



Xtreme3D is a 3D engine for Game Maker, a popular game creation tool. As we know, GM is mainly focused on 2D games, and provides only basic 3D graphics functionality that is not very suitable for serious use. Fortunately, GM supports plugging in dynamic links libraries (DLLs) that can greatly expand its capabilities. Xtreme3D is such a library. Using it you can render 3D graphics of any complexity in GM. Xtreme3D supports many advanced graphics technologies such as shaders and frame buffers, and allows you to create games with high quality graphics. Nowadays it is the only actively developed 3D engine for classic Game Maker.

This guide was created to help beginners, providing comprehensive material on using Xtreme3D: step-by-step tutorials with code examples, a list of engine's functions with detailed explanations, computer graphics glossary and much more. We hope that this will help learning Xtreme3D, making the process easy, interesting and fascinating.

Authors and contributors:

Gecko - initiator, chief editor and article writer;

Rutrable aka Hacker - technical editor and proofreader;

Bill Collins aka williac0374 - creator of an English version.

Kudos also go to **Xception**, **Bami** and all others who, in one way or another, kindly provided any useful information for the guide.

Visit our website: <http://xtreme3d.tk>. There you will find Xtreme3D examples, games, useful utilities, a collection of current and historical DLLs for Game Maker and much more.

For support, please refer to our forum: <http://offtop.ru/xtreme3d>

You can find Xtreme3D releases and source code in the project's GitHub repository: <https://github.com/xtreme3d/xtreme3d>.

Good Luck!

Xtreme3D version history

Xtreme3D 0.x to 2.x (2003-2006)

Xtreme3D project was launched in 2003. The engine was written in Delphi using GLScene, de facto standard 3D engine for Object Pascal, by German programmer known as **Xception**. Unfortunately, we don't know his real name.

The first stable version of Xtreme3D (0.9) was released in 2003. Its possibilities were quite modest: it supported only MD2 and MD3 animated models, 3DS, MS3D, and OBJ static models, a number of built-in primitives, sprites, 2D and 3D text, lensflare, particles and skydome. Its API resembled native GML function syntax.

The main goal of the project was creating an engine suitable for an RPG like Dungeon Master. While originally the author haven't planned any further development of Xtreme3D, in later versions its functionality quickly evolved to exceed the needs of a specific game genre.

For example, version 1.7 (2004) featured dynamic water. However, the much more serious breakthrough was version 2.0 (2006): it featured total API revision, material library and lightmaps support, dynamic cubemaps, whole lot of new models formats support, render-to-texture, loading assets from PAK archive, built-in shaders like bump mapping and cel shading, multitexturing, octrees and quadrees, DDS support, and, importantly, an integrated physics engine - ODE.

The latest release from Xception (2.0.2.0), in addition to numerous bugfixes, featured two major innovations: terrain renderer and DCE, an engine for collision detection and response. Functionality of the engine equaled the level of 1997-2003 commercial games. Xtreme3D 2.0 allowed to create something similar in terms of graphics to Quake 2 and 3, Half-Life, GTA 3, The Elder Scrolls: Morrowind, and so on (i.e., everything from pre-shader era).

Unfortunately, Xception abandoned his project. The engine was closed source and therefore haven't been developed for a long time. Technology, meantime, moved forward.

Xtreme3D 3.0 (25.08.16)

Since 2009 Russian-speaking Xtreme3D community constantly nurtured an idea to recreate the engine from scratch, fixing bugs and adding the missing functionality. Bearing in mind that Xtreme3D is a thin wrapper of GLScene, this task seemed simple and purely technical. However, unforeseen problems arose, and the project had been delayed. Ultimately this work, begun in 2009 and experienced a long period of stagnation and a few flashes of activity, was completed only in 2016. This is how Xtreme3D 3.0 appeared.

This version of Xtreme3D, like its ancestor, is written in Delphi using popular GLScene library - in fact, the engine can be seen as almost full GLScene wrapper for Game Maker. Xtreme3D 3.0 is based on modified and extended GLScene 1.0.0.0714.

The following is a brief list of new features in Xtreme3D 3.0:

- GLSL shaders support (GLSL version 1.1 and 1.2)
- Shadow maps
- MSAA anti-aliasing 2x and 4x with NVIDIA Quincunx support
- Fast offscreen rendering using p-buffers
- Proxy objects
- LOD (LODka 3D) and B3D (Blitz3D) model formats support
- MD3 tags support
- Animation blending
- Material scripts
- TexCombine shader
- Phong shader
- Procedural textures based on Perlin noise
- Linear waves support for the water
- New geometric primitives including frustum, dodecahedron, icosahedron, and teapot
- Improved explosion effect for meshes (in particular, now it is possible to return exploded mesh into its original state)
- Rendering of grids
- Debug rendering of Dummycube objects
- Many new functions for Camera objects
- An ability to copy the transformation matrix (local and absolute) from one

object to another. You can also attach objects to skeleton bones and update the local matrix of an object manually, which allows to use any transformation model without restriction to Euler angles

- An ability to apply impulses (instant velocity changes) to dynamic objects in DCE. Also DCE now supports terrain
- Improved ODE physics. Better support for Freeform objects (both for static and dynamic bodies), as well as terrain. In addition, it is now possible to set position when creating a geometry.

Xtreme3D 3.1 (30.09.16)

The first update of the new Xtreme3D branch. Most important changes include:

- Improved API for Freeform objects. It is now possible to assemble Freeform manually from vertices and triangles. There are functions to transform individual meshes, as well as to save Freeform to file
- New file formats support: CSM and LMTS (which were absent in 3.0), X, ASE, DXS
- Ragdoll support for ODE
- Movement object, which allows to define movement trajectories with line segments, Bezier curves, cubic splines and NURBS
- Improved BumpShader, which now supports shadow maps and automatic tangent space calculation
- Improved PhongShader, which now supports textures
- HUDShape objects - 2D shapes including rectangle, circle, line segment and polygon
- Improved API for sprites - now they support texture atlases (i.e., using only a portion of the texture - to make an animated sprite, for example), as well as user-defined origin for sprite rotation
- Transparency support for PNG
- Functions for querying texture size.

Xtreme3D 3.2 (21.10.16)

- FBO support. With FBOs you can effectively implement multipass rendering and various complex postprocessing effects
- New, shader-compatible multitexturing mechanism for materials. Now material can have up to 8 textures, and GLSL shaders can automatically accept them as parameters
- ViewerRenderObject function for rendering individual objects and their

children

- MaterialLoadTexture function
- Fixed a bug in ObjectSetParent function.

Xtreme3D 3.3 (26.11.16)

- New Freeform functions that allow to read and modify geometry (vertices, normal, indices, etc.)
- Material overriding for ordinary viewers and FBOs. This allows to specify a single material which should be applied to objects when rendering, ignoring their own materials
- Different color formats support for FBO, including 16 and 32-bit floating point
- FBORenderObjectEx function
- Optional rendering shadows to user-specified FBO instead of internal one
- ViewerGetSize, ViewerGetPosition, ViewerIsOpenGLExtensionSupported.

Xtreme3D 3.4 (30.12.16)

- TTF fonts support via Freetype library and rendering UTF-8 text
- ObjectHash functions (hash table for storing any Xtreme3D objects)
- Better FBORenderObjectEx - new arguments allow you to selectively clean color buffer and depth buffer, and copy the contents of the FBO in the main framebuffer
- Passing view and inverse view matrices to GLSL shader, and a special parameter that allows the shader to know whether there is a texture in a specified texture unit
- For performance reasons the engine now doesn't automatically generate octree and tangents/binormals when loading Freeform from file. This should be done manually, if necessary, with FreeformGenTangents and FreeformBuildOctree functions
- Fixed a bug in MaterialCubeMapLoadImage. Also in GLSL shaders now support seamless cubemapping for arbitrary mip levels of a cubemap (if GL_ARB_seamless_cubemap is supported).

Xtreme3D 3.5 (04.02.17)

- ClipPlane object
- Improved PhongShader and BumpShader, they now support lsSpot type lights, fog, transparency, and shadeless rendering (if lighting is turned off in the viewer

settings). Transparency is set via the alpha channel of the diffuse texture, or, alternatively, through the alpha value of material's diffuse color (only for PhongShader)

- New material functions: MaterialCullFrontFaces, MaterialSetZWrite
- New ODE functions to manually set velocity, position and rotation for dynamic bodies.

Xtreme3D 3.6 (17.12.17)

This is one of the biggest releases in 3.x branch, development of this version lasted for more than six months.

- Support for Windows encoding for TTF fonts, as well as any custom 8-bit encoding
- New material functions: MaterialSetTextureExFromLibrary, MaterialGetNameFromLibrary
- New Freeform functions: FreeformSetMaterialLibraries, FreeformMeshFaceGroupSetLightmapIndex, FreeformMeshFaceGroupGetLightmapIndex
- Special proxy object for Actors (ActorProxy)
- ActorMoveBone and ActorRotateBone are back. Also there is a function to switch visibility of Actor meshes (ActorMeshSetVisible)
- ObjectInFrustrum is back
- Functions for getting mouse and keyboard input
- Functions for window creation and management
- Functions for RGB color packing
- Experimental functions to save the scene to file and load from file
- GLSL shaders now don't show an empty error message if there are no errors
- Fixed bug with incorrect specular highlights in PhongShader and BumpShader.

Lesson 1

Basics of Xtreme3D. The theory of

Level: Beginner

Version of the Xtreme3D: 3.0.x

Author: Gecko

So, what is Xtreme3D?

First of all, this scene. The stage on which the play "actors" - the objects. The object - the main concept in the Xtreme3D. This is not the same as the object of Game Maker. Objects of Xtreme3D created and managed solely in the software code. The three-dimensional models, entities, special effects, the sky in the background, pictures and text on the screen - all this objects. The objects of any type are controlled, in most cases, by the same means.

One of them is the hierarchy. Any object can have one or more subordinate objects - "the descendants". Is it becomes their "parent" (without the quotes). The descendants, in turn, may have their own descendants, and so on. On this principle and builds the whole scene. What are the benefits? For example, if you disable any object (it will be invisible on the screen), all his descendants (and, accordingly, the descendants of the descendants of the default) will also be disabled. In this case, you can include any of the descendants, and this will not affect the parent. This "inheritance" - the most obvious from the features of the hierarchy. But this is only the tip of the iceberg. The descendants of the object can inherit not only his condition, but also a number of other characteristics. For example, the object becomes tied to its parent - where a parent, offspring. The descendant can move at any distance, but only concerning the location of its parent. It is easier to understand, imagine a steamship: passengers can move freely on the deck or stand on the spot, but all of them, as a matter of fact, moving along with the steamship.

Thus, we gradually come to the displacement. In computer graphics any movements, rotation, and scale are combined under the general term "transformation". The transformation of the Xtreme3D is carried out by three mutually perpendicular to the coordinate axes: X, Y, and Z. The X axis is

directed to the right, the Y axis is up, and the Z-axis - "in depth". This is a coordinate system is called the Cartesian rectangular, on behalf of the French mathematician Rene Descartes. The location of any point in the coordinate system is defined by three of its projections on the axle. For example, points (-1,0,-1), (-1,0,1), (1,0,1) and (1,0,-1) form a square 2x2, which lies on the XY plane. Thus, any object of Xtreme3D has three coordinates, describing the point of his position in space. Usually this point coincides with its own center of the object. The position of the object in space can be transferred on the absolute start coordinates scene (0,0,0), or on the coordinates of its parent, if there is one. In the second case referred to the so-called local coordinates of an object in which the point (0,0,0) the descendant is always the point of the provisions of the parent.

Moving an object is carried out by means of the vector. Vector - this segment, directed from the point (0,0,0), in local or absolute coordinates, to any other arbitrary point in the same coordinates and describes the coordinates of the point. The object can be moved to any distance to the side, which shows the vector. For example, vector (0,10,0) moves the object on the ten units of up.

But, since it was not always easy to manually calculate the vector for the desired direction of travel, used in the corners of the Eulerian model (named by the name of Leonard Euler, the Swiss scholar). They define the angle of rotation of the object around its local axes X, Y and Z. The zero angle is perpendicular to the axis. Rotate around the X-axis is called the pitch, around the Y-axis, Turn around the axis of Z - Roll. It is not difficult to guess that, for example, the angle (90,0,0) tips the object back to 90 degrees.

Received as a result of these turns of the Transformation determines the direction of the object (Direction), which also describes a single vector. The unit vector is different from the normal that its length equals one (for the vector describing the direction, the length does not matter). For example, vector directions (0,1,0) corresponds to the corner (90,0,0).

Thus, we get another characteristic of the object - its direction.


Moving an object in his direction is accomplished by multiplying the vector of the direction at a distance of displacement:

$$(0, 0, 1) * 10 = (0, 0, 10)$$

$$(1, -1, 1) * 10 = (10, 10, 10)$$

And, as a result, we get a new object coordinates (relative to the previous).

In addition to the vector of Direction, an object is also automatically calculated vectors Up and Left, indicating, respectively, up and to the left on the direction. These three mutually perpendicular vectors form a new coordinate system (Left = X, Up = Y Direction = Z), which they inherit all descendants of the object. For the descendant, it becomes the local and all of its transformation are specified in it. Is the following: if we create a descendant of the object and move it to some distance away, when the parent of the descendant will rotate around it, as the earth revolves around the sun! This is an incredibly useful property of the hierarchy. As you will soon make sure that it is used in the Xtreme3D literally every step of the way.

The third is the transformation - scale. It changes the size of the object on three axes (the width, height, length). You can scale on the current value of the scale or completely (to set a specific size in absolute units). Also scaled and coordinate system, The  descendants of the object. That is, the descendant will not only be reduced, but also close to the parent, as if we reduce the whole system of objects. Strictly speaking, the parent+descendants - this is the system objects. It can be considered as one big object, consisting of separate logically grouped items.

Thus, all of the objects in our scene, there are certain mathematical patterns. If you are aware of these patterns, explore the Xtreme3D will be easy.

Lesson 2

The creation of a simple scene

Level: Beginner

Version of the Xtreme3D: 3.0.x

Author: Gecko

Before starting the practical lessons, let something be clarified. Xtreme3D is a dynamic-link library (DLL). A DLL is nothing but a compiled set of instructions written in any programming language, with a view to using the programs in any other language that supports dlls. Xtreme3D, for example, written in Delphi, and it contains about 580 such instructions. Call from for ease of functions. Using the Game Maker we can create scripts, each of which will call a function from the library. This will cause the function through the GML code under names of scripts. This is a typical example of the functions of the Xtreme3D: [ObjectSetMaterial](#)(object,'material'). Some of the functions return different numeric and string values. For example, when an object is returned to its identifier (ID), which must be recorded in a variable for further work with the object.

The next important point: the constant. Many of the features used as arguments to the numerical codes, and it is not always easy to remember, what code is needed to achieve the desired effect or the desired mode. So you can code instead of entering the names of constants (the list of constants and their numeric values can be seen in the Global Game Settings tab, the constants). The TACIT tradition constants Xtreme3D look like: tmmCubeMapReflection. Lowercase letters at the beginning (TMM) indicate the property, which includes the constant. In this case, The TextureMappingMode and function, feeding it - [MaterialSetTextureMappingMode](#)('material',tmm), where instead of tmm populates the desired tmm-constant.

Remember that constants are part of the Game Maker/GML and to the Xtreme3D.dll files they have.

To start you will need a file *.gm6 (or *.gmk for Game Maker 7), with a ready set of functions and constants Xtreme3D. The file you can take from the official

distribution engine. In preparation for the work of the sufficiently removed from all objects. Let's call it conditionally project.gm6. Copy it to a separate folder and add to the same xtreme files3d.dll and ode.dll.

Open the project.gm6. Create a new object of Game Maker and name it o engine. Add an event Create and drag the action Execute a piece of code with the tab Control. If you have already worked with GML, no problems. If not, we strongly recommend you to leave until the Xtreme3D and explore the language on the built-in graphics Game Maker.


















The following code loads the functions from the library of xtreme3d.dll into memory and starts the operation of the engine:

```
Dll_init('xtreme3d.dll');  
EngineCreate();
```

We go further:


```
View = ViewerCreate(window handle, 0, 0, 640, 480);  
ViewerSetLighting(view, 1);
```

In order to observe anything in the window with the game, you will need a View (Viewer). A view is a rectangle, where the scene rendering Xtreme3D. All that outside this rectangle, "belongs" built-in graphics Game Maker. We have created a kind of resolution of 640x480, the size of the window, so that the Graphics Game Maker and will not be seen. The position of our species on screen - (0.0). This is essentially a coordinate of the upper left corner of the view, on the upper-left corner of the window.

Also in the function of the [ViewerCreate](#) transmitted window handle()function GML, which returns the id of the main window of the game. Thus, the appearance will be "tied" to the window of the game Game Maker that we need. Strangely enough, the view - it is also an object, so when creating we       its id in the variable, in our case, the view. We can use the ID of the Kind to change its properties. At the moment we are interested only in one thing - the use of light ([ViewerSetLighting](#)). If you turn off the lights (0), all objects will look flat and           . Therefore, we include (1). However, in order that the lights worked, you have to create the light sources:

```
Light = LightCreate(lsOmni, 0);
```

[ObjectSetPosition](#)(light, 0, 18, 0);

[LightCreate function](#) creates a light source and returns its id as the light is also an object. In the Xtreme3D there are three types of light sources - spot (constant lsOmni) aimed (lsSpot) and parallel (lsParallel). Dot emits light equally in all directions (as, for example, the lamp), aimed shines within the cone (as a flashlight), parallel emits parallel beams in the direction of one axis (simulation of sunlight). We can assign to the light source of the parent, but, since there is no scenic sites we have not yet,  instead of parents 0. Creating a point light source, you can clarify its position in space - the point (0,18,0).

We still do not see that as the light had nothing to cover. Create a simple visible object. But before that you need to create the root objects in our scene:

```
Global.back = DummycubeCreate(0);  
Global.scene = DummycubeCreate(0);  
Global.front = DummycubeCreate(0);
```

DummycubeCreate function creates a mannequin (Dummycube) and returns its ID. The object, this fun name, plays an important role in the formation of the hierarchy. The dummy is not visible, it is the object of a Ghost. But, at the same time, it has all the usual properties of objects, which we considered in the previous chapter - coordinates in space, the Direction vectors, Up, Left, and so on. You can freely move, rotate and scale. The dummy can have parents and descendants. In this case, we created three dummy root. The root - because it is above them in the hierarchy of the nothing will be. All of the other scenic objects will be the descendants of these three Mannequins:

Global.back - a parent for objects in the background (sky, background, etc.)

Global.scene - a parent for objects in the scenic plan (all 3D objects)

Global.front - a parent for objects on the screen (sprites, text, etc.)







It is important to keep this to create mannequins - first, then the scene, then the screen. This is necessary so that the engine can render the objects in the correct order. This procedure is called sorting: all of the objects in the redrawn in the order in which have been established, they themselves or their parents.

Create the first object of scenic plan - plane:















```
Plane = PlaneCreate(0,64,64,8,8,global.scene);
```

[ObjectPitch](#)(plane,90);

The plane is one of the entities of simple geometrical bodies, which are generated by the engine. [PlaneCreate function](#) creates a plane and returns its ID. Let's look at the arguments:

0 - Determines whether to submit to the plane of one square (abbreviated - ), or split into several; us for the beautiful lights just a few, so point 0;

64,64 - the size of the plane;

8,8 - the number of . In total, the plane will be broken up into $8 * 8 = 64$ ;

Global.scene - parent.

We have created a plane by default, vertical, therefore have to rotate 90 degrees on the X-axis. If you remember, turn on the X-axis is called the pitch, so we need the function [ObjectPitch](#).

We still do not see that as not created a camera. The camera is also object, invisible, as well as the dummy. Used for the projection of the 3d scene on the plane of the screen, rather, on the plane of the view. The projection is carried out from the standpoint of camera position in the direction of the vector Direction of the camera. Simply put, which looks at the camera, we see, as in real life.

