibility this will now be the normal orientation for a Terrain Tile.
- Wrappers built with g++ 4.4.0 are exhibiting some mathematical errors that can be seen in example 91. The supplied binary is built with g++ 3.4.5 and MSVC 6 and does not exhibit these errors.
- The supplied Linux binaries distributed in a separate package are built on Fedora 11 with g++ 4.4.0 using the mesa libraries. They have only been tested on Fedora 11. Any project using this wrapper should rebuild their own binaries.
- Certain cards have been shown to exhibit mipmap
texture corruption when used with textures that have dimensions that are not a power of two. It is unclear whether this is a problem with the Wrapper, Irrlicht or the Graphics Drivers. It is therefore recommended that power of two textures are always used (e.g: 128x128, 512x256, etc ...
v0.7.7

Added a new LOD manager that changes the meshes used by child mesh objects at particular distances and can fade the node in and out.

Added a new Billboard Group node that is an optimised collection of billboard meshes that can be used for custom particle effects.

Added a new Fade animator that fades, scales and eventually deletes the node from the scene.

Added a new Electrical Bolt effect node that simulates the effect of an electrical discharge. Scene Node created by Sudi with extensions from Trivtn

Added a new Beam effect node that simulates a laser or tracer beam type effect. Scene Node created by Gaz Davidson (Bitplane)

Added IrrAddBoltSceneNode to create an electrical bolt effect.

Added IrrSetBoltProperties to set the properties of an electrical bolt effect
Added IrrAddBeamSceneNode to create a beam effect.

Added IrrSetBeamSize to set the size of a beam effect.

Added IrrSetBeamPosition to set the start and end position of a beam effect.

Added IrrAddFadeAnimator to attach a fade, scale and delete animator to a node.

Added IrrAddLODManager to create an LOD management system.

Added IrrAddLODMesh to add a level of detail to a LOD management system.

Added IrrSetLODMaterialMap to allow the assignment of materials that are used as substitutes for a node's normal material to face them out over time.

Added IrrSetNodeMesh to set the mesh of a node.

Added IrrSetBillBoardColor to set the colour of a standard billboard.

Added IrrSetBillBoardSize to set the size of a standard
billboard

Added IrrAddBillBoardGroupToScene to add a billboard group management system

Added IrrAddBillBoardToGroup to add a billboard to a billboard group

Added IrrAddBillBoardByAxisToGroup to add a billboard fixed to an axis to a billboard group

Added IrrRemoveBillBoardFromGroup to remove a billboard from a billboard group

Added IrrBillBoardGroupShadows to shade billboards in a billboard group with lighting from a particular direction

Added IrrGetBillBoardGroupCount to retrieve the count of the number of billboards in a billboard group

Added IrrBillBoardForceUpdate to force a billboard group to update their orientations to the camera.

Added IrrStartAdvanced to provide an advanced start up function that allows for the use of anti-aliasing and high precision floating point calculations. Contributed by Agamemnus.
Added IrrBeginSceneAdvanced to provide an advance begin scene function providing control over whether the back buffer and buffer are erased. Contributed by Agamemnus.

Added IrrSetRenderTarget to provide control for switching the output of the drawing operations permanently to a texture. Contributed by Agamemnus.

Added IrrGet2DPositionFromScreenCoordinates to get a coordinate on a 2D plane from a ray cast through a camera. Contributed by Agamemnus.

Added IrrSetNodeColorByVertex to change the way vertex color effects all materials in a node. Contributed by Agamemnus.

Added IrrSetNodeAmbientColor to set the ambient color of all materials associated with a node. Contributed by Agamemnus.

Added IrrSetNodeDiffuseColor to set the diffuse color of all materials associated with a node. Contributed by Agamemnus.

Added IrrSetNodeSpecularColor to set the specular color
of all materials associated with a node. Contributed by Agamemnus.

Added IrrSetNodeEmissiveColor to set the emissive color of all materials associated with a node. Contributed by Agamemnus.
v0.7.6

Added IrrGetScreenSize to get the size of the screen. Contributed by Agamemnus.

Added IrrGetAbsoluteMousePosition to get the absolute position of the mouse. Contributed by Agamemnus.

