# Lighthouse3d.com

MOSTLY ABOUT 3D GRAPHICS



## GLUT Tutorial

GLUT stands for OpenGL Utility Toolkit. Mark J. Kilgard implemented it to enable the construction of OpenGL applications that are truly window system independent. Thanks to GLUT, we can write applications without having to learn about X windows or Microsoft's own window system. Kilgard implemented the version for X windows, and later Nate Robins ported it to Microsoft Windows. Thanks to both, you did a great job.

With GLUT you can open a window for OpenGL rendering with 5 lines of code! Another 3 or 4 lines and you can a keyboard and mouse with your application. GLUT really makes things simple, hence it is very usefull to learn and to build small applications. Although GLUT is not being maintained anymore it still serves its purpose.

The GLUT distribution comes with lots and lots of examples so after you read through the basics in here you'll have plenty of material to go on. Check out the GLUTs page.

In this tutorial I'll introduce you to the basics of building an application using GLUT. This tutorial won't introduce fancy visual effects in order to keep the code as simple as possible. I'll use OpenGL 2.0 since it is much simpler and avoids complicating the core subject of the tutorial: GLUT.

There are open source versions of GLUT, such as freeGLUT and OpenGLUT. They all kept the API so 99.9% of what will be presented in this tutorial is still valid. Nonetheless these new versions do have some extensions that make it worth a try. Check out the extensions in freeGLUT here.

Please do comment if something is not completely clear. Your feedback is important.

## Index

**Basics**

- Setup
- Initialization
- Resizing the Window
- Animation

**Input**

- Keyboard
- Moving the Camera I
- Advanced Keyboard
- Moving the Camera II
- The Code So Far
- Mouse
- Moving the Camera III
- The Code So Far II

**Pop-up Menus**

- Popup Menus Basics
- Sub Menus
- Modifying a Menu
- Swapping Menus
- The Code So Far III

**Fonts**

- Bitmap Fonts
- The Code So Far IV
- Bitmaps and the Ortho View
- Stroke Fonts

**Extras**

- Frames per Second
- The Code So Far V
- GLUT Game Mode
- The Code So Far VI

**Subwindows**

- Creating and Destroying Subwindows
- Reshaping Subwindows
- Rendering to Subwindows
- Subwindows Code

**Avoiding the Idle Func**

- glutPostRedisplay
- The Code So Far VII

**Source Code and Projects**

**15 Responses to "GLUT Tutorial"**

1. **Bobo** says:
02/12/2013 at 10:52 PM

Can you do a tutorial on texturing?

Reply

2. **Jin1974** says:
05/09/2012 at 11:43 AM

lighthouse3d is terrific. There's often all the appropriate info at the suggestions of my fingers. Thank you and maintain up the superior work!

Reply

3. **ibreezy** says:
12/07/2012 at 11:11 PM

Hello there, The second sentence in the second paragraph is not clear. . . *"Another 3 or 4 lines and you can a keyboard and mouse with your application."*

Reply

4. **SARWAR JAHAN** says:
19/04/2012 at 4:27 PM

I want to create buttons in a OpenGL window, i,e, in a window some buttons will appear like START GAME, SCORE, EXIT etc.
And whenever we click on that button respective event should be triggered. But the program should be written using GLUT library only..
please anybody help me…

Reply

5. **anonymous** says:
20/02/2012 at 12:30 PM

Hi there, Thank you very much for taking the time to write this. Cheers, it's big help. thanks again.

Reply

6. **SILVERMJAX** says:
13/12/2011 at 5:24 PM

YEAH VERY HELPFULL!

Reply

7. **GorflGorfx** says:
08/12/2011 at 11:34 PM

Shader Examples 0 comments | 11,288 views

Keyboard Example: Moving the Camera 0 comments | 11,203 views

OpenGL Framebuffer Objects 0 comments | 10,785 views

Importing 3D Models with Assimp 0 comments | 10,671 views

Keyboard 0 comments | 10,306 views

**Links**

- AMD Developer Central
- Codeflow
- Flipcode
- G-Truc Creation
- GameDev.net
- Geeks3D
- Hugo Elias' Site
- Humus 3D
- Iñigo Quilez Site
- Khronos Group
- Nehe
- nVidia Developer
- OpenGL.org
- Paul Bourke's Site
- Real-Time Rendering

**Google Ads**

# Lighthouse3d.com

MOSTLY ABOUT 3D GRAPHICS



## Setup Basics

Prev: Index Next: Initialization

**What you need**

In order to write applications with GLUT you should have the latest version – 3.7.6. You can find the windows binary files (h lib and dll), as well as GLUT's source code in Nate Robins site.

In order to write a C application using GLUT you'll need three files:

- glut.h – This is the file you'll have to include in your source code. The common place to put this file is in the *gl* folder which should be inside the include folder of your system.
- glut32.lib (Windows version) – This file must be linked to your application so make sure to put it your *lib* folder.
- glut32.dll (Windows) – You could place the dll file in your exe's folder.

**A clean house**

If you're serious about OpenGL, sooner or latter you'll start adding other libs to your application. Having them all in one place is a good idea to keep everything updated, and even in the case where you want to move all your stuff to other computer.

Hence my suggestion is to create a folder to keep all these files. This folder should have three sub folders, namely *includes*, *libs* and *dlls*. Then place all the files mentioned above in these sub folders. Whenever you need more libs just keep adding them to this folder structure.

**Setting up in Visual Studio 2010**

I'm not an expert in VS, so this is probably not the most efficient way of doing things. It should work if you follow these instructions step by step. If you know of a better/faster way of doing things let me know and I'll update the tutorial.

First make sure you start VS in C or C++ mode. Select File->New Project and a dialog opens. Select "Win32 Console Application" and give a name to your project. Press OK to move on.

In the next dialog press "Next". In the next screen select "Console Application and check "Empty Project". Press "Finish" and your project is ready.

Now go to solution explorer. If you can't see it go to "View" and select "Solution Explorer". The project you just created has a set of 'folders'. Right click on "Source Files" and select "Add->New Item". In the dialog for the new item select C++ or C File, give it a name and you are ready to start coding.