CamPos = [DummycubeCreate](#)(**global.scene**);

[ObjectSetPosition](#)(camPos, 0, 10, 0);

Camera=[CameraCreate](#)(camPos);

[ViewerSetCamera](#)(view camera);

Before the creation of the cameras we created for her parents - one of the dummy. This is to ensure that the camera itself can rotate freely, and its movement controlled through this mannequin.

[ViewerSetCamera function](#) indicates the mean, what kind of camera used to transfer images.

As you can see, until all is simple enough. Only the last:

Set_automatic_draw(0);

This function we disable the automatic graphics drawing Game Maker - all the same kind of it completely closes, it makes no sense to spend system resources


on its processing.

The scene we created is to make it work. Add an event Step and add the following code:

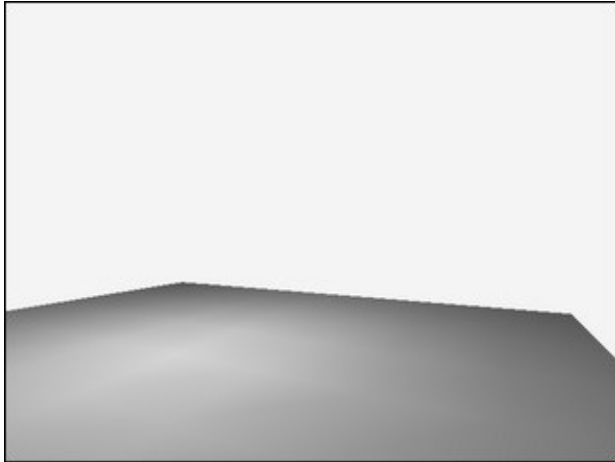
```
If the keyboard_check(vk_left) ObjectTurn(camPos,-2);  
If the keyboard_check(vk_right) ObjectTurn(camPos,2);  
If the keyboard_check(vk_up) ObjectMove(camPos,-1);  
If the keyboard_check(vk_down) ObjectMove(camPos,1);
```

Now, when the user presses, say, "Up", the camera will move forward (respectively, and for the key "Back"). To swivel the camera uses the keys "Left" and "right". Rotate the Y-axis (Turn), and therefore uses the [ObjectTurn](#). Note that the transformation we use not the camera, and its parent. Thanks to this then we will be able to rotate the camera using the mouse.

```
Update(1.0/room_speed);  
ViewerRender(view);
```

These two functions should be cause, otherwise the engine will be "paralyzed". [Update](#) updates the status of the objects in the scene, [ViewerRender](#) commits the drawing of the specified type. In the function of the [Update](#) should be referred to the step time to update the animation. It is measured in seconds and can be equal to the time between two frames of the rendering. Usually in the Game Maker this time is limited in the settings of rooms - is set by the so-called "speed rooms", the maximum manpower frequency, measured in frames per second. It's usually set The  60 - This value corresponds to the frequency of updating the monitor. We can calculate the time interval between frames, dividing the unit (1 second) on this value.

That is all! You can now put our object o engine in the room and run.



Congratulations, you have created its first working program Xtreme3D! Here is the complete source code:

In the event Create:

```
Dll_init();  
EngineCreate(window handle());  
View ViewerCreate =(0, 0, 640, 480);  
ViewerSetLighting(view, 1);  
Light=LightCreate(lsOmni, 0);  
ObjectSetPosition(light, 0, 18, 0);  
Global.back = DummycubeCreate(0);  
Global.scene = DummycubeCreate(0);  
Global.front = DummycubeCreate(0);  
Plane PlaneCreate =(0, 64, 64, 8, 8, global.scene);  
ObjectPitch(plane, 90);  
CamPos = DummycubeCreate(global.scene);  
ObjectSetPosition(camPos, 0, 10, 0);  
Camera = CameraCreate(camPos);  
ViewerSetCamera(view camera);  
Set_automatic_draw(0);
```

In the event of the Step:

```
If the keyboard_check(vk_left) ObjectTurn(camPos, -2);  
If the keyboard_check(vk_right) ObjectTurn(camPos, 2);  
If the keyboard_check(vk_up) ObjectMove(camPos, -1);
```

```
If the keyboard_check(vk_down) ObjectMove(camPos, 1);  
Update();  
ViewerRender(view);
```


Lesson 3

The hierarchy of objects

Level: Beginner

Version of the Xtreme3D: 3.0.x

Author: Gecko

The concept of hierarchy we have already met, but until now we have not viewed it in practice. Many who are not familiar with the approach of the objects, and do not know what a huge savings of time and effort here. The hierarchy allows without any work to do that too difficult or impossible without its use. It comes to the specifics of the movements of objects in some special cases.

Imagine, for example, a situation: it is necessary to simulate a simple star system - the sun and rotating around the planet. Around the world, in turn, rotates the satellite. For simplicity, we will still think in two-dimensional space. How can I do?

Let the Sun - the sun, the Planet - the planet, Moon - the satellite. Each object has two coordinates are X and Y, as well as the angle of rotation around its axis - A. Then (in the $\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond$)

$$\text{Sun.X} = 0$$

$$\text{Sun.Y} = 0$$

$$\text{Planet.X} = \text{Sun.X} + \cos(\text{Sun.A}) * 10$$

$$\text{Planet.Y} = \text{Sun.Y} + \sin(\text{Sun.A}) * 10$$

Taking into account that the distance between the sun and the planet is 10 conditional units. When turning the sun around its axis, the planet will rotate around it, moving the coordinates calculated from the rotation angle of the sun and the distances to be searched. Now it is easy to likewise calculate and coordinates of the Satellite:

$$\text{Moon.X} = \text{Planet.X} + \cos(\text{Planet.A}) * 2$$

$$\text{Moon.Y} = \text{Planet.Y} + \sin(\text{Planet.A}) * 2$$

Manually But this is not always convenient. Especially, if the system is not three object as well, for example, all ten. Or the location of objects varies periodically (for example, the satellite is lifting from one planet and goes to the other). Wise will automate the process by entering for each object in the property of the parent (Parent):

$$\begin{aligned} \text{Planet.Parent} &= \text{Sun} \\ \text{Moon.Parent} &= \text{Planet} \end{aligned}$$

And update the coordinates of the objects the same for all formula:

$$\begin{aligned} \text{Object.X} &= \text{Object.Parent.X} + \cos(\text{object.Parent.A}) * 2 \\ \text{Object.Y} &= \text{Object.Parent.Y} + \sin(\text{object.Parent.A}) * 2 \end{aligned}$$

And is the simplest hierarchy.

With 2D graphics all is relatively simple. But what about the 3d? In the 3D graphics in addition to the sinuses and \cos used vectors and matrices. Operations with them quite difficult and extremely difficult for understanding a newcomer. In addition, all too often, to carry out such operations at the level of the GML irrational: for the storage arrays under the matrix would require more memory and mathematical operations with them will reduce the FPS. But not all so terribly. Xtreme3D assumes all demanding computing, executing them at the level of the machine code, so its hierarchy will work much faster and more accurately than the written manually on the GML.

When using the built-in hierarchy of the Xtreme3D all work is directed by the parents to objects. The trick is that the descendant inherits the coordinate system of the parent. For example, the coordinates of the parent (X, Y, Z) are the coordinates of the Center, on which the count their own coordinates its descendant (X+x, y+Y, Z+z). The descendant, in turn, sends its own coordinates to their descendants, and so on. Own object coordinates are called local.

The coordinate system may be transformed displacement, rotation, or scale. Turning the Local coordinate system of the parent causes change of direction of the axes in the inherited coordinate system, the descendant, which automatically leads to its rotation in space. If, at the time of the rotation of a descendant was at

some distance from the center of the inherited them coordinate system, it will look like the rotation of the descendant of around its parent. Just as in our example!

To establish a system of sun and planets in our case it is sufficient to write something like this:

```
Sun=SphereCreate(4, 24, 24, global.scene);  
ObjectSetPosition(sun, 0, 0, 0).  
Planet=SphereCreate(1, 24, 24, sun);  
ObjectSetPosition(planet, 0, 0, 10);  
Moon=SphereCreate(0.5, 24, 24, planet);  
ObjectSetPosition(moon, 0, 0, 2);
```

[SphereCreate function](#) creates a sphere. You must specify its radius, as well as the number of meridian and parallels. Our Sun radius is equal to 4, the planet - 1, the satellites - 0.5. The meridians and parallels (slices, stacks) divide the sphere into squares, the number of which determines the quality of the appearance of the sphere. Usually it is sufficient to point 24 of the meridian and 24 parallel.

Now you can in the event Step turn, the sun and the planet:

```
ObjectTurn(sun, 2);  
ObjectTurn(planet ,6);
```

...And observe the manifestation of one of the most important properties of the object hierarchy. A proper use of these properties is the main task of working with the Xtreme3D. This kind of manifestations can be observed not only in space, but at every step, so it is important to have an effective means of modeling.

Lesson 4

The camera from the first person

Level: Beginner

Version of the Xtreme3D: 3.0.x

Author: Gecko

In Lesson 3 we considered the simplest version of the camera from the first person - it was managed by the arrow. In the meantime, the absolute most modern games in this case management is used with the mouse. Let's look at how to implement his means Xtreme3D.

To begin with, we create the parent of the dummy for the camera - camPos. We will be moving, not the camera as well.

```
CamPos = DummycubeCreate(global.scene);  
ObjectSetPosition(camPos, 0, 2, 0);  
Camera = CameraCreate(camPos);  
ViewerSetCamera(view1, the camera);
```

The fact is that the camera should only move in the xz plane - in other words, should not "fly" through the air. We will rotate the object of camPos in the Y-axis, when the user will unseat the mouse horizontally - thus, it will be possible to manage the direction of movement. Click on the offset in the vertical direction will cause the local turn the camera object on the X-axis - this way, the user will be able to look up and down, but this does not affect the direction of movement, because the camera inherits the movement from camPos.

Declare the following variables:

```
CenterX = display_get_width()/2;  
CenterY = display_get_height()/2;
```

It is the coordinates of the center of the screen. We will read the offset click on this point and then return it to the cursor.

You can also immediately put the cursor in the center of the screen, to the beginning of the game camera watched strictly forward:

```
Display_mouse_set(centerX and centerY);
```

Now go to the event Step. The following code calculates the offset of mouse cursor on the center of the screen and turns the camPos and camera on the corners, deltaX and deltaY:

```
DeltaX = (centerX - display_mouse_get_x()) / 3;  
DeltaY = (centerY - display_mouse_get_y()) / 3;  
ObjectRotate(camera, DeltaY, 0, 0).  
ObjectRotate(camPos, 0, -deltaX, 0);  
Display_mouse_set(centerX and centerY);
```

Left to realize the movement. We will use the standard for games from the first person the layout of the WASD:

```
Dt = 1.0 / room_speed;  
If the keyboard_check(ord('W')) ObjectMove(camPos, -10 * dt);  
If the keyboard_check(ord('A')) ObjectStrafe(camPos, 10 * dt);  
If the keyboard_check(ord('D')) ObjectStrafe(camPos, -10 * dt);  
If the keyboard_check(ord('S')) ObjectMove(camPos, 10 * dt);
```

The meaning of the multiplying by dt in the following. If moving objects with a fixed speed, their actual speed will be tied to human frequency of application. That is, for example, if we move the selected object 10 units for the frame, the speed at 60 FPS will be equal to $10 * 60 = 600$ units per second. At 120 FPS, respectively - $10 * 120 = 1200$. In the end, the object will move faster or slower, depending on the FPS. This is not what we need, so you need to set the speed of the other values, and not attached to frame. For example, in units per second. Consequently, the manpower speed will be equal to the V / FPS , where V is the speed. We simply explore how an object should move in one shot, if the second it moves to the V units. Thus, the object will move with the correct speed when any personnel frequency.

Not to clutter the code ticks (division, as it is known, relatively slow operation), we instead, multiply the speed of $1 / FPS$ - This value can be calculated only

once. It is also referred to as the step time (this step time should refer to the function of the Update, as mentioned in lesson 2). In the Game Maker 8 hr speed (FPS) usually is fixed and is set in the settings of the room (Room speed). It can be set equal to 60 or 120.

Lesson 5

The materials library

Level: Beginner


Version of the Xtreme3D: 3.0.x


Author: Gecko

One of the most remarkable features of Xtreme3D - use of library materials. The material is a set of parameters that define the appearance of the object. This set consists of the values of color, clarity, texture, the type of mixing, and so on. The concept of the material in the Xtreme3D is much more fundamental than in many other cursors. Very often under the material means the texture, but Xtreme3D material may not have texture. In other cases, the system of materials is poor or incomplete, which cannot be said about the Xtreme3D.

The material is created once and can be applied to any number of objects, supporting materials. Change the settings of the material touches all the objects using this material. The principle gives tangible benefits (saving memory, the amount of code, time and labor programr), although it may cause some inconvenience (for example, if you want to change the parameters of the material referred to only one particular object). The material in the Xtreme3D has a unique name, on which the change settings. The name of the specified string value, for example "mGround".

The library material (Material Library) is called the special design that contains a list of the materials. The library can be active or inactive. In the first case, to which you can add content, configure it , and applied to the objects. In the second case, it is impossible. Since different libraries can contain materials with the same name, cannot be made active, two or more of the library at the same time.

The meaning in the use of multiple libraries of materials is the ability to determine for them the individual path to . These paths are taken into account when loading models, using the external texture files. By default, the libraries are looking for texture in the working directory of the game.

Of course, store them there - not the best idea. Undoubtedly, it is better to organize them in separate folders. For example, if you have three folders with three different models of levels, it will be possible to create for each of them the library materials and assign them the path to  corresponding to the desired folders.

There is a huge amount and variety of settings of materials. In addition, materials allow to apply to them shaders, "photograph" in them pictures from the screen or with the camera, as well as store data although, for example, masks and maps of the heights. Cover in one article all the opportunities available to the system Xtreme3D, is unreal. Therefore, we consider only the most basic: color, texture and a few others.

Is First created and activated the materials library:

```
Matlib=MaterialLibraryCreate();  
MaterialLibraryActivate(matlib);
```

Now you can create the materials:

```
MaterialCreate('mTexture', 'texture.jpg');
```

This function at the same time, creates the material and assigns it the texture from the file. Xtreme3D supports BMP, JPG, PNG, TGA, DDS. It is recommended to use the textures with the party equal to 128, 256, 512 and other degrees of deuce. The texture does not necessarily have to be a square.

You can create a material without texture, just leaving the name of the file is empty:

```
MaterialCreate('mColor', "");
```


In this case, you can specify the color of the material. The easiest way to do this is the function of

```
MaterialSetDiffuseColor('mColor', c_red, 1);
```

The color you can pass the built-in constants GML (`c_red`, `c_yellow`, `c_green`, etc.), the functions `make_color_rgb(r, g, b)` or `make_color_hsv(h, s, v)` for

models of color RGB and HSV, respectively, as well as in the reverse hexadecimal format, for example, \$0000FF means red.

In addition to the colors, The [MaterialSetDiffuseColor](#) sets the value of transparency - Alpha - lies in the range from 0 to 1. Alpha in our case is equal to 1 (Full Opacity).

Strictly speaking, The [MaterialSetDiffuseColor](#) specifies only one of the components of the color of the material as well as all their four - Ambient, diffuse, specular and Emission. This separation is due to the fact that the surface areas with different levels of ambient light can have different color. Ambient determines the overall shade of material that is independent of the Lighting (the color of the shadow side), Diffuse - the color of the lit side, Specular - the color of the , Emission - the color of the simulation of luminous. Moreover, the light sources also have their own components of ambient, diffuse and Specular, and this makes the color of objects even more complex and diverse.

You can disable for material lighting (and, at the same time, the influence of the mist, which will be discussed later):

```
MaterialSetOptions('mColor', 1, 0);
```

The first parameter, in this case equal to 1, is responsible for the effect of fog, the second - for the impact of lighting.

Apply the created material to an object is very simple:

```
ObjectSetMaterial(object, 'mTexture');
```

By the way, back to the material with a texture. With this texture can do amazing things! For example, changing the mode of projection on the sphere will make a material similar to reflection on the metal:

```
MaterialSetTextureMappingMode('mTexture', tmmSphere);
```

And to repeat many times the texture on the surface of the object, change its size:

[MaterialSetTextureScale](#)('mTexture', 10, 10).

This texture is repeated ten times.

As an independent work try to create and apply the materials on the planet from the previous lesson about the hierarchy. As the textures can use real maps the surface of the Earth and the Moon. And the sun can be left without textures and do a simple yellow. Remember as well as creates a light source and make it the child of the sun that it radiated light (in this case, it is worth to disable lighting for his material).

Lesson 6

The entities

Level: Beginner

Version of the Xtreme3D: 3.0.x

Author: Gecko

Primitives are commonly referred to as either the simplest objects that can draw a GPU (point, segment, triangle), or geometric body, embedded in the graphical engine. In the Xtreme3D uses the second meaning of this term. The solids-
◆◆◆◆◆◆◆◆◆◆ include plane (plane), cube (cube), the sphere (sphere), the cylinder (cylinder), cone (cone), a hollow cylinder (annulus), Thor (torus), the disk (disk), a truncated pyramid (frustrum), the dodecahedron dodecahedron (), ◆◆◆◆◆◆◆◆◆◆ (icosahedron) and the kettle Utah (teapot). In the previous lessons we have already created some of them. Let's get acquainted with primitives closer.

Plane. The rectangular plane. In the Xtreme3D a plane can be represented by one rectangle (◆◆◆◆◆◆◆◆◆◆), or divided into a grid of ◆◆◆◆◆◆◆◆◆◆. The second option is preferable, if you create a great plane of Earth, as in this case, it turns out a better vertex lighting (however, when using pixel-by-pixel detail light plane of special significance, as a rule, does not have). Note that the number of ◆◆◆◆◆◆◆◆◆◆ affects the recurrence of the textures on the plane (One quad is one ◆◆◆◆◆ textures). The plane creates a function of [PlaneCreate](#).

Cube. The cube. Strictly speaking, this is not necessarily a cube and any rectangular parallelepiped. A function of the [CubeCreate](#).

Sphere. The scope. A function of the [SphereCreate](#).

Cylinder. The cylinder. A function of the [CylinderCreate](#).


Cone. The cone. A function of the [ConeCreate](#).



Annulus. A hollow cylinder (ring). A function of the [AnnulusCreate](#).

Torus. Thor (the body which resembles a donut). A function of the [TorusCreate](#).

Disk. The disc. A function of the [DiskCreate](#).

Frustum. A truncated pyramid. A function of the [FrustrumCreate](#).

Dodecahedron. A dodecahedron - a polyhedron, compiled from 12 correct . A function of the [DodecahedronCreate](#).

Icosahedron.  - a polyhedron, compiled from 20  triangles. A function of the [IcosahedronCreate](#).

Teapot. The kettle Utah. You can read more about this model [in the glossary](#). A function of the [TeapotCreate](#).









Lesson 7
































Download the file from the

Level: Beginner

Version of the Xtreme3D: 3.0.x




























Author: Gecko

The entities is, of course, is good, but to create a full-fledged games their is not enough. The most favorite occupation of everyone who starts to get acquainted with the 3D- - clearly, loading the engine own models produced in third-party editor. Xtreme3D supports loading of models of formats 3DS (3D Studio), OBJ (Maya), (LWO Lightwave), BSP (Quake), MS3D (Milkshape), B3D (Blitz3D), the LOD LODka (3D) and many others.

Static (i.e., The ) model in the Xtreme3D is loaded in a special object - Freeform:

```
Model = FreeformCreate('model.3ds', matlib, matlib, global.scene);
```

The second and third parameters of this function are responsible for library materials, which should be used, respectively, for the usual textures and lighting card model. A good form is the creation of a separate library of materials for each object in the Freeform to guaranteed to avoid conflict names of materials. In this case, for ease of use the same library.