Added IrrGetNodeScale to determine the scaling applied to a node.

Added IrrGetScreenSize to get the size of the screen.

Added IrrIsFullscreen to check if the application is fullscreen mode.

Added IrrIsWindowActive to check if the window is active.

Added IrrIsWindowFocused to check if the window has focus.

Added IrrIsWindowMinimized to check if the window is minimised.

Added IrrMaximizeWindow to maximise the window.
Added IrrMinimizeWindow to minimise the window.

Added IrrRestoreWindow to restore the size of the window.

Added IrrResizableWindow to make a window resizable by the user.

Changes to irrlicht_transforms.bi by Agamemnus to include the following things:

- Define multikeyold and getmouseold before undefining multikey and getmouse. Now people can still use the "old" multikey functions if they close the Irrlicht window later on in their program.
- Added "continue do" to all mouse select case cases.
- Added a clip test.

Reworked IrrlichtNewton.bi to provide a simple interface and basic example function on Newton collision. The original include file is supplied as "IrrlichtNewtonGamePhysics.bi"

Added IrrlichtODE.bi to provide almost identical physics support as is supplied for IrrlichtNewton.bi only for ODE physics. This is all based on the ODE include files.
contributed to FreeBASIC by D.J.Peters.

Added the Batching Meshes module from **Gaz Davidson (Bitplane)** if your scene contains hundreds of static scenenodes this technique can greatly increase the framerate by batching them all together in one set of geometry. You can realise as high as a **40 times** speed increase!

Added IrrCreateBatchingMesh to create an empty batching mesh that is a collection of other meshes and used to reduce node counts in a scene thereby increasing framerate.

Added IrrAddToBatchingMesh to add a standard static mesh to a batch of meshes.

Added IrrFinalizeBatchingMesh to complete the batching mesh once all meshes have been added and recover a standard mesh object.

Added IrrSetMeshMaterialTexture to set the texture image associated with a mesh.

Added IrrGetMeshVertexMemory to get the memory for the array of vertex data structures in the Irrlicht Engine.
Added the "**XEffects - Reloaded**" system produced by **BlindSide** from the irrlicht community, this contains several shaders that can have a massive visual impact on the realism of your scene. The Dynamic Shadow casting lights probably have the biggest impact. Of all of the shaders introducing quality realistic lighting into the scene.

Added IrrXEffectsStart to start the XEffects GLSL and HLSL shader support system.

Added IrrXEffectsEnableDepthPass to enable a depth rendering pass in XEffects used with the SSAO and water shaders.

Added IrrXEffectsAddPostProcessingFromFile to add a GLSL or HLSL shader to the engine from a file.

Added IrrXEffectsSetPostProcessingUserTexture to set the texture associated with certain XEffects.

Added IrrXEffectsAddShadowToNode to add a dynamic shadow to a node.

Added IrrXEffectsRemoveShadowFromNode to remove a dynamic shadow from a node.
Added IrrXEffectsExcludeNodeFromLightingCalculations to exclude a node from lighting calculations in the XEffects rendering passes.

Added IrrXEffectsAddNodeToDepthPass to add a node to XEffects depth pass rendering.

Added IrrXEffectsSetClearColor to set the background color when using XEffect post processing shaders.

Added IrrXEffectsAddShadowLight to add a shadow casting XEffects light.

Added IrrXEffectsSetShadowLightPosition to set the position of an XEffects shadow casting light.

Added IrrXEffectsGetShadowLightPosition to get the position of an XEffects shadow casting light.

Added IrrXEffectsSetShadowLightTarget to set the target of an XEffects shadow casting light.

Added IrrXEffectsGetShadowLightTarget to get the
target of an XEffects shadow casting light.

Added IrrXEffectsSetShadowLightColor to set the color of an XEffects shadow casting light.

Added IrrXEffectsGetShadowLightColor to set the color of an XEffects shadow casting light.

Tidied up styling of HTML documentation

Moved Lighting discussion to the IrrlichtWrapper Website.
v0.7.5

Remove the unsupported IrrLockOpenGLTexture from the reference and corrected heading for IrrUnlockImage.