After you got the code right and your app executes you may notice that two windows are opened: your window with the OpenGL rendering, and a console window. The console window is useful in case you want to print something out with printf. However, at the end of the day you may want to get rid of it.

Ricardo has posted a comment on another page with a solution to get rid of the console window. Thanks Ricardo. Go to:

*ProjectProperties » Linker » System » SubSystem*

and set the property to

```
Windows (/SUBSYSTEM:WINDOWS)
```

Then go to:

*Project Properties » Linker » Command Line*

and add:

/ENTRY:mainCRTStartup

to Additional Options.

A final note, you must tell VS where it can find the include and lib files for GLUT. Right click on the project in Solution Explorer window and select "Properties". In the left side if the properties dialog select VC++ Directories and add to the right side your path to the include and lib files.

**Notes on Using OpenGL with VS.NET**

Many people have experienced a small problem when using VS.NET to build OpenGL applications: a compiler error that, as far as I know, seems to exist only in these compilers (VS 2003 and 2005, VC 6.0 is OK). Here is the actual text generated by VS 2005:

c:\programs\microsoft visual studio 8\vc\include\stdlib.h(406) :

```
        error C2381: 'exit' : redefinition; __declspec(noreturn) differs
        c:\opengl\toolkits\includes\gl\glut.h(146) : see declaration of '
```

This problem seems to be caused by the inclusion of *glut.h* before *stdlib.h*. Reversing the order solves the problem. The following code causes the compiler error:

```
#include <GL/glut.h>
#include <stdlib.h>
```

But when including first *stdlib.h* the problem disappears:

```
#include <stdlib.h>
#include <GL/glut.h>
```

Do let me know if you have any tips so that I may include them in here and save time to all those who starting to learn OpenGL.

OK, so all is set, lets learn how to write GLUT applications. If there is anything that is not quite clear please let me know and I'll try to improve it. Your feedback is important.

Prev: Index Next: Initialization

**10 Responses to "Setup Basics"**

1. **Mihai** says:
07/01/2013 at 11:11 AM

You may add
#define _CRT_TERMINATE_DEFINED
before
#include
to get rid of "error C2381: 'exit' : redefinition;"

Reply

2. **Nathan** says:
20/02/2012 at 5:05 PM

Thanks for the instruction on removing the console window! That was huge for me. I don't know why this isn't in more tutorials as I think most people would be using Microsft Visual Studio.

Reply

3. **Abhishek Dey** says:
17/11/2011 at 6:42 PM

Very Nice Tutorials,

And use the freeglut available @ http://freeglut.sourceforge.net/index.php.
OR
download the source code, if you are thinking of custom made DLL's such as for 64-bit!

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,186 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models with Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,272 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

## Initialization

Prev: Setup Basics Next: Reshape

In this section we're going to build the main function of our application. The main function will perform the required initializations and start the event processing loop.

The first part of our main function code will initialize GLUT and create the window.

After GLUT enters the event processing loop it gains control of the application. GLUT will wait for an event to occur and then will check if there is a function to process it.

So before GLUT enters its event processing loop we need to tell which functions GLUT must call for each event we care to process.

At this point you might wonder what is an event. An event is something like a key pressed, a mouse move, a window that needs to be painted, or a window that was resized, or … The list can grow to become quite large. We will start by only dealing with the paint event.

Telling GLUT which function to call back when an event occurs is registering the callback function. For each event type GLUT provides a specific function to register the callback function.

The skeleton of our main function is going to start as:

```
int main(int argc, char **argv) {

        // init GLUT and create window

        // register callbacks

        // enter GLUT event processing cycle

}
```

**Init GLUT and create window**

All the functions in GLUT have the prefix *glut*, and those which perform some kind of initialization have the prefix *glutInit*. The first thing you must do is call the function *glutInit*.

---

void glutInit(int *argc, char **argv);

Parameters:

- argc – A pointer to the *unmodified* argc variable from the main function.
- argv – A pointer to the *unmodified* argv variable from the main function.

---

After initializing GLUT itself, we're going to define our window. First we establish the window's position, i.e. its top left corner. In order to do this we use the function *glutInitWindowPosition*.

---

void glutInitWindowPosition(int x, int y);

Parameters:

- *x* – the number of pixels from the left of the screen.
  -1 is the default value, meaning it is up to the
  window manager to decide where the window will appear.
  If not using the default values then you should pick
  a positive value, preferably one that will fit in your screen.
- *y* – the number of pixels from the top of the screen.
  The comments mentioned for the *x* parameter
  also apply in here.

---

Note that these parameters are only a suggestion to the window manager. The window returned may be in a different position, although if you choose them wisely you'll usually get what you want.

Next we'll choose the window size. In order to do this we use the function *glutInitWindowSize*.

---

void glutInitWindowSize(int width, int height);

Parameters:

- *width* – The width of the window
- *height* – the height of the window

---

Again the values for *width* and *height* are only a suggestion, so avoid choosing negative values.

Then you should define the display mode using the function *glutInitDisplayMode*.

---

void glutInitDisplayMode(unsigned int mode)

Parameters:

- *mode* – specifies the display mode

---

The mode parameter is a Boolean combination (OR bit wise) of the possible predefined values in the GLUT library. You use *mode* to specify the color mode, and the number and type of buffers.

The predefined constants to specify the color model are:

- GLUT_RGBA or GLUT_RGB – selects a RGBA window. This is the default color mode.
- GLUT_INDEX – selects a color index mode.

The display mode also allows you to select either a single or double buffer window. The predefined constants for this are:

- GLUT_SINGLE – single buffer window
- GLUT_DOUBLE – double buffer window, required to have smooth animation.

There is more, you can specify if you want a window with a particular set of buffers. The most common are:

- GLUT_ACCUM – The accumulation buffer
- GLUT_STENCIL – The stencil buffer
- GLUT_DEPTH – The depth buffer

So, suppose you want a RGB window, with double buffering, and a depth buffer. All you have to do is to OR all the respective constants in order to create the required display mode.

```
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT DEPTH);
```

After all the above steps, the window can be created with *glutCreateWindow*.