If a file with the model includes information about the textures, Xtreme3D will automatically attempt to download them. The only question is exactly where the engine will search for these textures. By default - in a working directory of the game. But to keep them there - not the best idea. It is much more convenient to put texture in any folder , the textures. Then we have to specify the active library of materials that the textures to be found there:

```
MaterialLibrarySetTexturePaths(matlib, 'textures');
```

The materials will be uploaded to the library under those names, which have been specified in the 3D editor. Using these names, you can adjust the characteristics of the material. This makes it possible to partially change the

appearance of the model. For example, imagine that you have downloaded the model of the vehicle. You can change the color of the housing or the salon, without affecting other parts to make transparent windows tinted, add the effects of reflections on wheels and so on.


However, as soon as you want to go, you will find that it is impossible to turn the wheel. This is not surprising: they are part of the same Freeform. Therefore, in such a situation, it should be split model for its constituents - the dwelling was:

```
Car = DummycubeCreate(global.scene);  
FreeformToFreeforms(model, car);
```

We create a dummy, which will be a parent for all parts of the vehicle, and we break the model into separate independent Freeform. Note that this operation is possible only if the parts of the vehicle (bodywork, wheels, doors, trunk, etc.) constitute a separate dwelling was - not all formats supported models such a division.

The original Freeform we have already is not needed, and we delete:

```
ObjectDestroy(model);
```

To manage the created objects, we need to get their ids. You can do this function of the [ObjectGetChild](#). This does not superfluous will know how many total  was in the original model. Suppose that the five - four wheels and bodywork:

```
Car_body = ObjectGetChild(map, 0);  
Car_wheel_1 = ObjectGetChild(map, 1);  
Car_wheel_2 = ObjectGetChild(map, 2);  
Car_wheel_3 = ObjectGetChild(map, 3);  
Car_wheel_4 = ObjectGetChild(map, 4);
```

Remember that the count is zero, so that the first descendant - zero.

Thus, we have received a new hierarchy, the structure is completely identical to the original model. You can now rotate the wheel:

```
ObjectPitch(car_wheel1, 3);  
ObjectPitch(car_wheel2, 3);  
ObjectPitch(car_wheel3, 3);
```

[ObjectPitch](#)(car_wheel4, 3);

[ObjectPitch](#) instead, you can use the [ObjectRoll](#) - depending on where "looks" the vehicle bodywork: along the Z-axis or X.

You can also replace some of the models on the other. For example, if you create a model with two wheels, you can hide some of the wheel, leaving the other, and vice versa. Or make several versions of the body with varying degrees of damage to dynamically switch between them, when the vehicle runs in an obstacle. And yet, you can use the coordinates of parts in the space, to create in them a variety of special effects - smoke or flame. Or, for example, if you have created a tank, you can rotate the tower and shoot from the muzzles of projectiles in the appropriate direction. Xtreme3D allows you to do with models of anything!

8. Вершинная animationLesson 8

animation

Level: Beginner

Version of the Xtreme3D: 3.0.x

Author: Gecko

The Object Freeform is intended mainly for inanimate objects. This is usually the elements of decoration, vehicles, various interactive objects, and so on. If we want to populate our virtual world of living creatures, we cannot do without the Actor objects. The name speaks for itself: the actor is a living character. In the Xtreme3D actors represent the animated model. And the animation, as is well known, is of two types - **vertex** and skeletal. In this lesson, we look at The **vertex** animation.

Vertex (or **vertex**) Animation is characterized by the fact that for the formation of an animation sequence of the slider moves each vertex of the model from one position to another. This type of animation was first applied in the Quake, and since then the formats of the models in the Quake (MD2, MD3) have become a standard in all **3D** cursors. Xtreme3D provides full support for MD2 and MD3. The difference between them lies in the fact that the MD2 stores all model entirely in one file, and the MD3 - in three (Head, torso and legs). With the help of special matrices the torso is synchronized with the feet, and the head - with the torso. This was done in order to animate the torso and legs separately. For example, during the shooting character can both run and walk slowly, and even simply stand in place.

In this lesson, we look at The **vertex** animation with the format of the MD2. The actor from the MD2 is created as follows:

```
Actor = ActorCreate('model.md2', matlib, matlib, global.scene);
```

Sometimes after loading the model is that it incorrectly rotated. This is because different editors axis direction is interpreted in different ways. Usually "swap" the Y axis and Z model peaks are recorded so that its vector of Up is directed along the Z axis (in DirectX applications, this means "up"), and, since the

direction of the "up" in Xtreme3D meets the Y axis, but not Z, it turns out that the model rotated 90 degrees on the X-axis. We can fix this misunderstanding in several ways. The most simple - just turn her back:

```
ObjectPitch(actor, 90);
```

But in some cases this is not enough. Turning the model, we also turn its Local coordinate system. This means that the vector Direction now indicates along the Y-axis and Z is not as it should be. If we now move the model using [ObjectMove](#), she will move up and not forward. You can, of course, instead of [ObjectMove](#) use [ObjectStrafe](#), but this will make the program less neat and tangle so long. Much better than the first place the actor in the descendants of the dummy and then rotate. And, accordingly, to move to use the dummy, not the actor. The code will be as follows:

```
Player = DummysceneCreate(global.scene);  
Actor = ActorCreate('model.md2', matlib, matlib, player);  
ObjectPitch(actor, 90);
```

The format of the MD2 provides for the separation of all frames of animation in separate groups. This is done in order to separate the, say, the animation runs from animation jump. By default, the Xtreme3D plays all frames one after the other, not paying attention to this division. But we can at any time switch to the desired animation:

```
ActorSwitchToAnimation(actor, 1, false);
```

And then will be played only group training under number 1. The third parameter of this function is responsible for the smooth change of animation: if set to true, the change will be gradual.

Approximately the same makes the function, indicating the range of frames for playback:

```
ActorSetAnimationRange(actor, 10, 20);
```

It is not difficult to guess that will be lost only the period between the tenth and twentieth frames. However, these two functions have one important distinction. [ActorSwitchToAnimation](#) every time the call switches to play the first frame of the target group, and [ActorSetAnimationRange](#) does not do this (if the range is

already playing). [ActorSetAnimationRange](#) therefore can be called repeatedly - for example, inside the loop that in some situations turns out to be very useful.

By default, the animation is reproduced cycle - that is, when it reaches the last frame, play starts again. In most cases this is what you need (for example, the animation of walking or jogging is always fixated). But we can specify and other playback mode:

[ActorSetAnimationMode](#)(actor, aam);

Instead of the AAM uses one of the following constants:

AamNone - Animation cannot be reproduced;

AamPlayOnce - Animation is reproduced once and stops when it reaches the end frame. This mode is sometimes referred to as the "one shot";

AamLoop - Animation is repeated cyclically (default);

AamBounceForward - Animation is repeated cyclically forward until the final frame, and then in the opposite direction to the initial frame, then again forward, and so on. This mode is sometimes referred to as the "ping-pong".

AamBounceBackward - the same thing, but in the opposite direction.

AamLoopBackward - animation cycle is repeated in the opposite direction.

Finally, there is also the possibility to disable the linear interpolation between frames:

[ActorSetFrameInterpolation](#)(actor,false);

The frames will be Ousting each other abruptly, without a smooth "spill". This can be useful, for example, in the races, where the vehicle bodywork can be deformed - in different frames topmost animations can store different variants of damage.



Lesson 9



Basics of the skeletal animation

Level: Beginner

Version of the Xtreme3D: 3.0.x

Author: Gecko

Sometimes opportunities  animation is not enough. This is the case, basically, games of the genre of action. For example, you may want to "give" their hero into the hands of the weapons or "wear" armor. When using the topmost animations it is impossible (with rare exceptions). In addition, the  animation can require too much memory for storage of frames. Therefore, if you are using the model with a large number of polygons, wise will select the skeletal animation.

Instead of storing the key frames (as in the case of The  animations) for each of the poses of the character, the use of skeletal animation implies one model in the neutral position and a large set of matrices that transform the various parts of the model. These matrices conditionally called bones. To each of the Bones tied group of vertices. One vertex can "belong" to multiple bones immediately, with varying degrees of influence that makes the animation more natural (This property is called the bone ).

For the first time, this technology has been used in the game Half-Life, and Xtreme3D supports the format of the models of Half-Life - SMD. As an alternative to the SMD, also supported by the format of the models of Doom III - the MD5.

To download the models with the skeletal animation is used, the same object Actor. You do not need to also specify, Xtreme3D is able to recognize the type of models and tune to the correct type of animation:

```
Actor = ActorCreate('model.smd', matlib, matlib, global.scene);
```

The peculiarity of the SMD format is that the animation model is stored in a separate file, which also has an extension *.smd. Such files can be several. In


theory, this method allows you to use the same animation files for different models (if they have the same skeleton).

After the creation of the actor should add these files:

```
ActorAddObject(actor, 'animation1.smd');
```

```
ActorAddObject(actor, 'animation2.smd');
```

```
ActorAddObject(actor, 'animation3.smd');
```

When you add the next smd file, to the animations of the actor adds a new group training, which is assigned a sequence number. The count is 1. That is, if we now  to Group 2:

```
ActorSwitchToAnimation(actor, 2, false);
```

...It will be lost animation loaded from a file animation2.smd.

The skeletal animation apply all the features that we considered in the previous lesson.



Lesson 10

The camera from a third person

Level: medium

Version of the Xtreme3D: 3.0.x

Author: Gecko

Many games use the view from a third person, where the camera shows the character "from the back" is, for example, many game genres Action and RPG, 3D-type  Spyro or Crash Bandicoot, Sports simulators, and so on. In this case, as a rule, the camera is not rigidly fixed at a certain distance from the character - it usually moves smoothly, with some delay that adds realism and .

The Xtreme3D similar to the operator to realize only slightly more complicated than the view from first person. The following code creates a hierarchy from the character, which the player will manage, and cameras, which will be for him to follow. As a symbol of the Character uses a simple cubic meters.

The code in the event Create:

```
Camera = CameraCreate(global.scene);  
CameraSetViewDepth(Camera, 800);  
CameraSetFocal(camera, 80);  
ViewerSetCamera(view1, the camera);
```

```
Actor CubeCreate =(1, 1, 1, global.scene);
```

```
Target = DummycubeCreate(actor);  
ObjectSetPosition(target, 0, 1, -4);  
CameraSetTargetObject(camera, actor);
```

Code in the event of the Step:

```
If the keyboard_check(vk_up) ObjectMove(actor, 10 * dt);
```

If the keyboard_check(vk_down) [ObjectMove](#)(actor, -10 * dt);

If the keyboard_check(vk_left) [ObjectTurn](#)(actor, -200 * dt);

If the keyboard_check(vk_right) [ObjectTurn](#)(actor, 200 * dt);

Cx = [ObjectGetAbsolutePosition](#)(camera, 0);

Cy = [ObjectGetAbsolutePosition](#)(camera, 1);

Cz = [ObjectGetAbsolutePosition](#)(camera, 2);

Tx = [ObjectGetAbsolutePosition](#)(target, 0);

Ty = [ObjectGetAbsolutePosition](#)(target, 1);

The Tz = [ObjectGetAbsolutePosition](#)(target, 2);

Dx = tx - cx;

Dy = ty - cy;

Dz = the tz - cz;

[ObjectTranslate](#)(camera, dx * 0.05, dy * 0.05, dz * 0.05);

The logic of the chamber is arranged so that its most long distance the distance from the character - when driving forward (so you can see what is happening around), and the closest - when driving in reverse. When turning the character the camera allows to consider it on the side. Approximately the same admission is used in racing simulations, so that, on the basis of this Code, it can be done and the engine races.

Lesson 11

Collision test

Level: medium

Version of the Xtreme3D: 3.0.x

Author: Gecko

In the games very often, it is required to define the fact of collision between two objects. They can be, for example, the character and the Platform, shell and purpose, and so on. The collision detection is based logic, the shooters, simulations, role-playing games and some of the Strategies. If this does not always want to find the exact intersection of two polygon - enough to test the intersection of limiting their scope (Bounding Sphere) or (Bounding Box). Xtreme3D includes easy-to-use tools that allow you to do this.

Function test collisions in the Xtreme3D begin with "ObjectCheck..." and operate on restricting the spheres and objects, which are calculated using the engine automatically, depending on the volume, which is their geometry. Restricting the (which in these functions are called the Cube) aligned to the local coordinate axes of the object - that is, can rotate with him. Such is often called Oriented Bounding Box, or abbreviated OBB. Functions return true (1) If a crossing, and lie (0) otherwise.

Xtreme3D includes the following function test collisions: [ObjectCheckSphereVsSphere](#), [ObjectCheckSphereVsCube](#), [ObjectCheckCubeVsCube](#), [ObjectCheckCubeVsFace](#), [ObjectCheckFaceVsFace](#). The last two of them operate on the objects of the type of Freeform - accordingly, can detect the crossing of the parallelepiped bounding one object with the polygon model of the other, as well as the intersection of the two models. This test is rather slow, therefore, it is recommended to optimize its use - for example, to carry out an accurate test between the models only if the collision was detected between their limiting areas:

```

If ObjectCheckSphereVsSphere(obj1, obj2)
{
  If ObjectCheckFaceVsFace(obj1, obj2)
  {
    // Do something
  }
}

```

These functions are useful when you need to perform a discrete test - that is, when it can be argued that the objects are moving with small speeds. If the speed of high, and the object in one step of game time flies the distance greater than the size of another object, the discrete checking can easily fail. A universal solution to this problem so far, no, but there are a variety of simplified methods. The easiest method - "throwing rays" (Ray Casting). In the Xtreme3D have enough effective implementation of this method. From the center of the object beam is produced in the direction of the Direction of the object. Then the intersection with this ray checked the target object, one or a few. Thus, it is possible to simulate the movement of the bullets (under the assumption that it is moving with infinite speed) - instantly find the point at which it enters. Using the "throwing rays" can be used to determine the height of the land under the character that is necessary for realization of the jumps. In addition, this method is indispensable for the building of the logic of the interaction of the character with interactive objects and triggers - Imagine, for example, the shooter, RPG or a quest from the first person, where the player can pick up objects and to press on the levers, by clicking on them with the mouse. To do this, you can estimate the distance between the player and the object, and then apply "throwing rays":

```

If ObjectGetDistance(player item) <= 1.0
{
  If ObjectRaycast(player item)
  {
    Hit_x = ObjectGetCollisionPosition(0);
    Hit_y = ObjectGetCollisionPosition(1);
    Hit_z = ObjectGetCollisionPosition(2);
  }
}

```

What is the most pleasant, "throwing rays" in the Xtreme3D is fully compatible

with the objects of Freeform and gives correct results in any transformation of the objects.

12. 2D graphics


Lesson 12

2D graphics

Level: medium

Version of the Xtreme3D: 3.0.x

Author: Gecko

Xtreme3D allows you to draw not only 3D objects, but 2d is a screen text and sprites on screen. On-screen text is displayed on top of the  images and applies for the submission of some textual information in the game: the number of lives or cartridges, different messages, debug data and etc. The on-screen sprites is simply 2D images that can be used to display user interface elements - the sight, icons, the scale of energy, etc. Also you can make the game menu with the background is more preferable than the Create menu via the built-in graphics Game Maker, as in this case, you can draw the menu on top of the 3D-scenes that looks very stylish.

Text objects use special font objects, storing images of text characters, letters, numbers, and punctuation. These images can be set in two ways: to generate from the system of vector fonts in Windows or download from the file. The second is more preferable as you can draw in the graphical editor font size of any color and complexity, with any symbols and in any language, while support for system fonts, severely limited (supported only latin). But the vector fonts, there is one indisputable advantage is the scalability without loss of quality: that is, you can from one and the same font system to generate symbols of different sizes.

The code for the creation of the font and the on-screen text is as follows:

```
Font = WindowsBitmapfontCreate('Arial', 14, 32, 95);  
Text = HUDTextCreate(font, 'Hello, World!', the global.front);
```

Please note that we specify the root object of **global.front** as a parent object text - This ensures that the text will  after the 3d scene.

Created by the text can be modified - ask him the color and transparency, as well as the position and rotation:

```
HUDTextSetColor(text, c_red, 0.5);  
ObjectSetPosition(text, 100, 100, 0);  
HUDTextSetRotation(text, 30.0);
```

The type fonts WindowsBitmapfont is a serious disadvantage: it only supports ANSI encoding. This means that in one application cannot use the symbols of several different alphabets. To solve this problem in the Xtreme3D was added support for the library and FreeType UTF-8 encoding, which allows you to display any characters without restrictions. Using the FreeType you can download ttf-fonts from the files, which is very convenient - you do not have to worry about whether the desired font on the user's system: all the necessary fonts may have shipped with the game.

The creation of the font and the on-screen text using the FreeType looks as follows:

```
Font = TTFontCreate('data/font.ttf', 14);  
Text = HUDTextCreate(font, 'Hello, World!', the global.font);
```

The text string you pass to the [HUDTextCreate](#), must be encoded in UTF-8. Unfortunately, the Game Maker 8 does not support UTF-8 in the built-in editor code, therefore the text containing the symbols of national alphabets should either download the file from the function of the [TextRead](#) or convert the [TextConvertANSIToUTF8](#).

To use the [TTFontCreate](#) place in the folder with the game of the freetype library.dll (look for it in the SDK).

Lesson 13

Shadows in real time

Level: medium

Version of the Xtreme3D: 3.0.x

Author: Gecko

Xtreme3D supports several ways to render shadows. Firstly, the shade (and lighting in general) can be `◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆` and `"◆◆◆◆◆◆◆◆◆◆"` in the texture - this technique is called the map light (Lightmapping). It allows you to get very beautiful and realistic result, but the resulting shade will be static. Accordingly, this technique is only applicable to the stationary objects - for example, the interiors and architecture. In addition, far not all formats of 3D models support the light maps, and not all of the 3D editors they can create.

Secondly, there is the object of the shadow plane (shadow plane). It all looks like a normal entity plane, except that the other objects can throw at him the shadows. The result is a very beautiful, but, unfortunately, the flat shadows of the applicable far not in all situations. For example, they are of little use if your game level consists of the platforms at different heights, or there is no ideal planes (for example, in the case of a realistic landscape). There are, however, a number of genres, where the shadow plane is justified is, for example, a wide variety of sports simulators (football, athletics, mini-golf, bowling, billiards, etc.), flat `◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆`, as well as some of the logical and casual games.

There are also two technology rendering shadow volumes - volume shadow and shadow mapping. The first is an object that defines the volume, inside of which points are in the shade. This method gives a very accurate shadows on any distance, but works quite slowly. More promising is the new technique, appeared in Xtreme3D 3.0 - shadow maps (shadow mapping). You can very quickly render soft shadows - but, the truth, at a limited distance from the camera.

We turn first to the shadow plane. Create it very simply:

```
ShadowTarget = DummycubeCreate(global.scene);  
ShadowPlane = ShadowplaneCreate(20, 20, 10, 10, shadowTarget, light,  
c_black, 0.5, global.scene);
```

Now all I have to do is add those objects, which should drop shadow, in the descendants to The shadowTarget.

To work with the shadow volume slightly more difficult:

```
Sv = ShadowvolumeCreate(global.scene);  
ShadowvolumeAddLight(sv, light);
```

Add the objects to be drop shadow, in the descendants of the sv. Those objects, which should drop shadow added as follows:

```
ShadowvolumeAddOccluder(sv, Obj);
```


Lesson 14

The establishment of the Sky


Level: medium

Version of the Xtreme3D: 3.0.x

Author: Gecko

Many games an important part of the graphics and gameplay is smooth change of day and night (for example, in the RPG and games in the GTA style). In the Xtreme3D This functionality provides an object Skydome, the literal translation means "heavenly dome", "a firmament." This is a spherical form, the stage and changes color depending on the time of day. On the dome in the literal sense of the "suspended" The sun and stars. The sun goes smoothly across the sky: when it is lowered toward the horizon, we are witnessing a sunset. And then the sky darkens and there comes the night. Sparkling stars as the real thing. The only minus - no clouds. They would have to do on their own. It probably was assumed not to deprive the developer to make, for example, The : so that you can as fall to the ground and climb to the clouds and see them from a close distance.