Added command IrrGetNodeMesh to get the mesh associated with a scene node contributed by agamemnus

Modified IrrStop to close the device, this now means that resources are tidied up correctly and a new device can even be opened to change resolution.

Modified the handling of Keyboard, Mouse and GUI events so that all events are received and the GUI does not block the other two.

Removed IrrGUIEvents as it is no longer needed.

Added IrrRenameMesh to allow a loaded mesh to be renamed.

Added IrrScaleMesh to allow for the uniform scaling of a mesh without effecting its vertex normals.

Added IrrDraw2DImageElementStretch to copy an image to the screen while scaling it too. Contributed by Eric
(The Car)

Added IrrSetNodeTriangleSelector to attach a triangle selector to a specific node for multiple nodes. Recommended by Agamemnus.

Changed IrrSetMeshVertexCoords, IrrSetMeshVertexCoords and IrrSetMeshVertexSingleColor to include an extra index so for example a group [0] to [0] would include one vertex.

Added IrrGetTextureInformation to get dimensions, pitch and color format of a texture.

Added IrrGetImageInformation to get dimensions, pitch and color format of an image.

Added IrrSetNodeRotationPositionChange to allow for the incremental rotation and direction dependant incremental position change of a node while returning information that can be used to attach a camera and locate other objects and effects. This code is based on examples from Chev and also in large part the cockpit example functions by Arras.

Added IrrGetMeshBoundingBox to get the bounding box of a mesh. Contributed by Agamemnus.
Added IrrGetNodeTransformedBoundingBox to get the transformed absolute position of a nodes bounding box. Contributed by Agamemnus.

Changed IrrAddNodeShadow to add an optional parameter allowing a low resolution mesh to be used in shadow calculation.
v0.7.4

Changed IrrMaterialVertexColourAffects to IrrMaterialVertexColorAffects

Changed IrrSetNodeAnimationSpeed to take a speed parameter of type floating point to conform with Irrlicht signature changes.

Changed IrrSetNodeAnimationFrame to take a frame parameter of type floating point to conform with Irrlicht signature changes.

Added function IrrGetDistanceBetweenNodes to get the distance between two nodes efficiently.

Added function IrrAreNodesIntersecting to determine if the bounding boxes of two nodes are intersecting.

Added function IrrIsPointInsideNode to determine if a point is inside the bounding box of a node.

Added function IrrGetCollisionResultPosition to add eplisoid collision management supplied by 'The Car'.

Corrected IrrGetCollisionGroupFromMesh to return a
triangle selection.highlighted by super_castle

Updated Newton functions provided by SiskinEDGE to support mesh buffer.

Corrected missing parameters in
IrrSetMeshVertexSingleColor highlighted by agamemnus

Updated IrrRemoveCollisionGroup to allow it to take a scene node parameter that lets the selector be removed from an associated node.

Updated IrrGetCollisionGroupFromComplexMesh and IrrGetCollisionGroupFromBox so the created selector is attached to the scene node they are created from.

Added function IrrGet3DPositionFromScreenCoordinates to get a 3D position on a plane from a set of screen coordinates, contributed by agamemnus

Included correction to CParticleEmissionEffector constructor used in IrrAddStopParticleAffecter to allow the stop time to be set correctly. Contributed by The Car

Added example 91_Example_DistanceAndCollision.bas to demonstrate new distance and collision functions.
Added example 92_Example_MovingEntitiesByCollision.bas to demonstrate moving entities around a scene through collision.

Added example 93_Example_3DPositionFromScreen.bas to demonstrate getting a point on a plane from screen coordinates.

Corrected example 50_Example_Fading_Nodes_Out.bas to correctly demonstrate fading nodes.

Resolved all compilation warnings at warning level 3 when built under msvc6 and g++3.4.5
v0.7.3

Added in new video feature queries supported by Irrlicht for Vertex Buffer Object, Alpha to Coverage, Color Masks, Multiple Render Targets, Blend settings for Multiple Render Targets and Geometry Shaders.

Added IrrSetMaterialBlend for setting the Blend settings of materials associated with the ONETEXTURE_BLEND material.

Added IrrAddBillboardTextSceneNode for adding a billboard with text on it that can be used as a scaling label in a 3D environment.