---

int glutCreateWindow(char *title);

Parameters:

- *title* – sets the window title

---

The return value of *glutCreateWindow* is the window identifier. You can use this identifier later within

GLUT but this is outside of the scope of this section.

So now here is a little bit of code to perform all the initializations:

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

int main(int argc, char **argv) {

	// init GLUT and create Window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D- GLUT Tutorial");

	return 1;

}
```

Note the include statements at the beginning of the code. The required include file comes with the GLUT distribution, however their location is different on MacOS systems, hence checking is required if the code is to be cross-platform.

**The render function and callback registration**

If you run this code, you'll obtain an empty black console window but no OpenGL window. There are two things left to do before we are ready to render something. The first is to tell GLUT what is the function responsible for the rendering. This function will be called by GLUT everytime the window needs to be painted.

Lets create an example function for the rendering. The function presented bellow will clear the color buffer and draw a triangle.

```
void renderScene(void) {

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glBegin(GL_TRIANGLES);
		glVertex3f(-0.5,-0.5,0.0);
		glVertex3f(0.5,0.0,0.0);
		glVertex3f(0.0,0.5,0.0);
	glEnd();

	glutSwapBuffers();
}
```

The name of this function is up to you. However, now you must tell GLUT that it should use the function we just wrote for the rendering. This is called registering a callback. GLUT will call the function you choose whenever rendering is required.

So lets tell GLUT that the function *renderScene* should be used whenever the window requires to be painted. GLUT has a function that takes as a parameter the name of the function to use when redrawing is needed. Note: the function you supply as a parameter is also called the first time the window is created. The syntax is as follows:

---

void glutDisplayFunc(void (*funcName)(void));

Parameters:

---

**GLUT event processing loop**

One last thing is missing, telling GLUT that we're ready to get in the event processing loop. GLUT provides a function that gets the application in a never ending loop, always waiting for the next event to process. The GLUT function is *glutMainLoop*, and the syntax is as follows:

```
void glutMainLoop(void)
```

**All together now**

The code so far is presented bellow. If you try to run this code you'll get a console window, and a few instants after the OpenGL window with a white triangle, hopefully at the desired position and with the desired size.

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

void renderScene(void) {

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glBegin(GL_TRIANGLES);
		glVertex3f(-0.5,-0.5,0.0);
		glVertex3f(0.5,0.0,0.0);
		glVertex3f(0.0,0.5,0.0);
	glEnd();

	glutSwapBuffers();
}

int main(int argc, char **argv) {

	// init GLUT and create Window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D - GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);

	// enter GLUT event processing cycle
	glutMainLoop();

	return 1;
}
```

Prev: Setup Basics Next: Reshape

**51 Responses to "Initialization"**

1. **Anthony** says:
07/01/2014 at 10:39 PM

When I debug any kind of C++ project in Visual Studio 2010 it says it needs dxerr.lib.
The only reason I can put towards it doing this is because in Linker, Additional Dependencies, Inherited Values, dxerr.lib and a few other Direct X libraries are there and I don't seem able to remove them, can you maybe shred some light on this? I've had a miserable day of going through 4 different kinds of Glut getting this to work.
Awesome website by the way, everything is very well documented.

Reply

2. **David** says:
08/11/2013 at 11:42 AM

Hey, I'm having a problem, my code is exactly as yours but, while I see a new app pop up in Dock(using Mac), nothing is drawn on the screen. It's responding and all, the console is working and not displaying any errors, but there's just nothing being drawn.

I'm using XCode by the way.

Reply

**ARF** says:
20/11/2013 at 11:42 PM

# Lighthouse3d.com

MOSTLY ABOUT 3D GRAPHICS



## Preparing the window for a reshape

Prev: Initialization Next: Animation

Run the app you've just created. You'll see two windows: a console window and the OpenGL window. Now resize the window so that the height no longer matches the width. The triangle gets distorted. This occurs because we're not setting the perspective correctly. By default, the perspective assumes that the ratio width/height is 1 and draws accordingly. So when the ratio is changed, the perspective gets distorted. Therefore, every time the ratio changes the perspective needs to be recomputed.

GLUT provides a way to define which function should be called when the window is resized, i.e. to register a callback for recomputing the perspective. Furthermore, this function will also be called when the window is initially created, so that even if you're initial window is not square things will look OK.

GLUT registers the callback functionwhen you call *glutReshapeFunc*.

void glutReshapeFunc(void (*func)(int width, int height));

Parameters:

- *func* – The name of the function that will be responsible for setting the correct perspective when the window changes size.

So what we must do is to go back to the main function we defined in the previous section, and add a call to *glutReshapeFunc*. Lets call our own function to take care of window resizes *changeSize*. The code for the main function with the call to *glutReshapeFunc* added in is:

```
int main(int argc, char **argv) {

	// init GLUT and create window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D- GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);

	// Here is our new entry in the main function
	glutReshapeFunc(changeSize);

	// enter GLUT event processing cycle
	glutMainLoop();

	return 1;
}
```

The next thing we need to do, is to define the function that we'll take care of the perspective. As seen by the syntax of *glutReshapeFunc*, the *changeSize* function has two arguments, these are the new width and height, respectively, of the client area of the window, i.e. without the window decorations.

```
void changeSize(int w, int h) {

	// Prevent a divide by zero, when window is too short
	// (you cant make a window of zero width).
	if(h == 0)
		h = 1;
```

```
        float ratio = 1.0* w / h;

        // Use the Projection Matrix
        glMatrixMode(GL_PROJECTION);

        // Reset Matrix
        glLoadIdentity();

        // Set the viewport to be the entire window
        glViewport(0, 0, w, h);

        // Set the correct perspective.
        gluPerspective(45,ratio,1,1000);

        // Get Back to the Modelview
        glMatrixMode(GL_MODELVIEW);
}
```

A few functions we're introduced in this piece of code, so let's go into a bit of detail in here before we all get lost. The first step was to compute the ratio between the *width* and the *height*. Note that, in order for this to be done correctly, we must take care of the case when the *height* of a window is actually zero, to prevent a division by zero.