The color of the sky in the skydome is composed of three components: Deep, Haze, the Night and the Sky.

Deep - the color of the so-called nadir - the point opposite the ; it is located under our feet. Usually in real life to see nadir is impossible, the earth obstructs :) But the color of this point is important as it determines how shade mixes up the color of the sky as the care of the horizon line.

Haze - the color of the horizon. Usually corresponds to the color of the mist.







Sky is the color of the zenith. In this color painted the entire heavenly bubble up to the line of the horizon.

Night - the color of the night. When the sun goes below the horizon, this color is gradually filled in all the components of the sky, in addition to the Deep. This is most often black or dark blue, although there may be other options.



Below is the code that creates the sky:





















```
Sky = SkydomeCreate(24, 48, global.back);  
SkydomeSetOptions(sky, true, true);  
ObjectRotate(sky, 90, 0, 0).  
SkydomeSetNightColor(sky, make_color_rgb(0, 0, 180);  
Angle = 0;  
SkydomeSetSunElevation(sky, angle);  
SkydomeAddRandomStars(sky, 50, c_white);
```



The sun moved across the sky, you need every step of time to change the angle at which it is on the horizon. The angle of 90 degrees,       -90 - Nader.

```
SkydomeSetSunElevation(sky, angle);  
Angle = angle of + 1.0 * dt;
```

You can also create a realistic starry sky with the familiar us constellations, although this is not as simple as it may seem. To do this you need to understand the celestial coordinates. In the Xtreme3D the position of the stars in the sky is set during the second equatorial coordinate system, which includes two values - direct ascent (right ascension) and declination (declination). The two values are in degrees, although the Astronomy of the direct ascent traditionally measured in hours, minutes, and seconds (1 hour is equal to $360 / 24 = 15$ degrees). To simplify the translation of these units in degrees, in the Xtreme3D SDK is The RightAscension script(hours, minutes, seconds). There is also a script(Declination degrees, minutes, seconds), with which you can get a float value from the degrees of angular minutes and angular seconds.

Here is an example of the creation of the well-known of the bucket - the seven major stars of the Big Dipper (coordinates i took from Wikipedia):

```
SkydomeAddStar(sky, RightAscension(11, 3, 44), the Declination(61, 45, 0),  
1.79, c_white); //       
SkydomeAddStar(sky, RightAscension(11, 1, 50), Declination(56, 22, 57), 2.37,  
c_white); //       
SkydomeAddStar(sky, RightAscension(11, 53, 50), Declination(53, 41, 41),  
2.44, c_white); //       
SkydomeAddStar(sky, RightAscension(12, 15, 25), the Declination(57, 01, 57),  
3.31, c_white); //     
```

```
SkydomeAddStar(sky, RightAscension(12, 54, 0), the Declination(55, 57, 35),  
1.77, c_white); // Suites  
SkydomeAddStar(sky, RightAscension(13, 23, 55), the Declination(54, 55, 31),  
2.27, c_white); // The   
SkydomeAddStar(sky, RightAscension(13, 47, 32), the Declination(49, 18, 48),  
1.86, c_white); // 
```

It will be more convenient to, of course, to create something like the star directory in the file and read it when loading, creating the stars procedurally.

Lesson 15

The creation of the Landscape

Level: medium

Version of the Xtreme3D: 3.0.x

Author: Gecko

The Landscape (Terrain) is an important part of the games many genres, modeling the situation in the real world is Racing, Strategy, many of the shooter and the various games with the open world. Usually the landscape is not modeled manually and is generated from the so-called card heights - the image, where the dark areas indicate lower height and bright - increase. The generation of the landscape can occur both in the program of the 3D modeling as well as in the game - in the latter case, there is a possibility to optimize the rendering of landscape, dynamically changing the detail depending on the distance from the camera (dynamic LOD). In the Xtreme3D also has support for such technology. In order to render the landscape, you must first download the map heights, in the terminology of the Xtreme3D - HDS (Height Data Source, the source of data about the height):

```
The Hds BmpHDSCreate =('heightmap.bmp');  
BmpHDSSetInfiniteWarp(hds, 0);
```

[BmpHDSSetInfiniteWarp function](#) can make the map heights endlessly looping in all four sides - very handy if you want to make a limitless world.

Now create the landscape is the object of the Terrain:

```
The terrain = TerrainCreate(global.scene);  
TerrainSetHeightData(terrain, hds);  
TerrainSetTileSize(terrain, 32);  
TerrainSetTilesPerTexture(terrain, 8);  
TerrainSetQualityDistance(terrain, 100);  
TerrainSetQualityStyle(terrain, hrsFullGeometry);  
TerrainSetMaxCLodTriangles(terrain, 10000);
```

```
TerrainSetCLodPrecision(terrain, 50);  
TerrainSetOcclusionFrameSkip(terrain, 0);  
TerrainSetOcclusionTessellate(terrain, totTessellateIfVisible);
```

If you run the game at this stage, the landscape is likely to be too high and rotated 90 degrees. This is easy to fix by installing the desired scale in the Z axis and Rotating an object on the X-axis:

```
ObjectSetScale(terrain, 1, 1, 0.1);  
ObjectRotate(terrain, 90, 0, 0).
```

Separate the words deserved the overlay textures on the landscape. This can be done in many different ways, I propose the following: first, the texture of the material (diffuse) will be tight on the whole landscape, and the second (the texture of detail) will be repeated many times with the magnitude of the overlying the first in the modulate (i.e., by changing the brightness of the previous). Thus, there is an illusion that the landscape uses huge detailed texture.

```
MaterialCreate('mTerrain', 'the terrain-diffuse.jpg');  
MaterialSetOptions('mTerrain', false, true);  
MaterialCreate('detmap', 'the terrain detail.jpg');  
MaterialSetTextureScale('detmap', 100, 100);  
MaterialSetSecondTexture('mTerrain', 'detmap');  
ObjectSetMaterial(terrain, 'mTerrain');
```

Please note that we switched lighting for landscape material - the fact is that the dynamic LOD does not allows you to define normal for the vertices (since the sets of vertices are constantly changing), which is necessary for the correct lights polygons. Therefore, for the landscape should use static lighting - the map light, combined with the diffuse texture.

Still another challenge: movement of the character on the landscape. This is easily done with the help of the designated functions - [TerrainGetHeightAtObjectPosition](#), which returns the height of the land in the point, which coincides with the absolute position of the object:

```
ObjectSetPositionY(camPos, TerrainGetHeightAtObjectPosition(terrain,  
camPos) + 1);
```


Lesson 16

The creation of water

Level: medium

Version of the Xtreme3D: 3.0.x

Author: Gecko

The realistic animated water is a kind of hallmark Xtreme3D. This is really beautiful, special allows you to achieve a high degree of realism of the scene. However, the use of water must be carefully, because the calculations associated with it, quite .

The water in the Xtreme3D is a flat surface, divided into a certain number of squares, on which there are concentric wave disturbances - you get the effect of "rain". Also supported are linear waves, as in the ocean - they are parallel, calculated on the sinusoid and uniformly moving across the surface of the water. In and of itself the water - only the geometry and any properties of the fluid, in addition to the waves, do not possess. Therefore, all other effects, enhancing the impression of water (reflection, sprays, physics of floating objects, etc.) you have to write on their own.

The Code creation of water looks as follows:


```
Water = WaterCreate(global.scene);  
WaterSetResolution(water, 128);  
WaterSetRainTimeInterval(water, 1000);  
WaterSetRainForce(water, 1000);  
WaterSetViscosity(water, 0.95);  
WaterSetElastic(water, 10);  
ObjectSetPosition(water, 0, 2.5, 0);  
ObjectSetScale(water, 1000, 1000, 1000);
```

The Object Water has several important properties:

Resolution (resolution)

The number of polygons on the side of the grid. The geometry of the water is constructed of squares on the grid, so the total number of polygons is R^2 , where R is the value of the Resolution. It is clear that the higher the value, the higher the quality of water, and, accordingly, the speed of its work. The default value is: 64.


Rain time interval (the time interval of rain)

The effect of "Rain", when the surface of the , creating in random places mesh wave disturbances. This option specifies the length of the pause in milliseconds between two disturbances. The default value is 500. The maximum value: 1000000. The minimum value is 0 (no rain).

Rain Force (force of rain)

The intensity of the disturbances; less than this value, the faster the waves of "fade". The default value is 5000. The maximum value: 1000000. The minimum value: 0.

Viscosity (viscosity)

The amplitude of the perturbation theory; in fact, the maximum height of the waves: the less the value, The  liquid. The default value is 0.99. The maximum value: 1. The minimum value: 0.

Elastic (elasticity)

Speed of the perturbations. In the real world this property depends on the density of the substance (for example, the density of the mercury is much higher than that of water, so waves spread faster). The default value is: 10.

Another point: for the creation of water requires a mask that defines the shape of the surface. The Mask in this case - material with a monochrome image, where the black dots indicate lack of water, white - the presence of. Thus, we can create, for example, round the pool.

[MaterialCreate](#)('mMask', 'watermask.bmp');

[WaterSetMask](#)(water, 'mMask');

Lesson 17

A system of particles

Level: medium

Version of the Xtreme3D: 3.0.x

Author: Gecko

Particles (particles) is a small, simple in form of moving objects, a lot of which simulates various complex dynamic substance, for example, fire, smoke, fireworks, etc. As the particles in the game engines are typically used billboards - toward the camera with the rectangle texture. In the Xtreme3D there are two different systems of particles - FireFX and ThorFX. FireFX - the system of particles, modeling fire (although you can use it to recreate and various other effects). ThorFX is designed for the simulation of lightning and various kinds of electric discharges.

In the games is typically used many similar systems of particles. For example, in a cave on the walls can hang torches, and each will burn the fire, made with the help of particles. In this case, the easier it is not to create a separate system for each of the spray, and just draw a one and the same system several times in different positions. Specifically for such a situation, a manager of The FireFX. Manager - this is something like "server" running all the calculations related to the effect. A separate system of FireFX is "customers", using settings specified for the manager. You simply create a manager, configure it as you want, and then add any number of systems in all the right places you in the scene. The changes made to the manager settings will automatically affect all their reporting systems.

First, create The FireFX manager:

```
Firefx FireFXManagerCreate =();  
FireFXSetParticleSize(firefx, 0.3);  
FireFXSetRadius(firefx, 0.1);  
FireFXSetBurst(firefx, 2.0);  
FireFXSetDensity(firefx, 1);
```

```
FireFXSetLife(firefx, 1);  
FireFXSetColor(firefx, c_yellow, 1.0, c_red, 0.0);
```

You will now add the effect of fire any objects in any quantity:

```
Fireobj1 = DummycubeCreate(global.scene1);  
ObjectSetPosition(fireobj1, 2, 0, 0).  
FireFXCreate(firefx, fireobj1);
```

```
Fireobj2 = DummycubeCreate(global.scene1);  
ObjectSetPosition(fireobj2, 2, 0, 0).  
FireFXCreate(firefx, fireobj2);
```


18. Shaders Lesson 18


Shaders



Level: experienced

Version of the Xtreme3D: 3.0.x

Author: Gecko

One of the most interesting features Xtreme3D is a shader. The notion of "shader" here is wider than the other engines. Usually this term is used to refer the program to the GPU that are performed for each top model, or for each pixel of the model on the screen. Such shaders in the Xtreme3D also has a (see next chapter), but in a general sense, shader special called, or modifies computer software replaces a material to which it is attached. Some of these special effects work, and on the old graphics cards that do not support  program, and some are based on the built-in engine programs.

Using shaders can be superimposed on the object several materials, render the contours of the object, make the object embossed or give it the "effect of comic books." Consider all possibilities of built-in shaders Xtreme3D within a single lesson is impossible, so we stop at the one- relief (Bump Shader).

The effect of relief greatly increases the realism of models - it is used in the games for more than 10 years of age and over the years has become the de-facto standard.  is usually achieved by using the method of normal mapping (Normal projection). This method is based and Bump Shader in the Xtreme3D. The essence of the normal mapping in that normal is set for each point of the surface (in contrast to the normal vertex lighting, where the normal set for each of the peaks, and then simply interpolated over the surface of the polygon. This is done using normal maps normal map) - A special texture, in which the color of pixels compared to the normal vectors (RGB = XYZ). Normal Map can be generated from the map heights or from  geometry by tracing - This function is virtually all professional packages of 3D-modeling.



The normal map was relatively invariant speed and transfer model (i.e., remained unchanged in these transformations), its set in a special space, called the space of the tangent (tangent space). In this space, the coordinate axis Z is perpendicular to the surface, and the X and Y axes, respectively, are mutually perpendicular to the tangents to the surface. The lighting also calculated in the space of the tangent - direction of light is transformed in this space using a special matrix, which is called TBN on the first letters of its components - the tangent and BINORMAL, Normal (tangent, normal). Normal here - the usual normal tops, and the tangent and binormal - vectors, perpendicular to the normal and perpendicular to each other. These vectors calculates the Xtreme3D. Currently, they are supported only for objects of type Freeform.

Despite the quite complicated for beginners the theoretical base, use the terrain effect in the Xtreme3D is very easy - all the complexity of the realization of the hidden under the convenient API.

First, create materials with the necessary textures:

```
MaterialCreate('mBumpDiffuse', 'diffuse.png');  
MaterialCreate('mBumpNormal', 'normal.png');
```

Now create the terrain shader and convey to him the textures:

```
Bump ShaderCreate =();  
BumpShaderSetDiffuseTexture(bump, 'mBumpDiffuse');  
BumpShaderSetNormalTexture(bump, 'mBumpNormal');  
BumpShaderSetMaxLights(bump, 3);
```

[BumpShaderSetMaxLights](#) function defines the number of light sources, which should take into account the shader. Recall that the Xtreme3D supports up to 8 light sources - the same applies to the terrain shader.

Now you can create material and attach to him our shader:

```
MaterialCreate('mBump', '');  
MaterialSetAmbientColor('mBump', c_black, 1);  
MaterialSetDiffuseColor('mBump', c_white, 1);
```



```
MaterialSetSpecularColor('mBump', c_ltgray, 1);  
MaterialSetShininess('mBump', 32);  
MaterialSetShader('mBump', bump);
```




Lesson 19

GLSL basics

Level: experienced

Version of the Xtreme3D: 3.0.x

Author: Gecko

GLSL (OpenGL Shading Language) is a high-level language description of shaders. With it you can program the graphics pipeline - in other words, to manage the rendering of objects on the  and pixel level. For the processing of peaks is responsible  GLSL program, for processing pixels - .

Work with the GLSL implies knowledge of the principles of the dither and graphics OpenGL pipeline, as well as linear algebra. Since the Xtreme3D does not require such knowledge, the use of GLSL can be a very difficult task for a novice, therefore, it is recommended to pre-read books or manuals on the topic. A very useful will be acquaintance with the principles of the work in OpenGL, as well as at least a basic knowledge of C/C++.

The types of data GLSL

GLSL is strictly  language - any variable, it has a certain type. The language supports the following basic types:

Bool - boolean value

Int - integer

Uint -  integer

Float - a floating point number single precision

Double - a floating point number double-precision

Bvec2, bvec3, bvec4 is a vector of boolean values dimension (2, 3, and 4)

Ivec2, ivec3, ivec4 is a vector of integers

Uvec2, uvec3, uvec4 is a vector of Unsigned integers

The vec2, the vec3, the vec4 is a vector of numbers with a floating point

Dvecn2, dvecn3, dvecn4 is a vector of numbers with a floating point double-precision

Mat2, mat3, mat4 is a matrix of 2x2, 3x3, 4x4

Sampler2D - the texture of the

Sampler2DCube - cubic texture

Sampler2DShadow - Shadow Texture

Void - keyword that indicates the absence of the type (for functions without the returned results).

Vertex Shader

◆◆◆◆◆◆◆◆◆◆ accepts the coordinates of the vertices and their attributes (such as normal the moment and) and, as a rule, puts them out of the object space into the amputation in the world or in species space.

- The **Object space** (object space) is a local space of object. The center of the coordinate system is the center of the object - top models are defined relative to the center.

- The **world space** (world space) is another name for the absolute space. The center of the coordinate system, it is the point (0, 0, 0). The total transformation of the object (Transferring, rotation and scaling) puts the tops from the local to the global space. This transformation is typically stored and transmitted in the shader in the form of a matrix 4x4 - the so-called matrix model (model matrix).

- **Species space** (eye space - a space in which the Center coordinate system is the position of the camera. Translation of the peaks of the world in species space is controlled by the reverse conversion Matrix - the so-called species matrix (view matrix). In OpenGL, as a rule, the model matrix and generic are combined in one ◆◆◆◆◆◆◆◆◆◆-◆◆◆◆◆◆◆◆◆◆ (modelview matrix).

- **Space amputation** (clip space - a space in which the vertices are translated a matrix projection (projection matrix).

It should be noted that the tops of the GLSL is stored in the so-called homogeneous coordinates (homogeneous coordinates) - that is, have an additional fourth coordinate W. Such coordinates, you can express the infinitely distant point where W is equal to zero. Conventional terms have the W is equal to 1.

Peaks in the space of amputation is the main result of the work of the vertex shader. The simplest vertex shader that performs only translation of vertices of the object space into the amputation, is as follows:

```
Void main()  
{  
  Gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;  
}
```

Gl_Vertex - input the coordinates of vertices

Gl_Position - The output coordinates of the vertices

Gl_ModelViewProjectionMatrix - integrated matrix of 4x4, the combination of The $gl_ModelViewMatrix$ -species and projection matrices of OpenGL.

For this procedure, by the way, in the GLSL has a built-in function to transform:

```
Gl_Position = ftransform();
```

Vertex Shader are also available from other attributes of the peak - normal, color and texture coordinates: gl_Normal , gl_Color , $gl_MultiTexCoordN$ (where N is a number from 0 to 7). Usually these attributes of the interpolated between the three vertices of triangle, and then come in the gl_Normal shader. To transfer any value to the interpolation, used in varying intermediate variables. For example, here is the shader communicating to interpolate normal:

Varying the vec3 normal;

```
Void main()
{
    Normal = gl_NormalMatrix * gl_Normal;
    Gl_Position = ftransform();
}
```

Please note that we translate normal vertices of the object space for species with a special built-in a matrix of 3x3 $gl_NormalMatrix$. This is necessary so that the best way to count lights in pixel $gl_NormalMatrix$ - this is done in the species space: the fact that the camera is at (0,0,0), greatly facilitates the calculations related to the $gl_NormalMatrix$ component of the light.

With the transfer of texture coordinates of the shader will look like this:

Varying the vec3 normal;

```
Void main()
{
    Normal = gl_NormalMatrix * gl_Normal;
    Gl_TexCoord[0] = gl_MultiTexCoord0;
    Gl_Position = ftransform();
}
```

Gl_TexCoord - this is a built in varying-variable, array, through which you can transfer any data, not only the texture coordinates.

◆◆◆◆◆◆◆◆◆◆ Shader

◆◆◆◆◆◆◆◆◆◆ accepts the interpolated varying variables (as well as the various options for the status of the OpenGL) and displays as a result of the color pixel. It is performed for each visible on the screen pixels of the object. Please note that checking visibility (Z-test) for pixel graphics is carried out prior to the completed ◆◆◆◆◆◆◆◆◆◆ program - if a pixel is discarded as an invisible, the program is not performed. The simplest ◆◆◆◆◆◆◆◆◆◆ shader, the painter's object a solid color, looks like this:

```
Void main()
{
  Gl_FragColor = the vec4(1.0, 0.0, 0.0, 1.0);
}
```

Gl_FragColor - output color pixel.
In this case, the vec4(1.0, 0.0, 0.0, 1.0) refers to the red color with transparency 1.0 (Full Opacity).