Added IrrSetTime for setting the irrlicht animation time in the system.

Updated example 23_Example_Video_Features.bas to display new feature capabilities.

Updated example 81_Example/GLSL_Shader_Materials.bas to display a range of GLSL materials.

Added example 86_Example_Screenshot_to_texture.bas
to demonstrate capturing screenshots to textures.

Added example 87_Example_MultiPass_Rendering.bas to demonstrate blending several rendering passes to the display.

Added example 88_Example_Billboard_Text.bas to demonstrate adding billboard text and attaching it to a node.

Added example 89_Example_Orthogonal_Camera.bas to demonstrate the use of orthogonal cameras.

Added example 90_Example_Collision_Point.bas to demonstrate a new ray collision test feature.

Added IrrDisableFeature for disabling video driver features.

Added IrrGetNodeBoundingBox to get the bounding box of a node. supplied by Agamemnus

Added IrrSetMaterialLineThickness to set the material line thickness of vertex drawing operation. supplied by Agamemnus

Added IrrSetMeshVertexSingleColor to set groups of
vertices in a mesh to a uniform color. supplied by Agamemnus

Added IrrGetCollisionGroupFromMesh to create a collision group from the triangles in a mesh.

Added IrrRemoveCollisionGroup to remove a collision group freeing the memory it uses.

Added IrrGetNodeAndCollisionPointFromRay to detect a collision between a ray and scene node objects that have had collision groups created for them.

Added IrrDisableFeature to disable particular video features.

Added IrrGetMeshFrameCount to get the number of frames in an animated mesh.

Added IrrGetMeshBufferCount to get the number of mesh buffers in a mesh.

Added IrrSetCameraOrthogonal to set the camera to display an Orthogonal view instead of the default perspective view.

Updated IrrGetMeshIndexCount, IrrGetMeshIndices,
IrrSetMeshIndices, IrrGetMeshVertexCount, IrrGetMeshVertices, IrrSetMeshVertices, IrrSetMeshVertexColors, IrrSetMeshVertexCoords and IrrSetMeshVertexSingleColor to work from individual mesh buffers instead of the entire mesh as a single object.

Corrected IrrSetMeshVertices and IrrSetMeshVertexCoords to correctly set the bounding box of the mesh they modify.
v0.7.2

Changed the IrrGetScreenShot command to capture a portion of the screen to texture rather than the whole screen to an image.

Corrected a bug where mouse events were not being passed to the active camera when keyboard and mouse is being captured.

Corrections for bugs in passing variables to GLSL materials.
v0.7.1

Updated documentation with new commands

Updated IrrlichtWrapper to compile with Irrlicht version 1.7.1

Updated IrrlichtWrapper to compile with g++ upto version 4.4.1

Added IrrSetMeshVertexColors for setting the color of vertices supplied by Agamemnus

Added IrrSetMeshVertexCoords for setting the coordinates of vertices supplied by Agamemnus

Added IrrAddEmptySceneNode to add an empty scene node to the scene supplied by Agamemnus

Added IrrGetCameraUpDirection to get the up direction of the active camera supplied by Agamemnus

Added IrrGetScreenShot to grab a screenshot into an image

Added IrrRemoveImage to remove an image from
memory

Added IrrMaterialVertexColourAffects to set the material property that vertex colour effects

Added IrrGUIClear to clear all GUI objects from the screen

Added IrrGUIRemove to remove a specific GUI object

Added IrrGUIGetText to get the text associated with a GUI object

Added IrrGUISetText to set the text associated with a GUI object

Added IrrAddWindow to add a blank window GUI object to the display

Added IrrAddButton to add a clickable button GUI object to the display

Added IrrAddScrollBar to add a scrollbar GUI object to the display

Added IrrAddListBox to add a listbox GUI object to the display
Added IrrAddListBoxItem to add items to a listbox GUI object

Added IrrInsertListBoxItem to insert items into specific positions in a listbox GUI object