We then set the current matrix to be the projection matrix. This is the matrix that defines the viewing volume. We then load the identity matrix to initialize it. Afterwards, we set the viewport to be the whole window, with the function *glViewport*. You can try with different values to see what you come up with, the first two parameters are the bottom left corner, and the last two are the width and height of the viewport. Note that these coordinates are relative to the client area of the window, not the screen. If you do try with different values then don't forget that the ratio computed above should also use the new *width* and *height* values.

The *gluPerspective* function is part of another library for OpenGL, the OpenGL Utility Library, or GLU. GLU is a standard component of the implementation of OpenGL. The *gluPerspective* function establishes the perspective parameters. The first one defines the field of view angle in the yz plane, the *ratio* defines the relation between the *width* and *height* of the *viewport*. The last two parameters define the *near* and *far* clipping planes. Anything closer than the *near* value, or further away than the *far* value, will be clipped away from the scene. Beware with these settings, or you may end up not seeing anything at all.

Finally, we tell OpenGL that all matrix operations that follow will use the *modelview matrix*. This is just to be on the safe side. Most operations, such as setting the camera and transforming objects will use this matrix. The idea is to have always the *modelview* matrix as default.

**The code so far**

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

void changeSize(int w, int h) {

        // Prevent a divide by zero, when window is too short
        // (you cant make a window of zero width).
        if (h == 0)
                h = 1;

        float ratio =  w * 1.0 / h;

        // Use the Projection Matrix
        glMatrixMode(GL_PROJECTION);

        // Reset Matrix
        glLoadIdentity();

        // Set the viewport to be the entire window
        glViewport(0, 0, w, h);

        // Set the correct perspective.
        gluPerspective(45,ratio,1,100);

        // Get Back to the Modelview
        glMatrixMode(GL_MODELVIEW);
}
```

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,186 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models with Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,289 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

Google Ads

```
void renderScene(void) {

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glBegin(GL_TRIANGLES);
		glVertex3f(-2,-2,-5.0);
		glVertex3f(2,0.0,-5.0);
		glVertex3f(0.0,2,-5.0);
	glEnd();

	glutSwapBuffers();
}

int main(int argc, char **argv) {

	// init GLUT and create window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D - GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);
	glutReshapeFunc(changeSize);

	// enter GLUT event processing loop
	glutMainLoop();

	return 1;
}
```

Prev: Initialization Next: Animation

**41 Responses to "Preparing the window for a reshape"**

1. Scott says:
14/02/2014 at 10:47 PM

I was running into the gluPerspective issue (not defined) and I ended up adding "-lglu32" to the end of my compilation line which fixed it. Hopefully this will help someone.

Reply

2. Noopur says:
12/07/2013 at 12:08 PM

Hi,
I need to know how can I preserve the size of the object in the window even after the window is resized.

Reply

3. Peon says:
17/05/2013 at 3:13 PM

Hi all, I am new bie with OpenGL. Could you please tell me how the procedure changeSize defines the parameters w and h when it is called by glutReshapeFunc(changeSize) ?
If w and h are global parameters, then where did we define them after calling changeSize?

Reply

ARF says:
03/06/2013 at 2:22 AM

Hi,

We don't define those parameters. That function is called by GLUT whenever the window size changes, and the function is called with the new window dimensions.

Reply

# Lighthouse3d.com

MOSTLY ABOUT 3D GRAPHICS



## Animation

Prev: Reshape Next: Keyboard

OK, so far so good. We have an OpenGL window with a white triangle. Nothing very exciting, but hey, its a start. Now to complete this part of the GLUT tutorial lets have that triangle spinning.

The first thing we must do is to tell GLUT that when the application is idle, the render function should be called. This causes GLUT to keep calling our rendering function therefore enabling animation. GLUT provides a function, *glutIdleFunc*, that lets you register a callback function to be called when the application is idle.

void glutIdleFunc(void (*func)(void));

Parameters:

- *func* – The name of the function that will be called whenever the application is idle.

In our case, when the application is idle we want to call the previously defined function that does the actual rendering: *renderScene*. OK, so the main function now looks like this:

```
int main(int argc, char **argv) {

	// init GLUT and create window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D- GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);
	glutReshapeFunc(changeSize);

	// here is the idle func registration
	glutIdleFunc(renderScene);

	// enter GLUT event processing cycle
	glutMainLoop();

	return 1;
}
```

Afterwards, we go and change the rendering itself. First lets declare a floating point variable *angle*, and initialize it to 0.0 . Then we add the necessary stuff to the *renderScene* function.

```
float angle = 0.0f;

void renderScene(void) {

	// Clear Color and Depth Buffers
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
        // Reset transformations
        glLoadIdentity();
        // Set the camera
        gluLookAt(      0.0f, 0.0f, 10.0f,
                        0.0f, 0.0f,  0.0f,
                        0.0f, 1.0f,  0.0f);

        glRotatef(angle, 0.0f, 1.0f, 0.0f);

        glBegin(GL_TRIANGLES);
                glVertex3f(-2.0f,-2.0f, 0.0f);
                glVertex3f( 2.0f, 0.0f, 0.0);
                glVertex3f( 0.0f, 2.0f, 0.0);
        glEnd();

        angle+=0.1f;

        glutSwapBuffers();
}
```

**A note on double buffering**

As you recall, when we set the display mode in our main function we requested double buffering. This feature provides two buffers for display. The buffer being shown is the front buffer, and the other buffer is the back buffer. It is in the latter buffer that drawing takes place. When we're done sending drawing commands we ask to swap the buffers, i.e. once the driver is done rendering the back buffer will become the front buffer, and the previous front buffer will become the back buffer, where the rendering of the next frame will take place.

The *glutSwapBuffers* function cause the front and back buffers to switch thereby showing what was previously drawn in the back buffer. The syntax is as follows:

```
void glutSwapBuffers();
```

**The code so far**

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

void changeSize(int w, int h) {

        // Prevent a divide by zero, when window is too short
        // (you cant make a window of zero width).
        if (h == 0)
                h = 1;

        float ratio =  w * 1.0 / h;

        // Use the Projection Matrix
        glMatrixMode(GL_PROJECTION);

        // Reset Matrix
        glLoadIdentity();

        // Set the viewport to be the entire window
        glViewport(0, 0, w, h);

        // Set the correct perspective.
        gluPerspective(45.0f, ratio, 0.1f, 100.0f);

        // Get Back to the Modelview
        glMatrixMode(GL_MODELVIEW);
}

float angle = 0.0f;
```

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,186 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models w ith Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,289 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

Google Ads

```
void renderScene(void) {

	// Clear Color and Depth Buffers
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	// Reset transformations
	glLoadIdentity();
	// Set the camera
	gluLookAt(	0.0f, 0.0f, 10.0f,
				0.0f, 0.0f,  0.0f,
				0.0f, 1.0f,  0.0f);

	glRotatef(angle, 0.0f, 1.0f, 0.0f);

	glBegin(GL_TRIANGLES);
		glVertex3f(-2.0f,-2.0f, 0.0f);
		glVertex3f( 2.0f, 0.0f, 0.0);
		glVertex3f( 0.0f, 2.0f, 0.0);
	glEnd();

	angle+=0.1f;

	glutSwapBuffers();
}

int main(int argc, char **argv) {

	// init GLUT and create window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D- GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);
	glutReshapeFunc(changeSize);
	glutIdleFunc(renderScene);

	// enter GLUT event processing cycle
	glutMainLoop();

	return 1;
}
```

Just copy the code above to your project and try it, or alternatively go to the download section. Isn't it great? OK, ok…but I warned you, no fancy stuff in here, keeping the code to the minimum and focusing on GLUT!

Prev: Reshape Next: Keyboard

**22 Responses to "Animation"**

1. **Terminator.** says:
22/08/2013 at 5:18 PM

Hey Tutorial is awesome, really i learnt many things from scratch. But few things are still missing,i.e., will you please explain me how to apply a texture in a simple way. Till now we are only dealing with the normal primitives…..
Expecting Texturing concept soon…

Reply

2. **gutierrez** says:
02/08/2013 at 2:24 PM

Thanks for the tutorial. But what is idle? I mean, when does the OpenGL/GLUT enters in an idle state? And, instead of registering an idle function, why don't you add glutPostRedisplay() at the end of renderScene()?

Reply

# Keyboard

Prev: Animation Next: Moving the Camera I

GLUT allows us to build applications that detect keyboard input using either the “normal” keys, or the special keys like F1 and Up. In this section we’ll see how to detect which key was pressed, what further information we get from GLUT, and how to deal with it.

As you have probably noticed by now, whenever you want to take control of the processing of an event you have to tell GLUT in advance which function is going to perform such task. Up until now we used GLUT to tell the windows system which functions we wanted to do the rendering when the window needed to be repainted, which function to call when the system was idle, and which function to call when the window was resized.

We must do the same thing for keyboard events. We must notify GLUT which function(s) will perform the required processing when a key is pressed.

GLUT provides two functions to register callbacks for keyboard events that occur when you press a key. The first one, *glutKeyboardFunc*, is used to tell the windows system which function we want to process the “normal” key events. By “normal” keys, we mean letters, numbers, anything that has an ASCII code. The syntax for this function is as follows:

void glutKeyboardFunc(void (*func) (unsigned char key, int x, int y));

Parameters:

- func – The name of the function that will process the “normal” keyboard events. Passing NULL as an argument causes GLUT to ignore “normal” keys.

The function used as an argument to *glutKeyboardFunc* needs to have three arguments. The first provides the ASCII code of the key pressed, the remaining two arguments provide the mouse position when the key is pressed. The mouse position is relative to the top left corner of the client area of the window.

A possible implementation for this function is to provide a way out of the application when the user presses the ESCAPE key. Note that when the *glutMainLoop* function was presented we mentioned that it was an infinite loop, i.e. it never returns. The only way out of this loop is to call the system *exit* function. So that’s exactly what our function will do, when the user presses escape it calls the system *exit* function causing the application to terminate (remember to include stdlib.h in the source code). Next we present the function code:

```
void processNormalKeys(unsigned char key, int x, int y) {

        if (key == 27)
                exit(0);
}
```

Note that we are using exactly the same signature as the one specified in the syntax of *glutKeyboardFunc*. If you don’t do this you’ll get an error when compiling this in VC, and we don’t want that, do we?

OK, ready to move on? Lets tackle the special keys now. GLUT provides the function *glutSpecialFunc* so that you can register your function for special key event processing. The syntax for this function is as follows:

void glutSpecialFunc(void (*func) (int key, int x, int y));

Parameters:

- func – The name of the function that will process the special keyboard events. Passing NULL as an argument causes GLUT to ignore the special keys.

---

We're going to write a function that changes the color of our triangle when some of the special keys are pressed. This function will paint the triangle using red if F1 is pressed, green if F2 is pressed, and blue if F3 is pressed.

```
void processSpecialKeys(int key, int x, int y) {

	switch(key) {
		case GLUT_KEY_F1 :
				red = 1.0;
				green = 0.0;
				blue = 0.0; break;
		case GLUT_KEY_F2 :
				red = 0.0;
				green = 1.0;
				blue = 0.0; break;
		case GLUT_KEY_F3 :
				red = 0.0;
				green = 0.0;
				blue = 1.0; break;
	}
}
```