The use of shaders in the Xtreme3D

Create GLSL shaders and connect them to the content is very simple:

```
Vp = TextRead('my_vertex_shader.glsl');  
Fp = TextRead('my_fragment_shader.glsl');  
Shader = GLSLShaderCreate(vp, fp);  
MaterialSetShader('myMaterial', shader);
```




Lighting on the GLSL

To realize the simplest lighting according to the formula of Lambert, we need the coordinates of the Point surface normal at this point, as well as the coordinates of the source of light. Thus, we need at least two varying variables - normal and interpolated vertex coordinates.

Vertex Shader:

Varying the vec3 normal;
Varying the vec3 position;

```
Void main()
{
    Normal = gl_NormalMatrix * gl_Normal;
    Position = (gl_ModelViewMatrix * gl_Vertex).xyz;
    Gl_Position = ftransform();
}
```

Gl_ModelViewMatrix is a built-in matrix of 4x4, The  generic matrix of OpenGL. It converts the coordinates of the object space for species in which we will calculate the lighting. As the result of this translation - a homogeneous vector of the vec4, we  coordinate W and charge only the vector of XYZ.

 shader:

Varying the vec3 normal;
Varying the vec3 position;

```
Void main()
{
    The vec3 N = normalize(normal);
    The vec3 L = normalize(gl_LightSource[0].position.xyz - position);
    Float diffuse = clamp(dot(N, L), 0.0, 1.0);
    The vec4 color = gl_FrontMaterial.diffuse * diffuse;
    Color.a = 1.0;
```

```
Gl_FragColor = color;  
}
```

Please note that transfer in the \vec{n} singular vectors (such as normal) after the interpolation is necessary - the graphics card does not make it for you. For this in the GLSL is a function `normalize`.

Access to the coordinates of the source of light is controlled by the position attribute the embedded object of `gl_LightSource` (array of 8 elements, according to the number of light sources OpenGL). These coordinates in the (x, y, z, w) is automatically transferred to the species space, which is very convenient - you do not need to do it manually. But if you are, for one reason or another, you need to calculate the lighting in another space - for example, in the space of the tangent - do not forget to transform them. These coordinates as a peer: point source of light, as a rule, has the coordinate of the W is equal to 1, aiming to equal 0.

The operation $\text{dot}(\vec{N}, \vec{L})$ - this is the calculation of the light sensor according to the formula: Lambert illumination at the point is determined by the density of the light, and it is linearly dependent on the cosine of angle of incidence of the light. The cosine of the angle between two singular vectors is equal to the product of The $\vec{N} \cdot \vec{L}$ (dot product).

Because the result of this operation - a scalar (float), for transfer to `gl_FragColor` need this value to any color. It is better to use Diffuse Color of material - `gl_FrontMaterial.diffuse`: thus, you can control the color of the object out of the shader, the function of the [MaterialSetDiffuseColor](#).

The textures

In The  can read color from the textures - for this is the function of the texture2D:


The uniform sampler2D diffuseTexture;

```
Void main()
{
  The vec4 texColor = texture2D(diffuseTexture, gl_TexCoord[0].xy);
  Gl_FragColor = texColor;
}
```

The textures are declared as uniform objects - that is, the modifiable parameters are transferred to the shader core program. This may be not only the texture, but also any other data types.

The transfer of the textures in the shader is done as follows:

```
Param GLSLShaderCreateParameter(shader, 'diffuseTexture');
GLSLShaderSetParameterTexture(param, 'myMaterial', 0);
```

In the function of the [GLSLShaderCreateParameter](#) is transferred to the name of the uniform object. In the function of the [GLSLShaderSetParameterTexture](#) is transferred to the name of the material, from which you want to read the texture, as well as the texture unit, through which you want to transfer the texture. The OpenGL standard guarantees 8 available texture units (0-7) - the modern video cards they can be and more (up to 16 or even 32), but for the best compatibility it is recommended not to use more than 8. In one  cannot transmit two different textures through one and the same texture unit - that is, if you submit multiple textures in different uniform-parameters, use the different blocks.

About the versions of the GLSL

Xtreme3D is based on the OpenGL 1.x and some of the functions of the OpenGL 2.x, which are connected through the expansion of the ARB. Thus, the engine supports GLSL versions 1.1 and 1.2 - later versions of the language defined in the OpenGL specification 3.0.

The default is GLSL 1.1. To switch to 1.2, use the preprocessor directive (on the first line of the shader):

```
#Version 120
```

Version 1.2 is built-in support for transposition of matrices (the function transpose), 4×4 matrices, as well as arrays.

From the Editors

In computer graphics have historically developed extensive specific terminology - this and the names of algorithms, and indicate the standards, and different kinds of abbreviations, which is quite complicated to sort out those who have just started to get acquainted with 3D graphics. We have compiled a glossary is intended to educate the beginners in this area, as well as to provide the necessary information on mathematical apparatus and technologies used in the Xtreme3D and other similar engines.

Engine

Engine | Engine, the engine

A central component of the program, designed to solve a specific problem. As a rule, the slider is formalized in the form of an independent module (for example, a dynamic library) for use in several projects.

The game engine is a universal or specialized software designed for the development of computer games. As a rule, contains a set of components for modeling of game situations, the withdrawal of the graphics and sound, as well as user interaction via the input device.



The graphics engine (2D, 3d or hybrid) is one of the key parts of the game engine, which is responsible for the output of graphics. Currently, the graphics engines are typically created on the basis of system of graphics API (DirectX, OpenGL, etc.).

See also [Physics Engine](#).


Environment Mapping


Environment Mapping | map projection environment

One of the forms of ibl is a method of texture overlay that simulates the effect of mirror reflections on the surface. As a result of the use of the environment map rendering on the reflections it takes much less time than when using a "fair" tracing.

Environment Mapping involves the use of one or more of the textures with The  or  reflections. The projection of the textures on the surface of the model occurs by the specific transformation texture coordinates. There are several methods of projection environment maps:

Sphere Mapping (spherical projection) - as the environment map is used the only texture, forming the surface of the imaginary of a hemisphere, the object from the viewer. This is the easiest and most effective method, but not realistic, as with any angle the viewer sees the same reflection. This method is also known as the Mirror Ball ("mirror ball").

Cube Mapping (cubic projection) - as a reader  uses six projection, forming the six sides of the imaginary cube surrounding the object. As a result, the effect is fully reflected from any point of the surface. The original images obtained by the preliminary or dynamic rendering. This method allows you to get the most realistic result at the cost of the high cost of video memory.

Equirectangular Mapping ( projection - a method of projection used in cartography. As the environment map is used only the texture covering the imaginary sphere around the object. Allows you to get a realistic result at minimum cost of video memory (artifacts of projection are the "poles", but they are usually not very visible). In the rendering of real time, this method has been widely only with the advent of shaders, as on a fixed line it is not maintained.

Dual-Paraboloid Mapping (the projection of dual paraboloid) is one of the most modern methods. Displays the top and bottom of the hemisphere of environment in two textures. This method has little artifacts (if not take a small deviation along the equator") and is the best compromise between memory consumption and the quality of the result.

See also [Reflection Mapping](#).



ERP

Error Reduction Parameter | The Coefficient of correction of an error

The coefficient, responsible for the restoration of the θ_{12} ties in the physical cursor (usually in the ODE), and showing what part of the θ_{12} will attempt to correct the θ_{12} link. So, for example, when the ERP = 1 θ_{12} tries for one iteration of the restore link in "Normal" position (minimize to each other the two ends of the θ_{12} of articulation, or push out entirely a couple of θ_{12} Tel).

See also [Physics Engine ODE](#).

Fillrate

Fillrate | fill speed

The rendering speed of pixels on the screen. The higher the fill rate, the better. The speed of modern video cards is measured in millions and billions of pixels per second.

Focal Length

Focal Length | Focus Distance

The distance in millimeters from the center of the lens to the image, which is part of the entity (presumably the infinite distance in front of the lens). Focused lenses short radius of action necessary for the formation of the wide angle images, while the distant focus distance is used for shooting with a telephoto lens.

Fog


Fog Mist |

The effect of gradual touching up model of a solid color (usually the background color) depending on the depth. The Fog attaches to the stage of realism, as well as allows you to hide unneeded geometry.



Forward Kinematics

Forward Kinematics | Direct kinematics


The movement of bones in the skeletal animation in the forward direction from the parent object to . For example, Turning Torso is the turn of the head, but the head can be rotated and regardless of the torso. See also the [Inverse Kinematics](#), [Skeletal Animation](#).

The FOV

Field of View | The Field of View

The angle (horizontal or vertical), which covers the projection of the 3d scene. In the frame buffer gets only some of the area of the scene - we look at the truncated pyramid visibility (view frustrum), the angle between the two opposing sides which forms a field of view.

In most 2D games it is impossible to speak about the angle of view - it is zero, and the pyramid is a rectangular tube.


The FOV seriously affects the perspective: when large values of the field of view (more than 90) visible object size decreases rapidly with distance. At small values, on the contrary, the visible size of an object is weakly depends on the distance (and it does not depend in the case of  of projection, i.e. in the absence of prospects - when the field vision is zero).


For understanding the term an analogy might be with a focal length of the camera, which directly affects its field of view.

In real life the horizontal angle of vision is approximately 140 degrees.

FPS

Frames Per Second | The number of frames per second

The number of frames per second, which  game. Than this number, the smoother will be animations in the game.

FPS is a crucial measure to evaluate the performance of graphics applications, as well as the graphics systems (the graphics card + driver). However, this is only possible when the vertical synchronization (VSync). When the vertical synchronization number of FPS may not be greater than the vertical frequency monitor (typically 60 Hz), so testing with vertical  always correctly.

See also the [VSync](#), [Fillrate](#).

First Person Shooter | First Person Shooter

The popular game genre that kind of action, in which the player observes the scene from "eye" of the character of the game. In the FPS distributed mainly battle themes, fight with monsters and space aliens. Classic examples: games series of Doom, Quake and Half-Life.

Frame Buffer

Frame Buffer | Frame Buffer

The memory for storing data about pixels required to display one frame image on the monitor screen. Capacity of the frame buffer is determined by the number of bits used to define each pixel.

See also the [Buffer](#).

Framework

Framework | Web Framework

The frame software system (or subsystem). May include support programs, the library code, scripting language and other, facilitating the development and consolidation of the different components of large software project. Usually association is due to the use of a single API.

Frame Buffer

Frame Buffer | Frame Buffer

The memory for storing data about pixels required to display one frame image on the monitor screen. Capacity of the frame buffer is determined by the number of bits used to define each pixel.

See also the [Buffer](#).

Frustum Culling

Frustum Culling | Selection of the pyramid of visibility

The method of selection of the invisible geometry, used for large polygonal objects (for example, landscape or interior scene). Redrawn only those objects that are fully or partially located inside of the truncated pyramid visibility (view frustum). All that is outside the pyramid, located outside of the screen.

Geometry

Geometry The Geometry |

A set of reference points, defining the shape of the object. For example, a cube is determined by eight points.













Global Coordinates

Global Coordinates | Global coordinates

The coordinate system, which defines the position of the object in space relative to the absolute coordinates.

GLScene

GLScene

Free outdoor graphics library based on the OpenGL for Delphi and Lazarus, originally developed in 1999, Mike     . With version 0.5 Development of the GLScene was continued by Eric        , and since 2006, the library is supported by a community of programrs, including Russia.

GLScene is continuously evolving, adapting to modern technology of 3D graphics. In addition to the graphics classes and components, the library provides tools for working with sound, I/O, the game logic and physics.

The high-level structure of the GLScene allows beginners to create games, not knowing any OpenGL commands, not presenting, as the matrix are multiplied together and how to write shaders. At the same time, professionals offering all the possibilities for the use of pure OpenGL, where necessary, modifying the source code under themselves and create professional applications.

On the basis of the modified the source code GLScene is built Xtreme3D.


Official site: <http://www.glscene.org>.

See also [Engine](#), [OpenGL](#).

The GPU

Graphics Processing Unit | The Gpu

The processor, designed for graphics rendering. From the central processing unit (CPU) is a special parallel architecture, which allows to solve some tasks much faster. Condition only one task must be observed parallelism.

The term was GPU  by NVIDIA, which in 1999 released a GeForce 256, and called it "the first GPU in history", although the Hardware dithering and relevant core appeared much earlier. So, the first arcade game machines with hardware T&L appeared in 1993 (Sega Model 2 and Namco Magic Edge Hornet Simulator). In consumer computer systems The first gpu appeared on the Sega Saturn, PlayStation and Nintendo 64 (1994-1996). On personal computers was one of the first dedicated graphics 3D-rendering was S3 Virge (1995), then the market leader has become the company 3Dfx Interactive was established, in 1996 and 2000, the Voodoo accelerator Graphics. Since 2000, the GPU market is divided between the NVIDIA, ATI (now absorbed by AMD) and Intel.

The GPU is designed for processing large arrays of similar data, such as points, vectors and pixels. Initially, the types of data that can be processed GPU (and the video processing algorithms) were strictly defined, but modern graphics processors are fully programmable - this means that the program can send arbitrary data and perform any algorithms. This has led to the introduction of the term GPGPU - General Purpose GPU (GPU).

HDR

High Dynamic Range | wide dynamic range

Description of the color of the real physical quantities.











The familiar model description of the image is RGB, when all colors are presented in the form of the amount of the Basic colors: red, green, and blue, with different intensity in the form of a possible integer values from 0 to 255 for each, the encoded eight bits of the color. Attitude of the maximum intensity to the minimum available to display a particular model or device is called the dynamic range. Thus, the dynamic range of the RGB model is 256:1 or 100:1 cd/m² (2). This model describe the color and intensity is generally accepted is called Low Dynamic Range (LDR).



























The possible values of the LDR in all cases is not enough, a person is able to see a much greater range, especially at low light intensity, and the RGB model is too limited in such cases (yes, and when large 10^{-6} up to 108 cd/m², that is, 100000000000000:1 (14 orders). At the same time, the entire range we see cannot, but the range, visible eye at a time, approximately equal 10000:1 (4). The vision adapts to the values from another part of the range of light conditions gradually, with the help of the so-called adaptation, you can easily describe the situation with the light off in the room at night - first the eyes see very little, but over time, adapt to changing light conditions and see the already much more. The same thing happens and when back to change the dark environment in the light.

So, the dynamic range of the RGB descriptions is not enough for the submission of images that people can see in reality, this model significantly reduces the possible values of light intensity at the top and bottom of the range. The most common example, driven in HDR, - the image of 10^6 spaces with window to the bright street on a sunny day. With the RGB color model can be obtained or normal display that is located outside the window, or just the fact that inside the premises. Values greater than 100 cd/m² in the LDR is truncated, this is the reason that the 3D-rendering it is difficult to correctly

display the bright light sources aimed directly in the camera.


The display device until that seriously cannot be improved, but the failure of the LDR in the calculations of makes sense. You can use the actual physical value of intensity and color (or linearly proportional), and on the monitor display maximum that it can. The essence of the submission of HDR in the use of intensity values and colors in real physical terms or linearly proportional to use integers and floating-point numbers with great accuracy (for example, 16 or 32 bits). This removes restrictions on the RGB, and the dynamic range images seriously increase. Then any HDR image can be displayed on any vehicle display (the same RGB monitor), with the highest quality possible for him with the help of special algorithms for tone mapping.

HDR rendering allows you to change the exposure after we           image, gives an opportunity to simulate the effect of the adaptation of human vision (movement of bright open spaces in the dark spaces and vice versa), allows you to physically correct lighting, as well as a unified solution for the application of the postprocessing effects (glare, flares, bloom, motion blur). Algorithms of image processing, color correction, gamma correction, motion blur, bloom and other methods of qualitative processing performed in HDR view.

In the annexes 3D rendering Real Time Gaming, mostly) HDR rendering began to use relatively recently, because it requires computing and render target in the formats of floating point operations, which first became available only on the          with support for DirectX 9. The usual route HDR rendering in games is as follows: the rendering of the scene in the buffer format of floating-point image postprocessing in the expanded color range (changing the contrast and brightness, color balance, the effects of glare and motion blur, lens flare and such), the application of tone mapping for the withdrawal of the final HDR images on the LDR screen. Sometimes the map used environment (environment maps) in the HDR formats, for the static reflections on objects, a very interesting application of HDR in the simulation of dynamic          and reflections, this can also be used dynamic maps in formats with a floating point. To this add another         (light maps), pre-calculated and stored in the HDR format. Much of the above made, for example, in the Half-Life 2: Lost Coast.

HDR rendering is very useful for post-processing complex of higher quality,

compared with conventional methods. The same bloom will look realistic in the calculations to HDR model of submission.

Unfortunately, in some cases, game developers can hide under the name of HDR filter simply bloom, calculated in the usual LDR range. Although a large part of that now make the games with HDR rendering, and there is a bloom of better quality, benefit from the HDR rendering is not limited to one these , simply to make it easier.



Hidden Surface Removal

Hidden Surface Removal | Remove hidden surfaces

Method of determining the visible to the observer of surfaces. Allows not to display invisible from the point of the surface of the object.

Hierarchy

Hierarchy the Hierarchy |

The system of relationships between objects, in which some of the objects ("the descendants") are subordinate to other ("parents"). A number of parameters of the descendants of, for example, movement, rotation and scaling, dependent on their parents. At the same time, direct change the descendants will not affect the status of the parent.

The HUD

Head-Up Display

Part of the GUI, which displays the most important information on the screen during the game. For example, the standard of living, glasses, ammunition, the mini-card and others. The information can be displayed as both a digital (symbols) and the analog form (scale).

Interpolation

Interpolation Interpolation |

The mathematical method to restore the missing (intermediate) information on the existing set of known values. The simplest type of interpolation method - linear: if there are two values A and B, the range of intermediate values between them is obtained by the formula $A * (1 - t) + B * t$, where t is a number from 0 to 1. Linearly interpolate can not only be real numbers, but also the vectors. The result of linear interpolation of two colors is a linear gradient.

Inverse Kinematics

Inverse Kinematics | Inverse Kinematics

The movement of bones in the skeletal animation in the reverse direction from the object generated by the parent. In this case, for example, the movement of the hands causes the movement of the shoulder.

Keyframing


Keyframing | Creating key frames

The process of animation values in time. Each key point is the value of the parameter in a certain frame. Values between key frames are calculated by interpolation. To create the key frames can practically for all parameters, which are determined by numerical values.

Lensflare

Lensflare | glare on the lenses

Light effect caused by scattering and refraction of light in the system of lenses, if within the camera's field of view gets a bright glowing object. This effect is often imitated in computer gaming from a third person to make the graph



Light Source

Light Source | Light Source

An element of the scene, which creates the light. Light sources can be of different types and have different characteristics. The analogues of lighting devices in the real world, which offer additional features, inaccessible real sources.



Typically, there are three main types of light sources:

Point Light (point) is the source of light which shines equally in all directions from one point (for example, the light in the room). Also there is the omni designation (omnidirectional) light.

Spot Light (spot) - a light source shining in all directions, and within some of the cone. Highlights the only objects that fall into this cone. A simple example is the flashlight.

Parallel Light (parallel light) - simulates the remote light sources, such as the sun. The light is emitted in the direction of the only one axis, from the source, of the infinitely great distance from the viewer, and all light rays are parallel. The synonyms of the term: distant light, directional light.

Point and directed light sources are the parameters affecting the lighting, for example, attenuation (attenuation). This setting affects the decrease of light intensity with distance. In the quadratic equation that determines the intensity of the light, composed of three parameters, called constant, linear and quadratic components (constant, linear, quadratic). By default, they are assigned a value of 1, 0 and 0 respectively - the intensity of the light from the source does not decrease with distance.

The equation is as follows:



Where a is the value of the attenuation; - constant component; l is a linear component; d is the distance from the light source, a q - quadratic component.

Lightmap

Lightmap | Map Light

The outdated, but still applied because of its simplicity, the method of static lighting surfaces. The principle is as follows: at the base texture impose another map light, the light and dark places which alter the light intensity baseline, blending in the modulate. To overlay the light card model, as a rule, provides an additional set of texture coordinates.