Added IrrRemoveListBoxItem to remove a specific item from a listbox GUI object

Added IrrSelectListBoxItem to select a specific item in a listbox GUI object

Added IrrAddEditBox to add an editable editbox GUI item

Added IrrAddImage to add a 2D image GUI item

Added IrrAddCheckBox to add a clickable checkbox GUI item

Added IrrCheckCheckBox to set the check state of a checkbox GUI item

Added IrrAddFileOpen to add a file open GUI item

Added IrrGUISetFont to set the font associated with the
GUI

Added IrrGUISetColor to set the colour of the GUI

Added IrrGUIEvents to start or end capturing GUI events

Added IrrGUIEventAvailable to check if there is a GUI event available

Added IrrReadGUIEvent to read a GUI event

Added IrrGetLastSelectedFile to get the last select file from a GUI file selection object

Added IrrSetViewPort to allow rendering to be confined to an area of the screen for split screen effects

Added IrrSetCameraAspectRatio to allow the aspect ratio of a camera to be changed primarily for split screen effects

Updated example 36 materials to work with Irrlicht 1.7.1

Added examples to demonstrate a GLSL material, GUI, Split Screen, Freetype fonts and Embedded OpenGL commands.
Added example 85_Example_Split_Screen.bas to demonstrate splitscreen rendering.
v0.7

Newton Physics Engine support for convex hulls by SiskinEDGE

Modified to work with Irrlicht v1.6

Support for hardware accelerated meshes allowing static models to.

Support for Crossed grass, correction to reduce grass 'popup'

Particle systems now have minimum and maximum particle size properties in their configuration structures
v0.6.2

Corrections to allow compilation under linux without modification

Captured keyboard commands are now also sent to the active camera.

Correction to example 44 to prevent a crash under build for Direct3D

Corrections to build for DirectX to support all 236 commands in the library
v0.6.1

Corrected examples to not use literal paths thanks AlexZ

Added column support to push particle affector

Added IrrSetColumnDistanceOfEffect to allow the push affector columnn effect to be dynamically changed

Added IrrCreateImage to create blank images

Added IrrLockImage to get the pixels for an image

Added IrrUnlockImage to release the pixels for an image

Added IrrGetSphericalTerrainSurfacePosition for getting coordinates from a sphereical terrains face number and a logical X,Z coordinate

Added IrrGetSphericalTerrainSurfacePositionAndAngle for getting coordinates from a sphereical terrains face number and a logical X,Z coordinate, this also returns the angle of the point at the surface, useful for placing objects on the spheres surface.

Added IrrGetSphericalTerrainLogicalSurfacePosition for
getting the logical face number and X,Z coordinates from normal spacial coordinates. (This is not 100% accurate yet) (not supported in DirectX in this release)

Added IrrAddMayaCamera for adding a maya style camera to the view you can click and drag to move, revolve and zoom this camera. Suggested by Alvaro

Updated documentation with new commands

Added examples 70 to 76 to demonstrate features

Minor correction to example 12 thanks Alvaro

Significant improvement to Example 14 to show proper manual camera control

Corrected IrrSetCameraUpDirection in the manual thanks AlexZ
v0.6

Included a simple hacha.jpg file to correct examples 21, 41 and 43

Added IrrAddClouds to create a clouds node by Gaz Davidson and Nikolaus Gebhardt

Added IrrAddGrass to create a patch of grass across a terrain by Gaz Davidson

Added IrrSetSkyDomeColor for tinting the texture on a skydome

Added IrrAddStopParticleAffector to stop particle flows

Added IrrSetSkyDomeColor for setting the color of vertices in a skydome

Added IrrAddZoneManager for adding automatic node management

Added IrrGetImage for loading image files that do not use video memory

Added IrrAddTerrainTile for adding terrain objects that
can be tiled!

Added IrrScaleTileTexture for scaling tiled terrain objects

Added IrrAttachTile for attaching tilable terrain objects together

Added IrrSetTileColor for loading the vertex color of terrain tiles

Added IrrSetTileStructure for loading the height and vertex color of terrain tiles

Added IrrSetGrassDensity for changing grass density

Added IrrSetGrassWind for changing the strength of wind affecting the grass

Added IrrSetZoneManagerAttachTerrain for attaching a terrain to a zone

Added IrrRevolveCamera for 6DOF (six degrees of freedom) camera motion for spacecraft and aircraft