The GLUT_KEY_* are predefined constants in *glut.h*. The full set of constants is presented next:

```
GLUT_KEY_F1             F1 function key
GLUT_KEY_F2             F2 function key
GLUT_KEY_F3             F3 function key
GLUT_KEY_F4             F4 function key
GLUT_KEY_F5             F5 function key
GLUT_KEY_F6             F6 function key
GLUT_KEY_F7             F7 function key
GLUT_KEY_F8             F8 function key
GLUT_KEY_F9             F9 function key
GLUT_KEY_F10            F10 function key
GLUT_KEY_F11            F11 function key
GLUT_KEY_F12            F12 function key
GLUT_KEY_LEFT           Left function key
GLUT_KEY_RIGHT          Right function key
GLUT_KEY_UP             Up function key
GLUT_KEY_DOWN           Down function key
GLUT_KEY_PAGE_UP        Page Up function key
GLUT_KEY_PAGE_DOWN      Page Down function key
GLUT_KEY_HOME           Home function key
GLUT_KEY_END            End function key
GLUT_KEY_INSERT         Insert function key
```

In order for the code defined above on *processSpecialKeys* to compile we must add the declaration of the *red*, *green*, and *blue* variables to the beginning of our code. Furthermore, for the code to have the desired effect we must change the function responsible for the rendering, *renderScene*.

```
void renderScene(void) {

	// Clear Color and Depth Buffers
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	// Reset transformations

	glLoadIdentity();
	// Set the camera
	gluLookAt(	0.0f, 0.0f, 10.0f,
			0.0f, 0.0f,  0.0f,
			0.0f, 1.0f,  0.0f);

	glRotatef(angle, 0.0f, 1.0f, 0.0f);

	// the function responsible for setting the color
	glColor3f(red,green,blue);
	glBegin(GL_TRIANGLES);
		glVertex3f(-2.0f,-2.0f, 0.0f);
```

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,186 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models with Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

Google Ads

```
                glVertex3f( 2.0f, 0.0f, 0.0);
                glVertex3f( 0.0f, 2.0f, 0.0);
        glEnd();
        angle+=0.1f;

        glutSwapBuffers();
}
```

OK, now we're ready to tell GLUT that the functions we just defined are the ones that will process keyboard events. In other words it is time to call GLUT's *glutKeyboardFunc* and *glutSpecialFunc*.

The call to these functions can be made anywhere, meaning that we may change the processing function for keyboard event processing at any time. However this is not an usual feature, so we'll place it on the main function. Next we present the new main function, with keyboard processing is presented (note that this function is based on the previous sections of this tutorial):

```
int main(int argc, char **argv) {

        // init GLUT and create window
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(320,320);
        glutCreateWindow("Lighthouse3D- GLUT Tutorial");

        // register callbacks
        glutDisplayFunc(renderScene);
        glutReshapeFunc(changeSize);
        glutIdleFunc(renderScene);

        // here are the new entries
        glutKeyboardFunc(processNormalKeys);
        glutSpecialFunc(processSpecialKeys);

        // enter GLUT event processing cycle
        glutMainLoop();

        return 1;
}
```

**CTRL, ALT and SHIFT**

Sometimes we may want to know if one modifier key, i.e. CTRL, ALT or SHIFT is being pressed. GLUT provides a function that detects if any modifier is being pressed. This function should only be called inside the functions that process keyboard or mouse input events. The syntax for this function is:

```
int glutGetModifiers(void);
```

The return value for this function is either one of three predefined constants (in *glut.h*), or any bitwise OR combination of them. The constants are:

- GLUT_ACTIVE_SHIFT – Set if either you press the SHIFT key, or Caps Lock is on. Note that if they are both on then the constant is not set.
- GLUT_ACTIVE_CTRL – Set if you press the CTRL key.
- GLUT_ACTIVE_ALT – Set if you press the ALT key.

So lets extent our *processNormalKeys* a little bit to see how to handle these modifier keys. Suppose that you want the variable *red* to be 0.0 when the user presses r, and 1.0 when the user presses ALT + r. The following piece of code will do the trick:

```
void processNormalKeys(unsigned char key, int x, int y) {

        if (key == 27)
                exit(0);
        else if (key=='r') {
                int mod = glutGetModifiers();
                if (mod == GLUT_ACTIVE_ALT)
                        red = 0.0;
                else
                        red = 1.0;
        }
}
```

One last thing, how do you detect CTRL+ALT+F1? In this case we must detect two modifiers at the same time. In order to achieve this we do a bitwise OR with the desired constants. The following piece of code only changes the color to red if combination CTRL+ALT+F1 is pressed.

```
void processSpecialKeys(int key, int x, int y) {

	int mod;
	switch(key) {
		case GLUT_KEY_F1 :
		   mod = glutGetModifiers();
		   if (mod == (GLUT_ACTIVE_CTRL|GLUT_ACTIVE_ALT)) {
		        red = 1.0; green = 0.0; blue = 0.0;
		   }
		   break;
		case GLUT_KEY_F2 :
		   red = 0.0;
		   green = 1.0;
		   blue = 0.0; break;
		case GLUT_KEY_F3 :
		   red = 0.0;
		   green = 0.0;
		   blue = 1.0; break;
	}
}
```