This method is applicable only for static models and fixed light sources.





Line Buffer

Line Buffer | Inline buffer

The memory buffer used to store one line in the video. If the horizontal display resolution is set to 640 and to encode color, use the RGB diagram, the linear buffer will be the size of 640x3 bytes. The Linear buffer is typically used in algorithms of filters.

LOD

Level of Detail | degree of detail

Optimization method of rendering in graphic engines, based on reducing the detail of the models as their distance from the camera. There are a discrete LOD, providing for storage of several models in memory and switching between them, and dynamic (or continuous), based on the   in real-time. Dynamic LOD is typically used for rendering landscapes.























































































Model

Model | Model

One of the ways of objects in the scene. The model can contain not only information about the geometry, but also functional curves, tenders, and many other properties that define the included in the elements. This term can also be attributed to the objects and characters.

Map

Map | Site Map

The scanned or drawn a raster image, representing those or other properties of the surface in each of its point - for example, the color or normal. The card is also referred to as textures, while the texture more accurate to call the image stored in the memory. For rendering the models use the color (diffuse) maps, normal, the lightmap, maps of heights,         maps, environment, a luminous, and so on. In modern physically sound rendering also applied                                   cards and roughness. The process of map overlay on the surface of the model (which is the display of the spatial coordinates in the texture) is called The         or                                  .

See also the [texture](#).

Material

Material | Material

The totality of the surface parameters that define the appearance of the object. These parameters are usually include color components, the importance of transparency, one or more of the diffuse texture, normal map, a map of the light and etc. It should be borne in mind that in game 3D cursors the concept of "material" refers to the surface, and not to the volume of the object, because the dither is subject to only the surface.



Matrix

Matrix | The Matrix

The two-dimensional table. The matrix are identified as $N \times M$, where N is the number of rows in the table, M is the number of columns. In linear algebra is typically used in the square matrix (where the number of columns and rows is the same). In the 3D graph square matrices are used for the linear transformation vectors and points from one space to another.

The matrix can be multiply, invert (calculate the inverse matrix), as well as transposing - that is, to swap rows and columns (the line becomes the column and vice versa). A special kind of a square matrix - a single. In a single matrix of all elements, in addition to the main diagonal equal to zero, and the main diagonal (from the top left corner to the lower right) contains the units. A single matrix is equal to its inverse and $I \cdot I = I$ matrices. Multiplication of the matrix on hop count will result in exactly the same matrix.

Matrix multiplication is performed on the columns and rows for each element of the resulting matrix corresponding to this element of the string one scalar matrix is multiplied by the appropriate column on the other. The multiplication of matrices $A \cdot B$ - that is, the matrices A and B work $A * B$ does not necessarily equal to $B * A$.

You can also $M \cdot v$ matrix and vector, if adhered to a certain conformity to their dimensions. There are two types of such multiplying - left and right. The left multiplies the $N \times M$ matrix to a vector-column in the dimension of M , and the result is a vector of dimension N . Right multiplies a vector-string dimension N of the matrix $N \times M$, and the result is a vector of dimension of M . Vector-string is a vector of dimension n , recorded in a matrix of $1 \times N$, vector-column is a vector of dimension M , recorded in the form of a matrix $M \times 1$. In the rest of the multiplication rules are the same as for matrices is a string in the column. The right multiplication corresponds to the left with $v \cdot M$ matrix (and vice versa), and this property is often used in computer calculations for the various optimizations.

See also the [Transformation](#), the [Vector](#).

Mesh

Mesh, Polygonal Mesh | The polygon feature mesh

The final lists of vertices, edges and faces are called the polygon mesh, if its components meet the following conditions:

- Each vertex must have at least one rib
 - Each edge must be at least one facet of the
 - If the two facets of intersect, top or ridge, which turned out as a result of the intersection should be components of the polygon mesh. If all the flats in the grid - triangles, the object is called a triangle mesh (triangle mesh or abbreviated trimesh).
-


Metaball Modeling


Metaball Modeling


The process of creating 3D models of individual spheres (or, in rare cases, from other geometrical Tel), which interact with each other as the distance between them. Such technology is applied, in particular, in the re-establishment of organic objects.




Mipmapping

Mipmapping  |

The -mip (multum in parvum) means "done in one".

 optimized set of images associated with one texture and is designed to increase the speed of rendering and improve image quality.

Each image in the set of less than half the . I.e. the first has a size equal to the size of the texture, the second half less, the third - four times, etc. up to size 1x1 texel.

The meaning of the  sets is that when texturing will  image with the most suitable size. For example, when rendering remote surfaces or surfaces under a small angle to the camera is preferable to choose the texture size smaller to eliminate the effect of the .





Model



Model | Model

One of the ways of objects in the scene. The model can contain not only information about the geometry, but also functional curves, tenders, and many other properties that define the included in the elements. This term can also be attributed to the objects and characters.

Motion Blur

Motion Blur | blur when driving

The effect of  (""), arising during the photo and the houses of the movement of objects in the scene during the exposure time frame. In the 3d animation virtual camera has an infinitely small exposure, therefore, lubrication, such acquired camera and human eye at the sight of the fast-moving objects, missing - it is usually imitate artificially using directional blur (directional blur).

The blur when driving is used in almost all racing games (to create the effect of high-speed drive), Sports simulations (for fast moving objects, like the ball or washers), as well as The  (fast movement of cold arms, hands and legs). The blur sometimes when driving is used in games from the first person when quickly turning the camera - to make the picture .



Motion Capture

Motion Capture | Gripper movements

The Synchronization Technology movements alive actor and a virtual character in real-time using special hardware and software complex. Used to write complex movements (for example, sports) for use in 3D animation and computer games.

Multitexturing

Multitexturing | 

The process of imposing of two or more textures to the object.

Normal

Normal | Normal

Vector value, indicating the direction perpendicular to the surface. The normal triangle is defined as a vector product of two edges of the triangle. In programming, a normal graphics can be set for each triangle, describing a certain surface, and for each of the peaks depending on the desired result. The length of the normal equal to 1. Bringing the vector length to one with its directions is called normalization.

Normal Map

Normal Map | Map of normal

Map that defines a vector normal at each point of the surface. Used to generate surfaces and bumpmapping and other algorithms. Normal Map usually presented texture, in which the data is written in such a way that the RGB values into the XYZ Company, where Z is perpendicular to the surface of the vector.



NURBS

Non-Uniform Rational B-Spline | Mixed rational B-spline

The common way of specifying the parametric curves and surfaces.

OBB

Oriented Bounding Box | oriented bounding parallelepiped

The area in the space surrounding a certain object, in the form of rectangular parallelepiped, which rotates with the object. Used in various geometric operations, replacing the object itself in order to simplify and accelerate the computing - for example, if the clashes, the layout of the rays and the partitioning of space. Has the advantage over the AABB, consisting of the same degree of approximation in any rotating object.

See also , The AABB [Collision Detection](#).

Object

Object The object |

The general term used to denote the informational nature. The object is a fundamental concept in object-oriented programming and appropriate policy environments where under it refers to the special structure of the data.

The object is often the information model of any object or phenomena in the real world. Operate the information at the level of the objects significantly easier and more clearly than at the level of the data, so information systems on the basis of the objects distributed everywhere.





Occlusion

Occlusion

An overlap of three-dimensional space of one object to others.

Octree

Octree |  Tree

The structure of the data representing the Euclidean space in the form of a  tree, in which each element is Aabb. Each cube is divided in three planes at 8 (usually a mutually equal) cubes. Octree are typically used to partition large open  spaces. Note that under the "flat"  refers to the space, in which the camera movement is limited mainly some plane: An example may be a . For the "flat" spaces more suited quadtree. Like many other methods of splitting, The Octree is used to optimize the collision detection and the frustum culling.










ODE
















Open Dynamics Engine

Free outdoor library of industrial quality for modeling the physics of solids. Suitable for the simulation of vehicles, creatures with feet and moving objects. ODE Fast, flexible and reliable, has a built-in system of collisions. ODE a physicist Russell Smith , together with a team of volunteers.











OpenGL

Open Graphics Library

Graphics Library approved by the industrial standard, developed in 1992, nine leading IT-companies: Digital Equipment, Evans & Sutherland, Hewlett-Packard, IBM, Intel, Intergraph, Silicon Graphics Corp., Sun Microsystems and Microsoft. The iso is The          IRIS GL, developed by Silicon Graphics. The OpenGL library is quite simple to use and training, has a very wide range of opportunities. Here are some of its advantages:

The stability of the OpenGL is an entrenched standard. All changes made to it,            ahead and implemented so that the already existing is not     on new maps.

Reliability - all applications that use OpenGL, guarantee the same visual result, regardless of the hardware and operating system.

          - applications using OpenGL, can run on different architectures and different operating systems (OpenGL provides portability at the source code).

The main feature of OpenGL - his client-server architecture that theoretically allows you to place the client (application using OpenGL) and the server (the executive part of the OpenGL) on different machines.

OpenGL develops with the help of the mechanism of "extensions" - special modifications to the basic version of the API, which add new features and/or expand existing ones. When the accumulated solid luggage such changes (extensions), a consortium of the OpenGL specification releases a new version of OpenGL. At the moment, the latest version of the specification - 4.0.

The Parallax Mapping

The Parallax Mapping

Mimics the effect of terrain visibility on the surface. Represents a superior Normal mapping. Improvement is that the normal map not only affects the illumination of surfaces, but also on the offset of the texture coordinates. That is, the terrain becomes more realistic and different looks from different angles.

See also the [Normal Map](#), [Bump Mapping](#), [Displacement Mapping](#).

Particle System


Particle System | The particle system

Animation System, consisting of a large number of very small objects whose behavior is defined mathematically. The particle system usually consists of $\diamond\diamond\diamond\diamond\diamond\diamond\diamond$ (which may be point, surface or volume, and can emit particles or sent in all directions) and a number of areas that define the behavior of the particles (attractors, the deflectors, and changers $\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond$). Each particle has the ultimate life expectancy, and can have their attributes (color, radius, transparency), which change during this lifetime. The particles are usually used for the simulation of fire, smoke and other effects.



Physics Engine

Physics Engine | Physics Engine

Under the concept of "Physics Engine" in programming decided to imply the program (or program), The  a physical process. A classic example is engine, simulating the physics of solid bodies on the basis of the pulses.

Pipeline

Pipeline | Conveyor

Step-by-step method of rendering 3D graphics (rasterization). Its name was due to the fact that the output of each step (stage) are the input for the next.

1. Transformation. This stage begins to vertex conveyor, where the processing unit is the top of the (point in the \mathbb{R}^3 space). A set of vertices, forming a three-dimensional object is transferred from model space in the world, and then in the species, where the beginning of the coordinates is the position of the camera (these two conversions are usually combined into one). The peaks of the transferred promising matrix into the space of amputation. In the programmable pipeline stage of transformation is made in the \mathbb{R}^3 .

2. Amputation. The invisible top (outside the pyramid of Visibility) is discarded and not transferred further.

3. Normalization. From the amputation of the vertices into normalized device coordinates (NDC) - that is, in fact, in the two-dimensional coordinates on the on-screen a plane + Z coordinate, indicating the depth of the screen on the plane.


4. Dithering. The set of points is used to rasterize polygons (usually triangles) according to the rules of building the ribs. In the programmable pipeline these rules can be configured in a geometrical \mathbb{R}^3 . Also at this stage, the interpolation of the vertex attributes (color, normal, texture coordinates, etc.) on the surface of the polygon. Is perspective correction texture coordinates. From this moment begins the pixel pipeline - the data is processed pixel.

5. Calculation of the color. In the programmable pipeline the resulting pixel color is calculated in the \mathbb{R}^3 . At this stage, the calculated the pixelwise lighting, texturing, terrain, reflection, shadows, transparency and other effects. As input used interpolated vertex attributes, as well as the texture.

6. Checking the depth. Before you write a pixel in the frame buffer may be depth (depth test). The Z coordinate of the pixel is compared with the corresponding depth in the Z-buffer. If it is less than, the pixel should be \mathbb{R}^3 , if not - has been ignored.

7. Mixing. The color of the pixels can according to the specified rules mixed

with the already present color in a buffer frame. The total value is written to the buffer.

It should also be taken into account that your conveyor, there are the second most common method of 3D-rendering - tracing. It is more simple and intuitive and operates mainly rays. Sets of vertices are only used for determining the beam, the depth buffer no, and perspective projection is performed by calculating the starting direction of rays for each pixel of the image. The final pixel color is calculated similar to the shader  manner.

Pixel

Picture Element | Pixel

The combined term that refers to the smallest image element or the monitor screen. The image on the screen is composed of hundreds of thousands of luminous points, united to form the image. A pixel is the minimum segment of the raster line, which is controlled by the discrete system, forming the image.

Polygon

Polygon Polygon |

The polygon, which is a constituent part of any 3D object. In modern 3D-graphics under the polygon often imply the triangle, described by the coordinates of the vertices in space.

Post Processing

Post Processing | Post-processing

Series of transactions over an image obtained as a result of rendering, which includes the correction of brightness, contrast, saturation, as well as the use of a variety of effects and filters.

Primitive

Geometric primitive | Geometry

The point, segment or polygon.

Procedural Texture

Procedural Texture | Procedural Texture


Texture is described by mathematical formulas. Such textures do not occupy the memory locations, they are created pixel shader "on the fly", each element (Texel) is obtained as a result of the execution of the commands shader. The most common procedural textures: there are different types of noise (for example, the fractal noise), wood, water, lava, smoke, marble, fire, etc., that is, those that are relatively easy to describe mathematically.


Unfortunately, the procedural textures not received until the proper application in games.



Projection

Projection Projection |

Displays the three-dimensional space on a plane by building a  lines. In computer graphics distributed three types of projection:

Orthographic (orthogonal) - the location of the spectator is infinitely removed from the scene, so all the lines along the same axes are parallel. The  projection is isometric.

Parallel (parallel) is a type of orthogonal projection, parallel to the coordinate axes. Parallel to the projection may include types of Front (Front View), Top (top view) and Right (Right Side).

Perspective (perspective) - Parallel Lines visually converge at one point. The perspective is the most famous projection type and, as a rule, used most often.

Projection Matrix

Projection Matrix | projection Matrix

The matrix size of 4×4 , is used for the transformation of the entities of the species space into the amputation.


Quad

Quadrilateral | Quad

The Spatial quadrilateral (including composed of two triangles).

Quadtree

Quadtree | Wood quadrants


The structure of the data representing the Euclidean space in the form of a quadratic tree, in which each element is Aabb. Each square is divided into 4 (usually a mutually equal) squares. Quadtree are typically used to partition large flat spaces, in which the camera movement is limited mainly some plane: An example may be a .

Like many other methods of splitting, The Quadtree is used to optimize the collision detection and the frustum culling.

Ragdoll



Ragdoll | Rag Doll




The method used to create the physical model of the behavior of the human body.

In most cases, the rag doll is used in conjunction with the skeletal animation, as the principle of their work in many respects similar. And in the skeletal animation, and in the rag doll is a special hierarchy of "Bones" for the vertex shadowing model. However, unlike the skeletal animation, tops are moved not by predefined rules, and on the basis of the physical model of solids. For example, one bone of the skeleton rag dolls can be represented by a triangle mesh (or  or LOD'57 in order to accelerate the computing), and then all vertices move, as well as in the skeletal animation.

Also known as

Also known as | Dithering


The rendering method in which a raster image is obtained by finding the pixels belonging to the specified . , as a rule, segments and polygons (usually triangles). In generalized sense dithering is the process of translation vector data to raster image.

Dithering is by far the most common method of rendering 3D graphics because of its simplicity and efficiency. Dithering  easily, so for rendering in real time were created dedicated multi-core processors, graphics  (GPU), which are capable of  millions of triangles per second. It is DITHERING lies at the basis of virtually all modern 3D games.

See also [GPU](#), [Pipeline](#), [primitive](#), [Rendering](#).

Raycasting

Raycasting | throwing rays

Limited  tracing without tracking the reflected and refracted rays. Used to determine the visibility and collision detection. Along some directions is available beam, are all the crossings of the beam with objects and select the nearest.

Ray Tracing

Ray Tracing | Ray Tracing

The rendering method in which a raster image is obtained by tracking rays and their intersections with three-dimensional objects. Ray tracing in many respects is similar to how it turns out photographic image in the real world, and allows you to receive images, the approximate on the reliability of the photographs (sometimes up to 100%).

The trace algorithm is as follows: for each pixel of the image from the camera position is available beam, which is then checked for intersection with the objects in the scene. On the basis of points and normal border, as well as the characteristics of the surface and a set of known sources of light, the color of the calculated surface at this point, which is assigned to the pixel.

The algorithm naturally supports the forward-looking projection (the rays diverge as the distance from the camera), the selection of the Depth (only the nearest to the screen plane intersection) and the building of shadows (to check if the point in the shadow, is yet another beam in the direction of the light source - if it crosses the surface before it reaches the source, the point is shaded). The introduction of additional rays traced is expanded to support the soft shadows, reflections, 100%, indirect lighting and other optical phenomena in the real world.

Lack of tracing, which is preventing the full use of the method for rendering in real time, is the huge amount of 100% computing. In recent years, the opportunity to trace the parallelization using GPU's general purpose - it allows to hope for changing the situation in the future. Now possibilities of modern video cards allow you to organize a hybrid rendering - rasterization with elements of the trace for some complex effects like reflections.

See also the [rendering](#).

Real-Time

Real-Time | Real Time Mode

The time scale at which data processing flows with the same speed as the simulated events.

Reflection Mapping

Reflection Mapping | map projection reflect the

See [Environment Mapping](#).

Rendering

Rendering | Rendering, visualization of the

The process of formation of a flat image on the basis of mathematical models. Most often this term is used 3D-visualization - building a virtual images of 3D objects and scenes. The program performs rendering, is called the rendering engine or the rendering engine (rendering engine).

Rendering can be carried out both in real time (that is, the rendering of animation with the immediate withdrawal of the received frames on the screen and a sufficiently high frame rate is 30 frames per second and above) and in the offline mode (i.e., without restrictions on the time frame). Of course, that in order to achieve high speed to render sacrifice volume calculations and use a simplified mathematical apparatus, so the engines offline rendering and produce much higher quality and realistic images. The rendering in real time - the basis of all modern Games, offline rendering is used in film and animation, design, advertising, industrial and scientific field.

There are two basic methods of 3D-rendering - rendering and ray tracing. Dithering is more often used in the rendering of real time, a trace is in offline rendering, although it is not uncommon, and exceptions: for example, there are popular offline rendering engines that use the rasterization. There are also hybrid engines rendering, combining the rasterization and ray tracing.

See also [Also Known As](#), [Ray Tracing](#).

Resolution

Resolution |

The number of pixels per unit length or area.

RGB

Red, Green, Blue

◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆ system, in which the end color is obtained by mixing with varying intensity of three basic colors: red (Red), green (Green) and blue (Blue). The most well known device that uses the RGB, it is a color monitor.

ROAM

Realtime Optimally-Adapting Meshes | optimum adaptation of the polygon mesh in real time

The algorithm of adaptive approximation of complex surfaces, used mainly for optimization of the drawing of landscapes.


Scene

| The Scene The Scene

The totality of objects in three-dimensional space, modeling a limited environment: interior, exterior, landscape, part of outer space, and so on. The scene can contain both static objects and dynamic.

Seamless

The Seamless Seamless |


This term is used to denote the type of texture. The texture is made in such a way as to any side could dock another the same without clearly visible seams docking. This is the effect of the opposite side of the images have the same figure. Seamless textures used mainly for landscapes, as well as for the  large static objects (buildings, roads, water surface, etc.)

Sometimes used animated seamless texture, consisting of a few consistently displayed frames - usually to simulate fluids or moving objects - the water surface, lava, clouds, etc.


Shader

Shader | Shader

The firmware for one of the stages of the graphics conveyor, used to determine the final parameters of an object or image. It may include arbitrary complexity of the description of the absorption and scattering of light, texture overlay, reflection and refraction, shading, offset surfaces and the effects of post-processing.