Added IrrSetMousePosition for moving the mouse on the screen
Added IrrTransparentZWrite for ordering transparent objects. This causes a performance hit but can correct situations where transparent objects appear in the wrong order on the screen

Added IrrGetTerrainTileHeight to get the height of a point on a tile terrain

Added IrrGetTerrainHeight to get the height of a point on a normal terrain

Added IrrAddLensFlare to create a camera lens flare effect

Added IrrSetFlareScale to set the scale of the lens flare effect

Added IrrSetSkyDomeColorBand to add a band of color to a skydome

Added IrrSetSkyDomeColorPoint to add a circle of color radiating out from a point to the vertices of a skydome

Added IrrSetZoneManagerProperties to set the properties of a zone management object

Added IrrSetNodeName to set the name of a node
Added IrrGetNodeName to get the name of a node

Added IrrGetRootSceneNode to get the root object within which all other nodes are contained

Added IrrGetChildCollisionNodeFromRay to get a child object hit by a ray

Added IrrSetZoneManagerBoundingBox to get the bounding box of a zone management object

Added IrrGetChildCollisionNodeFromPoint to get a child object contained in a specific location

Added IrrGetNodeFirstChild to get the first child object of a specific object

Added IrrIsNodeLastChild to check if an object is the last child object of a specific object

Added IrrGetNodeNextChild to get the next child object of a specific object

Added IrrGetNodeAbsolutePosition to get the absolute position of a node taking into account the positions of all of its parents
Added IrrSetNodeParent to set the parent node of a specific node

Added IrrSetCameraUpAtRightAngle to adjust the camera up angle so that it is at right angles to the camera vector

Added IrrAddSphericalTerrain to create a spherical terrain object for planets

Added IrrScaleSphericalTexture to scale the textures of a spherical terrain object

Added IrrAddStopParticleAffector to create a particle affector that stops the flow of particles from a specific emitter

Added IrrAddParticlePushAffector to create a particle affector that pushes particles in particular directions

Added IrrAddColorMorphAffector to create a particle color morph affector that changes the color of a particle over its lifetime

Added IrrSetFurthestDistanceOfEffect to set the furthest distance that the push affector has any influence
Added IrrSetNearestDistanceOfEffect to set the nearest distance that the push affector has any influence

Added IrrSetCenterOfEffect to set the center of the effect of a push affector

Added IrrSetStrengthOfEffect to set the strength of a push affector

Added IrrAddSplineAffector to create a spline path affector for particles
v0.5.1

Included a new parameter into IrrStart that allows the wrapper to enable Irrlichts vertical syncronisation to prevent 'tearing' of the picture by syncronising the refresh of the display. (suggested by Daiwa)
V0.5

Eponasoft contributed support for skydomes with IrrAddSkyDomeToScene

Eponasoft contributed support for water nodes with IrrAddWaterSurfaceSceneNode

Eponasoft contributed support for hill planes with IrrAddHillPlaneMesh

Eponasoft contributed support for loading IrrEdit scenes with IrrLoadScene

Added support for saving the current scene to a file that can be loaded into irrEdit with IrrSaveScene.

Added support for finding a node in the scene by its ID with IrrGetSceneNodeFromId

Added support for finding a node in the scene by its name with IrrGetSceneNodeFromName

Support for adding a simple cube scene node

Support for adding a simple Sphere scene node
Added support for the Irrlicht call to set texture creation flags with IrrSetTextureCreationFlag

Added support for the Irrlicht call to draw a 3D line into the display with IrrDraw3DLine

Added support for the Irrlicht call to create a texture that is suitable for the scene manager to use as a surface to which it can render its 3D scene with IrrCreateRenderTargetTexture. Each dimension must be a power of two for example 128x128 or 256x256

Added support for the Irrlicht call to draw scene manager objects to a texture surface with IrrDrawSceneToTexture. The texture must have been created with a call to IrrCreateRenderTargetTexture

Added support for the Irrlicht call to create a blank texture with IrrCreateTexture

Added support for the Irrlicht call to lock the texture and returns a pointer to the pixels with IrrLockTexture

Added support for the Irrlicht call to unlock the texture with IrrUnlockTexture, presumably after it has been modified and recreate the mipmap levels
Added support for the Irrlicht call to remove a texture from memory with `IrrRemoveTexture` freeing the resources.