**The code so far (without the modifiers)**

```
#include <stdlib.h>

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

// all variables initialized to 1.0, meaning
// the triangle will initially be white
float red=1.0f, blue=1.0f, green=1.0f;

// angle for rotating triangle
float angle = 0.0f;

void changeSize(int w, int h) {

	// Prevent a divide by zero, when window is too short
	// (you cant make a window of zero width).
	if (h == 0)
		h = 1;
	float ratio =  w * 1.0 / h;

	// Use the Projection Matrix
	glMatrixMode(GL_PROJECTION);

	// Reset Matrix
	glLoadIdentity();

	// Set the viewport to be the entire window
	glViewport(0, 0, w, h);

	// Set the correct perspective.
	gluPerspective(45.0f, ratio, 0.1f, 100.0f);

	// Get Back to the Modelview
	glMatrixMode(GL_MODELVIEW);
}

void renderScene(void) {

	// Clear Color and Depth Buffers
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	// Reset transformations
	glLoadIdentity();

	// Set the camera
```

```
        gluLookAt(      0.0f, 0.0f, 10.0f,
                        0.0f, 0.0f,  0.0f,
                        0.0f, 1.0f,  0.0f);

        glRotatef(angle, 0.0f, 1.0f, 0.0f);

        glColor3f(red,green,blue);
        glBegin(GL_TRIANGLES);
                glVertex3f(-2.0f,-2.0f, 0.0f);
                glVertex3f( 2.0f, 0.0f, 0.0);
                glVertex3f( 0.0f, 2.0f, 0.0);
        glEnd();

        angle+=0.1f;

        glutSwapBuffers();
}

void processNormalKeys(unsigned char key, int x, int y) {

        if (key == 27)
                exit(0);
}

void processSpecialKeys(int key, int x, int y) {

        switch(key) {
                case GLUT_KEY_F1 :
                                red = 1.0;
                                green = 0.0;
                                blue = 0.0; break;
                case GLUT_KEY_F2 :
                                red = 0.0;
                                green = 1.0;
                                blue = 0.0; break;
                case GLUT_KEY_F3 :
                                red = 0.0;
                                green = 0.0;
                                blue = 1.0; break;
        }
}

int main(int argc, char **argv) {

        // init GLUT and create window
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(320,320);
        glutCreateWindow("Lighthouse3D- GLUT Tutorial");

        // register callbacks
        glutDisplayFunc(renderScene);
        glutReshapeFunc(changeSize);
        glutIdleFunc(renderScene);

        // here are the new entries
        glutKeyboardFunc(processNormalKeys);
        glutSpecialFunc(processSpecialKeys);

        // enter GLUT event processing cycle
        glutMainLoop();

        return 1;
}
```

Copy the code above to your project, or go to the download section, and try it!

Prev: Animation

Next: Moving the Camera I

**7 Responses to “Keyboard”**

1. **Tim** says:
26/02/2012 at 5:53 AM

I found the "Ctrl + Alt + F1" works correctly in my program, but "Ctrl + Alt + R" did not! My code is just as follow:
void processNormalKeys(unsigned char key, int x, int y)
{
int mod = 0.0;
if (key == 'r' || key == 'R') {
mod = glutGetModifiers();
if (mod == (GLUT_ACTIVE_CTRL | GLUT_ACTIVE_ALT)) {
//if (mod == (GLUT_ACTIVE_ALT)) { //"Alt + r" works
triangleColor[0] = 1.0f;
triangleColor[1] = 0.0f;
triangleColor[2] = 0.0f;
}
}
}
It seems that the program can not detect the "Ctrl" when using "processNormalKeys()", but ok in "processSpecialKeys()".
But how if I just want to use the "Ctrl + Alt + r" to trigger my program, how to code?

Reply

2. **Qwerty** says:
11/12/2011 at 9:48 PM

The solution is probably here http://ernesthuntley.wordpress.com/2010/08/20/smooth-keyboard-input-with-glut/

Reply

3. **Qwerty** says:
11/12/2011 at 9:04 PM

Hello, my objects don't move smoothly when I keep pressed movement key. Is there a workaround for it? It behaves like in Word when you hold single key.

Reply

**Diogo Guerra** says:
15/01/2012 at 2:35 AM

Possibly what ur doing is reading the keys at full speed (speed that the sistem scans for new entrys), so what u need is a timer to execute that function from x to x seconds…

try to use sleep(int ) or msleep(int ) -> ubunto…. don't know if it works on other's…. include on windows

Or u can search msdn for time lybraries

Reply

4. **Saya** says:
02/11/2011 at 6:21 PM

Hi, I am trying to create a game in which the user crouches or crawls when they hold down the SHIFT or CTRL key, respectively. I want them to be able to do this without having to press any other buttons at the same time. However, you only show how to use these helper keys when another key is being pressed. Is there a way to call these keys without relying on other keys being pressed? Or is that not possible in GLUT?

Reply

5. **Dan** says:
26/05/2011 at 5:49 PM

I tried to use the CTRL+ALT_F1 function, but it didn't seem to work. Deleting GLUT_ACTIVE_ALT made things work with CTRL+F1.

Could you first explain why using an OR operator takes into account that you are pressing both ALT and CTRL?

Reply

**ARF** says:
26/05/2011 at 6:38 PM

Hi, I've tested it again, and the CTRL ALT F1 works correctly in my pc. Have you tried the ALT-F1 combination? Try printing out the values of the modifier with and without pressing the ALT key and see if you can get any hints from that.

# Lighthouse3d.com

MOSTLY ABOUT 3D GRAPHICS



## Keyboard Example: Moving the Camera

Prev: Keyboard Next: Advanced Keyboard

OK, so lets see a more exciting use for the keyboard using GLUT. In this section we're going to go through the code of an application that will draw a small world populated with snowmen, and we're going to use the direction keys to move the camera in this world. The left and right keys will rotate the camera around the Y axis, i.e. in the XZ plane, whereas the up and down keys will move the camera forward and backwards in the current direction.

The code for this sample application is now presented with comments where appropriate. First we need some global variables to store the camera parameters. The variables will store both the camera position and the vector that gives us the aiming direction. We will also store the angle. There is no need to store the *y* component since it is constant.

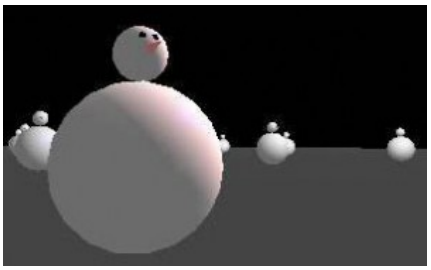
Hence, we need:

- angle: the angle of rotation in the y axis. this variable will allow us to rotate the camera
- x,z: The camera position in the XZ plane
- lx,lz: A vector defining our line of sight

Lets deal with the variable declarations:

```
// angle of rotation for the camera direction
float angle=0.0;
// actual vector representing the camera's direction
float lx=0.0f,lz=-1.0f;
// XZ position of the camera
float x=0.0f,z=5.0f;
```