Currently, shaders are divided into four types: Vertex, geometry, The  (pixel) and computing.



Vertex Shaders (Vertex Shader)

Vertex Shader operates with the data, with vms with vertices polyhedra. To such data, in particular, the coordinates of the vertices in space, texture coordinates, the tangent-vector, normal vectors and . Vertex shader can be used for species and future transformation of vertices, generate texture coordinates, a simple calculation of lighting, etc.


The geometry shaders (Geometry shader)

The Geometry shader, in contrast to the vertex, is able to handle not only one peak, but also the whole entity. This can be a piece of (two peaks) and the triangle (three tops), and if the information about the related tops (adjacency) can be processed up to six vertices of the triangle for the entity. In addition the geometry shader can generate entities "on the fly", without using the central processor.

(pixel) shaders (Fragment Shader)


 shader works with the fragments of the image. Under the image fragment in this case refers to the pixel, which put in a line some set of attributes, such as color, depth, texture coordinates.  shader used on the last stage of the graphics pipeline for the formation of the pixel of the image.

The Compute shaders (Compute Shader)

Fully universal shader by which it is possible to carry out arbitrary computations on the GPU not related directly to the rastering polygons.  computing are available for reading and writing data into the memory, which can then be used in the process of rendering.

There are three main groups of programming languages for the GPU. The first group includes the languages used when rendering images and animation in areas such as film, television, industrial design and architectural rendering:



RenderMan Shading Language (RSL) - developed and used by the studio Pixar. Is the de facto standard in professional rendering.



Open Shading Language (OSL) - designed Sony Pictures Imageworks for rendering engine Arnold, however, supported and in many other render engines. Focuses on the rendering using ray tracing, is intended to describe the BSDF -  functions of surface scattering the.

Gelato - developed by nVidia. Represents a hybrid system image rendering and animation that uses for the calculation of the central processors and hardware capabilities of professional series graphics cards Quadro FX.

Vector Expressions (VEX) - developed Side Effects Software as part of a package of Houdini. Is the analogue of RenderMan.

The second group includes the languages, providing access to the computing capabilities of the graphics card when rendering in real time. They are widely used in the development of computer games and other multimedia applications.

Low-level language  OpenGL (ARB) - the syntax is similar to the . Is available in the form of extensions of ARB_vertex_program, ARB_fragment_program. Is the approved industry standard.

The OpenGL Shading Language (GLSL) - a high-level language  OpenGL. Based on the syntax of the ANSI C. The Majority of C has been saved, the added vector and matrix data types that are often used when working with 3D graphics. In the context of the GLSL shader is called regardless  unit, written in this language.

The program is a set of compiled shaders are associated together. Initially, GLSL 1.10 was available in the form of a set of extensions of GL_ARB_shading_language_100, GL_ARB_shader_objects. Starting with the OpenGL 2.0, became part of the standard. With the release of OpenGL 3.3, the GLSL is changing the numbering of the versions. Now, the version number of the GLSL corresponds to the version of OpenGL.

With the for graphics (CG) - a high-level language, developed by nVidia , jointly with Microsoft (a similar language from Microsoft - The Hlsl is part of DirectX 9 and 10). Works with both OpenGL and DirectX supports various software and hardware platform.

Based on C, uses similar data types. Supported by the functions and structure. Includes a kind of optimization in the form of packed arrays (packed arrays - The announcement of the "float a[4]" and "float4" in it correspond to the different types. The second announcement, and there is a packed array, which are faster than normal.

Currently, Cg has already practically is not used, fully HLSL and the GLSL.

Low-level language DirectX (ASM), the syntax is similar to the . There are several versions, differing on a set of commands, as well as on the required equipment.


High Level Shader Language (HLSL) - a high-level language DirectX (also supported by the game consoles Xbox and Xbox 360). Is the superstructure over DirectX ASM. The syntax is similar to C, allows the use of the structure and function.

The third group of languages of broad specialization, intended mainly for scientific computing. They effectively use multi-core CPU and GPGPU Support to expedite the processing of large data sets.

Sh - a high-level programming language for GPU, included in a subset of the language C++. It was initially developed by a group RapidMind (which later became part of Intel), currently is licensed under the GNU LGPL and is supported by the community.

Compute Unified Device Architecture (CUDA) is a technology developed by

NVIDIA for parallel computing on graphics cards GeForce (8 and older), Quadro and Tesla. CUDA uses specialized language version C with a set of instructions for the GPU.

OpenCL (Open Computing Language) is a cross-platform equivalent CUDA, independent of hardware computing API with your own C-like programming language. OpenCL specification is currently being developed by a consortium of Kronos Group in parallel with OpenGL - there is the possibility of interaction between the  two API.

BrookGPU - project at Stanford University. Originally emerged as a language for programming streaming architectures. Is a C-like language, which added the data type is an array of special form ("Thread" in the terminology of the language). In 2004, got his realization for graphics processors.

Shading Components

Shading Components | Lighting

In the light of the surface is calculated as the sum of the components of the ambient, diffuse and specular from all light sources in the scene (ideally from all, often neglected by many). Impact on the value of each source of light depends on the distance between the light source and a point on the surface.

Uniform component of light (**ambient**) - the approximation of the global lighting, "initial" lighting for each point of the scene, in which all points covered equally and illumination does not depend on other factors.

The diffuse component of the light (**diffuse**) depends on the position of the light source and from normal surface. This component of lighting is different for each of the peaks of the object that gives them the volume. The light is no longer fills in the surface of the same shade.

◆◆◆◆◆◆◆◆ component of light (**specular**) is manifested in the specks of reflected rays of light from the surface. For its calculation, in addition to the vector of the source of light and normal, used two of the Vector: vector of the Direction sight and the vector of reflection. ◆◆◆◆◆◆◆◆ component first proposed Fong. These glare significantly increase the realism of the image, because rare real surface, do not reflect light, so the specular component is very important, especially when driving, because the lens flare i can change the position of the camera or the object itself. In the future, researchers have invented different ways to calculate this component, the more complicated (◆◆◆◆◆◆, Cook Torrance, Ward), taking into account the energy distribution of light, his absorption of the materials and dissipate in the form of diffuse component.



Shading model

Shading model | Lighting Model

The method of calculating light polygons. In the real time most prevalent received three lighting models:

- Flat (flat)
- On the $\diamond\diamond\diamond\diamond$ (Gouraud)
- The $\diamond\diamond\diamond\diamond\diamond$ (Phong)

When the light flat polygons as a stand (this is due to the calculation of the same color for each pixel on the edge), so when applying the fill a polygon will appear to be continuous.

The $\diamond\diamond\diamond\diamond$ and $\diamond\diamond\diamond\diamond\diamond$ give a smooth gradation of chiaroscuro. In the model of $\diamond\diamond\diamond\diamond$ lighting $\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond$ is calculated, the model $\diamond\diamond\diamond\diamond\diamond$ - pixel.

There is also a variety of other models of light, which in recent years have become widespread in 3D games and offline rendering: $\diamond\diamond\diamond\diamond\diamond$ -Phong, the Cook Torrance, Ward, Torrance- $\diamond\diamond\diamond\diamond\diamond\diamond\diamond$, lafortune, etc. Many of them are quite accurate approximation to the real physics of light. In the terminology of optics the most close to the lighting model concept - BRDF (dual beam function of reflectivity).



Shadow Map

Shadow Map | shadow map

One of the methods of construction of shadows, namely, the use of Z-buffer to determine the pixel, whether the target point in the shadows. The method of shadow maps based on the idea that the lit terms are those terms that "visible" light source. "Appearance" in this case means that this point has successfully passes the test depth when rendering from the source of light - that is, they are not superseded by other objects. Therefore, all the points that are "invisible" from the position of the light source, are in the shade.

The method works in two passes: first is the rendering of the source of light - the depth values are recorded in a special buffer. And then done the normal rendering, during which the buffer is used to check if a pixel in the shade (this test is carried out in the $\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond$).

Usually the shadow cards are used with light source (such as the sun), to render the depth buffer is used orthogonal projection. However, the method is compatible with point light sources - for this instead of a depth buffer rendering is cubic maps (6 depth buffers on the sides of the Cube, ambient light source) from the perspective projection. When a large number of light sources, this technique greatly increases the load on the GPU, so in practice the shadow $\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond$ usually only for a few of the most important sources of light, depending on the nature of the $\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond$ scene.

Shadow maps are very effective - they are much faster than the shadow volume. But they have and the lack of - a strong aliasing: In other words, if you do not use a giant depth buffer, the shadows are much $\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond$. This artifact is usually eliminate the filtering (blurring) of a sample of the depth buffer core 3x3 or 5x5 - as a result of which are soft shadows without The $\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond$. This extension method of the shadow maps has received the name of the PCF (Percentage Closer Filtering).

The classic shadow cards have a limited area coverage. That is, it is impossible to make all visible objects in the scene $\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond\diamond$ qualitative shadow - increasing the size of the projection of the reduced detail and, consequently, increased aliasing. When decreasing, respectively, the remote objects fall out of the field of view" of the light source and do not cast shadows




of the most popular technique that solves this problem - cascading shadow maps (Cascaded Shadow Maps, CSM). It is the rendering of several shadow buffers instead of one, with different sizes of projections - they are called shadow cascades. Typically used 3-4 cascade. Then The desired buffer is selected depending on the coordinates of the current pixel - usually a sample between neighboring cascades to get smooth transitions. As a result of the shadow map covers almost all the visible scene: are qualitative shadows in the vicinity of the camera and distant shadows practically not noticeable to the audience - he sees only that distant objects also cast shadows, and this is enough. The main difficulty of the CSM method is the effective location of the cascades on the pyramid of visibility. The most simple solution - align with the center at the position of the camera, but in this case, the effective area of the cascades will be only about a third of their real size, since each time the viewer sees not the entire cascade, the only part of it, the corresponding horizontal angle of view camera. In modern implementations of CSM position and size of the projections of the cascades is usually picked up so that they completely fell inside the pyramid of visibility and covered it as soon as more tightly.

There is also a popular method of expansion of shadow maps - Variance Shadow Map (VSM). It depth buffer stores two values for each pixel - actually the depth and its square, this uses the buffer values with a floating point. To obtain a sample of the used buffer Chebyshev inequality. The advantage is that of the VSM-buffer can be pre-filter once, and then use for further without rendering the PCF, greatly increasing the productivity. However, the VSM brings its artifacts, the most serious of which is the so-called light-bleeding, when in the zone of shadows appear bright spots. There are several ways to fix the problem, but they either require more memory, or make the shadows not as soft as we would like.

See also the [Shadow Volume](#), the [Z-Buffer](#).

Shadow Plane



Shadow Plane | The Shadow plane

The method of the shadows,  geometry on a plane using a special matrix of transformation. The projection and then drawing a solid color, on the need to  on the Limits of the surface with the help of  buffer.

This is the easiest and fastest way to rendering of shadows, but not the most versatile - it only applies in specific cases where objects have a limited area of movement and do not cast shadows on each other.

The Volume Shadow

Shadow Volume | Volume Shadow


One of the methods of construction of shadows, namely, the creation of the object, determining the volume, inside of which points are in the shade. The result is accurate, the detailed shadows at any distance from the camera. This method of building shadows requires large drawing speed and very "heavy" for The  models. Besides, there is no effective way to render thus has now been practically is not used, as the  method of shadow maps (Shadow Map).

See also the [Shadow Map](#).



Skeletal Animation

Skeletal Animation | skeletal animation

For the first time, this technology has been used in the game Half-Life and later received a large spread in computer games.

Instead of storing the key frames (as in the case of The  animations) for each of the poses of the character, the use of skeletal animation implies one model in the neutral position and a large set of matrices that transform the various parts of the model. These matrices conditionally called bones.

In comparison with the more simple,  animation, skeletal has the following advantages:

- Reducing the amount of data stored for the animation, because it does not need to keep all options geometry for each frame of animation, it is enough to store only the position of the bones of the skeleton. This becomes more relevant in connection with the increase in the number of polygons in the models.
 - The ability to use one set of animation for different models.
 - So you can manage your bones directly, which allows to realize the  kinematics and the technology of ragdoll.
 - Allows more flexibility in mixing different animations and interpolate frames that, as a result gives a smooth and realistic animation.
 - The animation requires less computing resources of the processor and RAM.
 - On the skeleton, you can construct a dwelling was composite. For example, on the skeleton, you can "hang" at the same time, and the body of the character, and his clothing, weapons and various objects, then all of this change in dynamics.
- Disadvantages are that using skeletal animation can be done high-quality animations of flexible material such as cloth or hair, as well as the inability to complex geometry morphing objects (for example, transformation in the cube), but for such purposes, you can just use the  animation.
-

Snapping

Snapping

Automatic precise alignment for any object of a control structure along a straight line or a curve, grid, etc.

Sprite

Sprite | Sprite

Two-dimensional image of something in a 3D scene or on the screen.

Teapot

The Utah Teapot Kettle | Utah

Kettle of Utah, or kettle Newell - a computer model, which has become one of the reference objects in the community of 3D computer graphics. This is a simple, rounded, solid and partially concave mathematical model of the conventional заварного kettle.

The kettle was established in 1975, the researcher in the field of computer graphics Martin Ньюэллом, participant of the program research in computer graphics at the University of Utah. Newell needed for its work in moderately simple mathematical model of a familiar object. His wife Sandra Newell suggested to simulate their tea set, because at this moment they drank tea. Martin took The миллиметровку and pencil and зарисовал the entire set by eye, then returned to the laboratory, he manually introduced checkpoints of beziers on the handset memory of the Tektronix.

Although together with the famous kettle were digitized cup, saucer and the teaspoon, one only the kettle has achieved widespread use. It is considered, that also was modeled milkman, but data about it were lost.

The kettle is composed of 32 portions of the bicubic surface of Beziers, the coordinates of control points which are the original description of the model. Points form an array of 306 elements numbered from 1 to 306. Most of the kettle (housing) formed from 12 portions, handle - from the following four, following four portions form the spout, cover the kettle elaborated best - it took eight portions of the bicubic Bezier surfaces. And the remaining four form the bottom. These data have been widely distributed among professionals on 3D computer graphics and are widely used to demonstrate and when checking algorithms.



Tesselation

Tesselation Tessellation |

Process approximation of complex surfaces on the elementary forms. For the description of the nature of the surface of the object it is divided into various polygons. The most frequently when the graphical object is divided by triangles and четырехугольники, as they are most easily calculated and easily manipulated.

Texel

Texel Texel |

Abbreviation of two words: Texture and Element - "texture" and "element", i.e. the element of texture.

Texture

Texture Texture |

Two-dimensional image stored in the memory of a computer or graphics card in one of the pixel formats. Usually the texture is stored in the memory of the uncompressed, but modern graphics accelerators support and various compression algorithms with декомпрессией "on the fly".

Texture Filtering

Texture Filtering | Texture Filtering

One of the most important methods to improve image quality. It happens several types:

1. Pt sample - rather than type of filtration, and its absence. The texture is split up into boxes - not the most pleasant sight.
2. Bilinear Filtering is used to suppress the effect of the squares. Color of the four neighboring pixels are averaged, then two of the current unit and two of the neighboring, and so on. The squares disappear, but the picture blurred.
3. Trilinear Filtering is designed to improve the image sharpness and smooth transitions between mip-levels, working on their borders.
4. Anisotropic Filtering - removes the effect of motion blur caused by the bilinear filtration, returns the texture acutance. Requires large computational cost, but the effect is very noticeable.

the best effect is achieved by the simultaneous operation of all three types of filtration.

Technique

Technique | Technology

in computer graphics, the term "technique" is usually the technique of rendering" - the established program is a simulation of the computer graphics of those or other physical or optical phenomena in the real world. Technique usually is an extension of one of the two main methods of rendering - The dither or tracing. Sometimes these methods of rendering is also referred to as technicians.

Popular rendering technology include construction of shadows, reflections and преломлений, direct and indirect lighting, shading environment (ambient occlusion), the simulation of relief on surfaces (bump mapping), the effect of the расфокусированности (depth of field) and смазанности (motion blur), etc.

Sometimes in the rendering engines under "Equipment" means a series of transactions carried out over the graphical data for the final image - building the buffers, препроцессинг, minimizing and постпроцессинг. To respect the balance between quality and performance, as well as for compatibility with a wide range of system configurations, the engine can support different sets of such operations. For example, two common techniques in this sense are the direct rendering (direct render) and deferred rendering (deferred render), which have fundamental differences in the calculation of the lighting.

Tiling

Tiling | Тайлинг, замощение

Multiple repetition of the textures on the surface of the object.

1

Transformation | Transforming

Linear operation on geometry (usually a set of vertices).

Under the conversion in graphics engines is generally implied an affine transformation - that is, conversion, the event straight lines straight and flat surface is flat. Such changes include the shift, rotate, scale, offset and mirroring, as well as their combination in any order. But to change also applies, for example, the Perspective projection, which is not an affine transformation.

for submission of an affine transformation use, as a rule, Matrix 4x4, where the upper left подматрица specifies the rotation and scale, the right-hand column is offset, and the bottom line is always equal to [0, 0, 0, 1]. The multiplication of matrices transformation gives a matrix in which these two conversions combined (in the order in which produced multiplication). In game graphics engines typically use the following procedure for the перемножений: Transfer * * rotation scaling. The resulting matrix (also referred to as its model) is transferred to the graphics conveyor. The аффинную matrix can be inverted, i.e., calculate its inverse matrix - it will represent the inverse transform, which is very often used in computer graphics. Some of the properties of affine matrices, they can invert very efficiently.

Say that model the matrix moves the top of the model space (where the coordinates of the vertices are defined relative to the center of the model) into the world space (where the coordinates are defined relative to the absolute center of the scene). This process is the first stage of the graphics pipeline and is the vertex processor graphics card. When using shaders programr has the ability to program this stage in вершинном шейдере. In addition to the matrices, affine transformation is also possible using the кватернионов and dual кватернионов. Кватернионы are used for storage of rotation (including drawn), dual кватернионы - for simultaneous storage of rotation and transfer.

The main advantages of the кватернионов - computational efficiency and save memory (4 numbers instead of 16 matrices) and the possibility of interpolation. Кватернионы are widely used in the skeletal animation, kinematics and physics, but the graphics pipeline, they usually are not transferred - all conversion for this translated into a matrix.

See also [Matrix](#), [Vector](#).

Tree

Tree | Wood

Coherent graph without cycles. A tree with n vertices always has $n-1$ edges. Between any two vertices of the tree there is a single route. Therefore, the tree is sometimes defined as the minimum coherence graph. Top of the tree which is connected to the rib with only one top, called the sheet.

oriented wood is a graph with a dedicated top (root), which between root and any top there is the only way.

Trees are used in various mathematical models: in the theory of formal systems, describing and designing hierarchical structures (in particular, in the information systems, including databases), scheduling, etc.

Triangle Strip/Fan

Triangle Strip/Fan

If adjacent triangles, describing the surface of the figures, is not required to transmit information about all three vertices of each of them, but simply transferred as soon as the sequence of triangles, each of which shall be determined by the only one top. As a result of the reduced requirements to the width of the bandwidth.

Triangulation

Triangulation Triangulation |

partitioning method complicated polygons on the components of their triangles.
Used to need to create polygons with more than three peaks.

Tweening

Tweening | Твининг

Process of interpolation of key personnel in the вертексной animation. Твининг implies that for this model order of vertices in different frames, one and the same.

UV Coordinates

UV Coordinates | texture coordinates

In cases where a model is superimposed image (texture), the description of the vertices are added texture coordinates. Designated as a rule, they pair of U and V. The U Coordinate sets in the image pixel on the horizontal coordinate V - vertically. The value (0.0) corresponds to the upper-left corner of the texture, value (1.1) - lower right.