Added support for the Irrlicht query feature command. The video card can now be queried to see which video card features it supports.

Added support for the Irrlicht call to add a mesh to the scene as a static object with `IrrAddStaticMeshForNormalMappingToScene`, the mesh is altered so that it is suitable for the application of a Normal or Parallax mapping material, at this time any animation information is lost.

Added support for Irrlicht call to make a normal map from a greyscale bump map texture with `IrrMakeNormalMapTexture`.

Added support for Irrlicht call to Get the number of materials associated with a node with `IrrGetMaterialCount`.

Added support for Irrlicht call to Get the material associated with the node at the particular index with `IrrGetMaterial`.
Added support for Irrlicht call to Set how shiny the material is with IrrMaterialSetShininess the higher the value the more defined the highlights

Added support for Irrlicht call to set the color of specular highlights on the object with IrrMaterialSetSpecularColor

Added support for Irrlicht call to set the color of diffuse lighting on the object with IrrMaterialSetDiffuseColor

Added support for Irrlicht call to set the color of ambient light reflected by the object with IrrMaterialSet AmbientColor

Added support for Irrlicht call to set the color of light emitted by the object with IrrMaterialSetEmissiveColor

Added support for Irrlicht call to set material specific parameter with IrrMaterialSetMaterialTypeParam

Added a materials interface with a series of functions for adding GPU Shader programs for modern graphics cards. This interface is in an early phase of development and requires more work.

Added support for Irrlicht calls for creating Materials
based upon GPU programs for cards that support the feature: IrrAddHighLevelShaderMaterial, IrrAddHighLevelShaderMaterialFromFiles, IrrAddShaderMaterial and IrrAddShaderMaterialFromFiles.

Added support for Irrlicht calls for creating constants to materials with: IrrCreateNamedVertexShaderConstant, IrrCreateNamedPixelShaderConstant, IrrCreateAddressedVertexShaderConstant and IrrCreateAddressedPixelShaderConstant.

IrrGetMS3DJointNode and IrrGetDirectXJointNode are now replaced with a universal IrrGetJointNode call.

Added support for Irrlicht calls for setting the current frame number being played in the animation with IrrSetNodeAnimationFrame.

Added support for Irrlicht calls for setting the time in seconds across which two animation frames are blended with IrrSetTransitionTime. This can be used to smoothly animate a model between two poses where no animation data exists.

Added support for Irrlicht calls for animating the mesh based on the position of the joints, this should be used at
the end of any manual joint operations including blending and joints animated using IRR_JOINT_MODE_CONTROL and IrrSetNodeRotation on a bone node with IrrAnimateJoints

Added support for Irrlicht calls for setting the animation mode of joints in a node with IrrSetJointMode allowing them to be blended or controlled programatically.

Added support for Irrlicht call to write the first frame of the supplied animated mesh out to a file using the specified file format with IrrWriteMesh

Added support for Irrlicht calls for creating a particle emitter for an animated mesh scene node with IrrCreateAnimatedMeshSceneNodeEmitter

Added support for Irrlicht calls for creating a point attraction affector with IrrAddParticleAttractionAffector. This affector modifies the positions of the particles and attracts them to a specified point at a specified speed.

Added support for Irrlicht calls for creating a rotation affector with IrrCreateRotationAffector. This affector modifies the positions of the particles and attracts them to a specified point at a specified speed per second.
Added support for Irrlicht calls for setting direction the emitter emits particles with IrrSetParticleEmitterDirection.

Added support for Irrlicht calls for setting minimum number of particles the emitter emits per second with IrrSetParticleEmitterMinParticlesPerSecond.

Added support for Irrlicht calls for setting maximum number of particles the emitter emits per second IrrSetParticleEmitterMaxParticlesPerSecond.

Added support for Irrlicht calls for setting minimum starting color for particles with IrrSetParticleEmitterMinStartColor.

Added support for Irrlicht calls for setting maximum starting color for particles IrrSetParticleEmitterMaxStartColor.

Added support for Irrlicht calls for enabling or disabling an affector with IrrSetParticleAffecterEnable.

Added support for Irrlicht calls for altering the fadeout affector changing the fade out time with IrrSetFadeOutParticleAffecterTime.
Added support for Irrlicht calls for altering the fadeout affector changing the target color IrrSetFadeOutParticleAffectorTargetColor.

Added support for Irrlicht calls for altering the direction and force of gravity with IrrSetGravityParticleAffectorDirection.

Added support for Irrlicht calls for setting the time in milliseconds when the gravity force is totally lost and the particle does not move any more with IrrSetGravityParticleAffectorTimeForceLost.

Added support for Irrlicht calls for setting whether or not this will affect particles in the X direction with IrrSetParticleAttractionAffectorAffectX.

Added support for Irrlicht calls for setting whether or not this will affect particles in the Y direction with IrrSetParticleAttractionAffectorAffectY.

Added support for Irrlicht calls for setting whether or not this will affect particles in the Z direction with IrrSetParticleAttractionAffectorAffectZ.

Added support for Irrlicht calls for setting whether or not
the particles are attracting or detracting with 
IrrSetParticleAttractionAffectorAttract.

Added support for Irrlicht calls for setting the point that 
particles will attract to with 
IrrSetParticleAttractionAffectorSetPoint.

Added support for Irrlicht calls for setting the point that 
particles will rotate about with 
IrrSetRotationAffectorSetPivotPoint.

Added support for Irrlicht calls for clearing all meshes 
that are held in the mesh cache but not used anywhere 
else with IrrClearUnusedMeshes

Added support for Irrlicht calls for removing all nodes in 
a scene with IrrRemoveAllNodes. This is useful for 
example for clearing the scene between levels.

Added support for Irrlicht calls for saving a screenshot 
out to a file with IrrSaveScreenShot

Added support for Irrlicht calls for setting the Ambient 
color emitted by the light with IrrSetLightAmbientColor

Added support for Irrlicht calls for setting the the light 
strength fading over distance with IrrSetLightAttenuation
Added support for Irrlicht calls for setting whether the light casts shadows with IrrSetLightCastShadows

Added support for Irrlicht calls for setting the Diffuse color emitted by the light with IrrSetLightDiffuseColor

Added support for Irrlicht calls for setting the lights strength's decrease between Outer and Inner cone with IrrSetLightFalloff

Added support for Irrlicht calls for setting the angle of the spot's inner cone with IrrSetLightInnerCone

Added support for Irrlicht calls for setting the angle of the spot's outer cone IrrSetLightOuterCone

Added support for Irrlicht calls for setting the Radius of light with IrrSetLightRadius. Everything within this radius be be lighted.

Added support for Irrlicht calls for setting the Specular color emitted by the light with IrrSetLightSpecularColor

Added support for Irrlicht calls for setting the type of the light with IrrSetLightType
Adding lighting attributes has provided support to defining spot lights and distant light, the position and direction of these directional light sources can be changed by moving and rotation the light node.
Overview

IrrlichtWrapper is supplied with three sets of files. MSVC versions, GCC versions and Linux Versions. The default version of the files are those compiled with MSVC, these provide support for OpenGL, Direct X 8, Direct X 9 and software rendering. The GCC and Linux versions support both OpenGL and Software rendering.
GCC Compiler

To use the GCC DLLs you need to rename the supplied files: -

1. Rename Irrlicht.dll to Irrlicht_ms.dll
2. Rename IrrlichtWrapper.dll to IrrlichtWrapper_ms.dll
3. Rename Irrlicht_gcc.dll to Irrlicht.dll
4. Rename IrrlichtWrapper_gcc.dll to IrrlichtWrapper.dll

You will now be able to build the IrrlichtWrapper with the GCC compiler and run the examples. You will not be able to use either Direct X rendering mode..
Linux Platform

The supplied linux library files libIrrlicht.so and libIrrlichtWrapper.so can be installed to your, /usr/lib directory, you will then be able to use the freebasic compiler to build and run the example files.

Linux support is limited at the moment the release is tested under Software Rendering on Fedora 11. The supplied libraries may provide reduced performance and it is recommended that you rebuild both the Irrlicht Library and the IrrlichtWrapper library on your own system to utilise the OpenGL support that may be available on your system.