Vector

Vector | Vector

element of the linear space. Describes one or more numbers (vector from one number is a scalar). In computer graphics are commonly used vectors of 2, 3, and 4 numbers. The vector has a length, which is calculated using the Pythagorean theorem (the square root of the sum of the squares of all elements of the vector). Vectors are all basic arithmetic operations (Addition, subtraction, multiplication, division). The section of mathematics, studying the operations on vectors is called the vector algebra (she, in turn, is a special case of a more general directions - linear algebra).

Using a vector can be described as a point in the Euclidean space and the direction corresponding to the "look" at this point from the beginning of the coordinates. The direction is typically set to a specified, or a single vector (i.e., a vector with the length equal to 1). Any non-zero vector can be нормировать, separating the покомпонентно on its length. The direction is not equivalent to the turn - to get a full rotation, there are three mutually perpendicular vectors, forming the so-called basis. The basis is an integral part of the affine transformation matrix.

affine transformation are carried out over the space of affine vectors. Аффинный vector is a vector of 4 numbers, as symbolised by the XYZW, where XYZ - normal евклидовый vector, and the additional coordinate W allows you to express infinitely remote point (where W is equal to 0). Conventional terms have the W is equal to 1.

See also [Matrix, 1](#).

Vertex

Vertex | Top Of The

Traditionally in computer graphics under the top refers to the point of the 2D or 3D space. Set of three peaks forms a triangle - the most common entity used to build in -plane or spatial objects. Each vertex describes a particular set of parameters, such as the coordinates in the space, normal, тангент, битангент, color, texture coordinates and etc. Of these attributes are mandatory only the coordinates in space, the rest are optional. If the triangle drawing defined peaks, each of them is projected onto the screen plane. Further there is the interpolation of various parameters (the reconfigured position, color, texture coordinates, etc.), the results of which we actually see on screen.

Vertex Animation

Vertex Animation (Per-vertex Animation Morph Target Animation) |
Вершинная (вертексная) Animation

animation method, in which a sequence of key frames is a series of modified positions of the vertices of the polygon mesh. When playing tops simply interpolated from one state to another.

Вершинная animation received distribution in the Games Quake series and today is an alternative to the skeletal animation, due to some advantages. In particular, the animator has the ability to control any top separately, it is impossible when using skeletal animation. This allows, for example, the Animate clothing, face, etc. the elements of models, which are too difficult or even impossible to bind to the bones.

Вершинная animation has its disadvantages. The main of them: memory consumption, the cost of computing resources on the Calculation интерполяций. This makes the irrational use of the topmost animation in высокополигональных models.

Voxel

Volume Pixel | Воксель

Volume point. Is actually a cube in space; The воксельная printing surface is constructed of such cubes.

VSync

Vertical Synchronization | Vertical sync

Optional parameter behavior of the driver for the video card. Complimentary vertical synchronization means that after drawing the next frame, while shifting the buffers, the driver will be waiting for the next retrace the monitor, and only then will shift the screen buffers.

Image on monitors with cathode ray tube drawing a beam of electrons, which consistently отрисовывает row from left to right, then returns to the beginning of the next line (the delay of the horizontal sync), then отрисовывает the following line, etc. After the beam has got in the bottom right corner of the screen, he returns to the upper left corner (the time for which he returns, is called the latency of the vertical sync).

Why Do I need a vertical synchronization? The fact is that the delay time synchronization of the vertical retrace is ideal for switching screen buffers. If the switch buffers at any other time, the part of the image on the screen will belong to the old frame, and part is new. Because of this, the artifacts appear between frames - may become noticeable unpleasant jitter, and even at high FPS animation visually will not look smooth.

However, as in the vertical synchronization delay is done, the FPS will inevitably be less than a similar scene, but with the VSync off. This is sometimes unacceptable, for example, in a variety of graphic tests.

Weighting

Weighting Развесовка |

in the skeletal animation - the distribution of supplies the part of the surface model to this or that the bones of the skeleton. Развесовка helps improve the quality of the deformation of the анимируемой surface, bringing it to the natural.

Wireframe

Wireframe

frame displays the surface of a 3D object.

Z-Buffer

Z-Buffer | The Z-buffer

part of the graphics memory, which stores the distance from a point in space up to the screen plane (Z value). The Z-buffer determines which of the many overlapping dots closest to the plane of the observation. As well as increasing the number of bits per pixel for color in the frame buffer corresponds to a greater number of colors, available in the system image, and the number of bits per pixel in the Z-buffer corresponds to a greater number of elements. Usually, the Z-buffer has no less than 16 bits per pixel for submission to the color depth. Some of the realization of the Z-buffer is used for storing is not an integer value depth, and a floating point value from 0 to 1.

Z-Buffering

Z-Buffering | Z-buffering

Process of removing hidden surfaces, uses the values of depth, stored in the Z-buffer. Before displaying a new frame, the buffer is cleared, and the variables Z are set to large values. When rendering the object sets the Z values for each pixel: the closer the pixel is located, the less the value of Z. For each new pixel depth value is compared with the value stored in the buffer, and the pixel is recorded in the frame only if the depth is less than the stored value.

Z-Sorting

Z-Sorting | Z-sorting

Process of removing invisible surfaces using the sort the polygons in order of "bottom-up" before rendering. Thus, when rendering the upper surface of the processed last. The results of rendering are true only if objects are close together and do not intersect. The advantage of this method is no need to store values of depth. Disadvantage is the high CPU utilization and restriction on the overlapping objects.

The License Agreement: Xtreme3D

Xtreme3D, Copyright © 2016-2017, Timur Gafarov.

GLScene, Copyright © 2000-2016, The GLSTeam.

All rights reserved.

Library Xtreme3D and required for its compilation of modified code GLScene (hereinafter referred to as the "Draft Xtreme3D") are licensed under the Mozilla Public License (MPL) 1.1. You can get a copy of the full text of this license at <https://www.mozilla.org/MPL>.

This license grants you the right to freely and безвозмездно Use Draft Xtreme3D as in the non-profit and for commercial purposes, as well as copy and modify its source code under the following conditions:

- When using the source code of the Project Xtreme3D in the closed проприетарном product, the product documentation should be placed the notice on the use of the Xtreme3D with reference to the site of the project: <http://xtreme3d.narod.ru>;
- in the case of modification of the source code of the Project Xtreme3D, the modified source files should be available under the terms of the license Mozilla Public License 1.1.

Using the original (unmodified) Xtreme3D in the form of compiled dlls (xtreme3d.dll) to dynamically link with other programs does not entail the need to fulfill the above conditions.

DRAFT XTREME3D IS DISTRIBUTED ON AN "AS IS" WITHOUT ANY EXPRESS AND/OR IMPLIED WARRANTIES OF VALUE OR FITNESS FOR A PARTICULAR PURPOSE. The COPYRIGHT HOLDERS ARE NOT LIABLE FOR ANY CONSEQUENCES RESULTING FROM THE USE OF THIS SOFTWARE.

Draft Xtreme3D includes a modified project code GLScene (<http://glscene.sourceforge.net>), (which is also available on the MPL license. The project also includes third-party code libraries that are not subject to the MPL license:

- Simple Dictionary (<https://github.com/martinusso/simple-dictionary>)
- Hash Library of Ciaran McCreesh

- Decoder files in a LOD (<http://lodka3d.narod.ru>)
 - A modified version of the FreeType mapping from The Anti-Grain Geometry (<http://www.antigrain.com>)
 - CrystalLua (<https://github.com/d-mozulyov/CrystalLua>)
- Information About The правообладателях and distribution terms of code these projects you can find in the corresponding source files.
-

Xtreme3D, Copyright © 2016-2017, Timur Gafarov.
GLScene, Copyright © 2000-2016, The GLSTeam.
All rights reserved

"Xtreme3D Project" refers to Xtreme3D library and GLScene modified that it depends on.

Xtreme3D Project is distributed under the Mozilla Public License (MPL) 1.1. You can obtain a full text of this license at <https://www.mozilla.org/MPL>. This license grants you a right to freely use Xtreme3D Project both in freeware and commercial products, and to copy and modify its source code, at the following conditions:

- In case of using Xtreme3D source code in a closed-source product, an acknowledgment must be provided that the product uses Xtreme3D with a link to [Http://xtreme3d.narod.ru](http://xtreme3d.narod.ru);
- modifications made to Xtreme3D Project must be released under the MPL as well.

These requirements are only applied to the Xtreme3D Project source code. If you are using the original (unmodified) Xtreme3D in the form of a compiled dynamic library (xtreme3d.dll) for linking with an application, you are not required to fulfill conditions above.

THE XTREME3D PROJECT IS PROVIDED 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ANY OF THE AUTHORS OR COPYRIGHT MY ACCOUNT CUSTOMER SERVICE BE LIABLE FOR ANY DAMAGES CAUSED BY THE USE OR THE INABILITY TO USE OF THE XTREME3D PROJECT.

Xtreme3D Project includes modified code of GLScene (<http://glscene.sourceforge.net>) (which is also available under the MPL. Xtreme3D also includes some third-party source code files that use different licenses:

- Simple Dictionary (<https://github.com/martinusso/simple-dictionary>)
 - Ciaran McCreesh's Hash Library
 - LOD file decoder (<http://lodka3d.narod.ru>)
 - Modified FreeType binding from Anti-Grain Geometry project (<http://www.antigrain.com>).
 - CrystalLUA (<https://github.com/d-mozulyov/CrystalLUA>)
- Such files have an explicit copyright and licensing notices attached to them.

The License Agreement: ODE

Open Dynamics Engine (ODE)

Copyright © 2001-2010, Russell L. Smith

All rights reserved.

the dissemination and use of the software, ODE ("Software") in the form of source code and/or in binary form is permitted provided that the following conditions are met:

- Distribution of the Software in source code form must contain the text of this license.
- Distribution of the software in binary form must contain the text of this license in the documentation and/or other materials provided with dissemination.
- The names of the owner and his volunteers (hereinafter "owner") may not be used for advertising the products and/or services without the prior written permission.

THIS SOFTWARE extends the franchisor ON AN "AS IS" WITHOUT ANY EXPRESS AND/OR IMPLIED WARRANTIES OF VALUE OR FITNESS FOR A PARTICULAR PURPOSE. The Rightholder shall not be liable for any consequences, direct or indirect damages caused by this software, including, without limitation, damages for loss of profits, business interruption, loss of business information, or other financial losses).

Open Dynamics Engine (ODE)

Copyright © 2001-2010, Russell L. Smith.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this

list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the names of ODE's copyright owner nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE MY ACCOUNT CUSTOMER SERVICE COPYRIGHT AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The License Agreement: The FreeType

The FreeType Project

Copyright © 1996-2002, 2006, David Turner, Robert Wilhelm, Werner Lemberg.

All rights reserved.

Introduction

The FreeType Project is distributed in several archival bags; some of them may contain, in addition to the шрифтовому The FreeType engine, various tools and code, on which it is based, or related to the FreeType. This License applies to all files in these packages and is, thus, Font Engine FreeType, test programs, documentation, and make-files.

This license was based on the model of the BSD license, the artistic and the IJG (the Independent JPEG Group), each of which endorses the use of licensed free software in commercial and free software products. As a consequence, its main provisions are as follows:

- We do not warrant that the software is running (spread on an "as is"). Nevertheless, we are interested in any of the records error messages.
- You can use this software for any purpose, partially or completely, without money payments in favor of the owners.
- You cannot arrogate to itself the authorship of this software. If you use it, partially or completely, with or without modification, you must provide the notice somewhere in the documentation for your program that you used the project code of the FreeType.

We allow and endorse the inclusion of this software, with or without modification, in the composition of the commercial products. We do not provide any guarantees on the Code relating to the draft FreeType, and disclaim any liability for consequences resulting from the use of this Code.

Finally, many people ask us about the preferred form of notification of sponsorship that meets the requirements of this license. We recommend that you

use the following text:

This software contains code FreeType. Copyright © (year) Draft FreeType (www.freetype.org). All rights reserved.

Note: (year) should be replaced by the year of release versions of FreeType, you are using.

0. Determine

In this license, the terms "package", "Draft FreeType" and "Archive of FreeType" refers to the set of files, распространяемому holders (David Turner, Robert Wilhelm, Werner Lemberg), whether it is alpha, beta or final release.

Under the term "you" means the licensee (licensee), a person or organization that uses the FreeType, where "use" refers to the compilation of the source code FreeType, as well as linking programs with the FreeType library through the layout. The program, which contains the source code of the FreeType or related to the FreeType through layout, is referred to as "the program that uses the FreeType".

This License applies to all files of the Project FreeType, including all the source code, binaries and documentation unless explicitly stated otherwise. If you are not sure whether this license to a project file FreeType, contact us.

Holders Freetype Project are David Turner, Robert Wilhelm, and Werner Lemberg. All rights, except as indicated below, the reserved.

1. No warranty

The FREETYPE project extends the franchisor ON AN "AS IS" WITHOUT ANY EXPRESS AND/OR IMPLIED WARRANTIES OF VALUE OR FITNESS FOR A PARTICULAR PURPOSE. The RIGHTHOLDER SHALL NOT BE LIABLE FOR ANY CONSEQUENCES RESULTING FROM THE USE OF THIS SOFTWARE.

2. Dissemination of the

This license provides global, free, unlimited and irrevocable right to use, compile, perform, display, copy, modify, distribute and sublicense The FreeType Project (as in the original, and in binary form) and any derivative works from it, as well as to provide these rights, partially or completely, to third parties, subject

to the following conditions:

- Distribution of the original or modified the source code FreeType should be accompanied by the unchanged text of this license. Any additions, delete, and modify the source code FreeType should be described in the accompanying documentation. The copyright notice must appear in all copies of the source code FreeType.
- Dissemination of the FreeType in binary form must contain a notice that product includes FreeType. We welcome the reference to the URL address of the web site of the Project FreeType, but this is not a mandatory requirement.

These terms apply to any software that uses the FreeType or is a derivative work based on the FreeType code. If you are using the FreeType, please notify us. Remuneration in our favor is not required.

3. Advertising

the names of the sponsors of the Freetype, as well as his, see may not be used for commercial advertising without their written permission.

We recommend (but do not require), use one of the following phrases to refer to this software in the documentation or promotional materials: "The Project FreeType, The FreeType Engine", "The Library FreeType" or "The FreeType Package".

Because you do not sign up under this license, you are not required to accept its terms and conditions. However, since the project is the FreeType masterpiece that is protected by copyright law, only the license (or the other, concluded with right holders) gives you the right to use, distribute and modify the FreeType. Thus, using, distributing or modifying the FreeType, you acknowledge and agree with the terms of this license.

4. Contact Information

There are two mailing list related to the FreeType:

- **freetype@nongnu.org**. On the overall use of FreeType, as well as innovations in the library and distribution. If you need technical support, and you have not found help in the documentation, ask a question in this list of mailing lists.
- **freetype-devel@nongnu.org**. Devoted to the elaboration of FreeType, discussion of errors, the engine porting to other platforms, design, license, etc.

Site FreeType is located at <http://www.freetype.org>.

The FreeType Project

Copyright © 1996-2002, 2006, David Turner, Robert Wilhelm, Werner Lemberg.

All rights reserved.

Introduction

The FreeType Project is distributed in several archive packages; some of them may contain, in addition to the FreeType font engine, various tools and contributions which rely on, or relate to, the FreeType Project.

This License applies to all files found in such packages, and which do not fall under their own explicit license. The license affects thus the FreeType font engine, the test programs, documentation and makefiles, at the very least.

This license was inspired by the BSD, artistic, and ijk (the Independent JPEG Group) licenses, which all encourage inclusion and use of free software in commercial and freeware products alike. As a consequence, its main points are that:

- We don't promise that this software works. However, we will be interested in any kind of bug reports. ('As is' distribution)
- You can use this software for whatever you want, in parts or full form, without having to pay us. ('Royalty-free' usage)
- You may not pretend that you wrote this software. If you use it, or only parts of it, in a program, you must acknowledge somewhere in your documentation that you have used the FreeType code. ('Credits')

We specifically permit and encourage the inclusion of this software, with or without modifications, in commercial products. We disclaim all warranties covering The FreeType Project and assume no liability related to The FreeType Project.

Finally, many people asked us for a preferred form for a credit/disclaimer to use in compliance with this license. We thus encourage you to use the following text:

Portions of this software are copyright © (year) of The FreeType Project (www.freetype.org). All rights reserved.

Please replace (year) with the value from the FreeType version you actually use.

0. Definitions

Throughout this license, the terms 'package', 'The FreeType Project', and 'The FreeType archive' refers to the set of files originally distributed by the authors (David Turner, Robert Wilhelm, and Werner Lemberg) as the 'FreeType Project', be they named as alpha, beta or final release.

'You' refers to the licensee, or person using the project, where 'using' is a generic term including compiling the project's source code as well as linking it to form a 'program' or 'executable'. This program is referred to as 'a program using the FreeType engine'.

This License applies to all files distributed in the original FreeType Project, including all source code, binaries and documentation, unless otherwise stated in the file in its original, unmodified form as distributed in the original archive. If you are unsure whether or not a particular file is covered by this license, you must contact us to verify this.

The FreeType Project is copyright (C) 1996-2000 by David Turner, Robert Wilhelm, and Werner Lemberg. All rights reserved except as specified below.

1. No Warranty

THE FREETYPE PROJECT IS PROVIDED 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ANY OF THE AUTHORS OR COPYRIGHT MY ACCOUNT CUSTOMER SERVICE BE LIABLE FOR ANY DAMAGES CAUSED BY THE USE OR THE INABILITY TO USE, OF THE FREETYPE PROJECT.

2. Redistribution

This license grants a worldwide, royalty-free, perpetual and irrevocable right and license to use, execute, perform, compile, display, copy, create derivative works of, distribute and sublicense the FreeType Project (in both source and object code forms) and derivative works thereof for any purpose; and to authorize others to exercise some or all of the rights granted herein, subject to the following conditions:

- Redistribution of source code must retain this license file ('FTL.txt') unaltered; any additions, deletions or changes to the original files must be clearly indicated in accompanying documentation. The copyright notices of the unaltered original files must be preserved in all copies of source files.
- Redistribution in binary form must provide a disclaimer that states that the software is based in part of the work of the FreeType Team, in the distribution documentation. We also encourage you to put an URL to the FreeType web page in your documentation, though this isn't mandatory.

These terms apply to any software derived from or based on the FreeType Project, not just the unmodified files. If you use our work, you must acknowledge us. However, no fee need be paid to us.

3. Advertising

Neither the FreeType authors and contributors nor you shall use the name of the other for commercial, advertising, or promotional purposes without specific prior written permission.

We suggest, but do not require, that you use one or more of the following phrases to refer to this software in your documentation or advertising materials: 'FreeType Project', 'FreeType Engine', 'FreeType library', or 'FreeType Distribution'.

As you have not signed this license, you are not required to accept it. However, as the FreeType Project is copyrighted material, only this license, or another one contracted with the authors, grants you the right to use, distribute, and modify it. Therefore, by using, distributing, or modifying the FreeType Project, you indicate that you understand and accept all the terms of this license.

4. Contacts

There are two mailing lists related to FreeType:

- **freetype@nongnu.org**. Discusses general use and applications of FreeType, as well as future and wanted additions to the library and distribution. If you are looking for support, start in this list if you haven't found anything to help you in the documentation.
- **freetype-devel@nongnu.org**. Discusses bugs, as well as engine internals, design issues, specific licenses, porting, etc.

Our home page can be found at **<http://www.freetype.org>**.

Links

Xtreme3D:

<http://xtreme3d.narod.ru> - site Xtreme3D <http://offtop.ru/xtreme3d> - the accompanying forum

<https://github.com/xtreme3d> - The Xtreme3D on GitHub

GLScene:

<http://glscene.sourceforge.net> - official site of The GLScene

OpenGL:

<http://www.opengl.org> - official site of the OpenGL

<http://pmg.org.ru/nehe> - English translation of lessons on OpenGL from the famous Neon Helium (NeHe)

And more...

Editorial expresses its gratitude to the following resources:

<http://www.gamedev.ru>

<http://gcup.ru>