The code to draw a snowman is now presented. The result looks like this



```
void drawSnowMan() {

        glColor3f(1.0f, 1.0f, 1.0f);

// Draw Body
        glTranslatef(0.0f ,0.75f, 0.0f);
        glutSolidSphere(0.75f,20,20);

// Draw Head
        glTranslatef(0.0f, 1.0f, 0.0f);
        glutSolidSphere(0.25f,20,20);

// Draw Eyes
        glPushMatrix();
        glColor3f(0.0f,0.0f,0.0f);
        glTranslatef(0.05f, 0.10f, 0.18f);
        glutSolidSphere(0.05f,10,10);
        glTranslatef(-0.1f, 0.0f, 0.0f);
```

```
        glutSolidSphere(0.05f,10,10);
        glPopMatrix();

// Draw Nose
        glColor3f(1.0f, 0.5f , 0.5f);
        glRotatef(0.0f,1.0f, 0.0f, 0.0f);
        glutSolidCone(0.08f,0.5f,10,2);
}
```

Next we have the new render function. It contains all the commands to draw our little world. Another change is in the *gluLookAt* function. The parameters of the *gluLookAt* function are now variables instead of fixed values. Just in case you aren't familiar with this function, here goes a brief explanation. The *gluLookAt* function provides an easy and intuitive way to set the camera position and orientation. Basically it has three groups of parameters, each one is composed of 3 floating point values. The first three values indicate the camera position. The second set of values defines the point we're looking at. Actually it can be any point in our line of sight.The last group indicates the up vector, this is usually set to (0.0, 1.0, 0.0), meaning that the camera's is not tilted. If you want to tilt the camera just play with these values. For example, to see everything upside down try (0.0, -1.0, 0.0).

As mentioned before x, y, and z represent the camera position so these values correspond to the first vector in *gluLookAt*. The second set of parameters, the look at point, is computed by adding the vector which defines our line of sight to the camera position.

- Look At Point = Line Of Sight + Camera Position

Here is the code for the render function:

```
void renderScene(void) {

        // Clear Color and Depth Buffers

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Reset transformations
        glLoadIdentity();
        // Set the camera
        gluLookAt(      x, 1.0f, z,
                        x+lx, 1.0f,  z+lz,
                        0.0f, 1.0f,  0.0f);

        // Draw ground
        glColor3f(0.9f, 0.9f, 0.9f);
        glBegin(GL_QUADS);
                glVertex3f(-100.0f, 0.0f, -100.0f);
                glVertex3f(-100.0f, 0.0f,  100.0f);
                glVertex3f( 100.0f, 0.0f,  100.0f);
                glVertex3f( 100.0f, 0.0f, -100.0f);
        glEnd();

        // Draw 36 SnowMen
        for(int i = -3; i < 3; i++)
                for(int j=-3; j < 3; j++) {
                        glPushMatrix();
                        glTranslatef(i*10.0,0,j * 10.0);
                        drawSnowMan();
                        glPopMatrix();
                }

        glutSwapBuffers();
}
```

Now where ready for the function that will process the arrow keys events. We're using the left and right arrow keys to rotate the camera, i.e. to change the vector that defines the line of sight. The up and down keys are used to move along the current line of sight.

When the user presses the left or right keys the variable *angle* is changed accordingly. Based on the angle value it computes the appropriate values for the new *lx* and *lz* components of the line of sight vector. Note that we're only moving in the XZ plane, therefore we don't need to change the *ly* coordinate of the line of sight vector. The new *lx* and *lz* are mapped onto a unitary circle on the XZ plane. Therefore, given a angle *ang*, the new values for *lx* and *lz* are:

- lx = sin(ang)
- lz = -cos(ang)

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,203 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models w ith Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

Google Ads

Just like if we wanted to convert from Polar coordinates to Euclidean coordinates. The lz is negative because the initial value is -1.

Note that the camera doesn't move when updating *lx* and *lz*, the camera position remains the same, only the look at point is altered.

We also want to move the camera along the line of sight, i.e. the next camera position must be along the line of sight vector. In order to achieve this we're going to add or subtract a fraction of the line of sight vector to our current position when the up or down arrows are pressed, respectively. For instance, to move the camera forward the new values for *x* and *z* are:

- x = x + lx * fraction
- z = z + lz * fraction

The fraction is a possible speed implementation. We know that (lx,lz) is a unitary vector (as mentioned before, its a point in the unit circle), therefore if the fraction is kept constant then the speed will be kept constant as well. By increasing the fraction we're moving faster, i.e. we're moving farther in each frame.

```
void processSpecialKeys(int key, int xx, int yy) {

	float fraction = 0.1f;

	switch (key) {
		case GLUT_KEY_LEFT :
			angle -= 0.01f;
			lx = sin(angle);
			lz = -cos(angle);
			break;
		case GLUT_KEY_RIGHT :
			angle += 0.01f;
			lx = sin(angle);
			lz = -cos(angle);
			break;
		case GLUT_KEY_UP :
			x += lx * fraction;
			z += lz * fraction;
			break;
		case GLUT_KEY_DOWN :
			x -= lx * fraction;
			z -= lz * fraction;
			break;
	}
}
```

Next follows the code for the main function using GLUT. The only news in here is the depth test enabling.

```
int main(int argc, char **argv) {

	// init GLUT and create window

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D - GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);
	glutReshapeFunc(changeSize);
	glutIdleFunc(renderScene);
	glutKeyboardFunc(processNormalKeys);
	glutSpecialFunc(processSpecialKeys);

	// OpenGL init
	glEnable(GL_DEPTH_TEST);

	// enter GLUT event processing cycle
	glutMainLoop();

	return 1;
}
```

Prev: Keyboard Next: Advanced Keyboard

**28 Responses to "Keyboard Example: Moving the Camera"**

1. **jctank** says:
26/06/2012 at 7:11 AM

Great tutorial… I grasp the material very quickly..Thanks a lot

Reply

2. **pramod** says:
04/05/2012 at 8:03 AM

```
#include
#include
#include
GLfloat bi;
GLint s,cr;
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(0.0,1.0,0.0);
glPointSize(6.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,900.0,0.0,640.0);
glMatrixMode(GL_MODELVIEW);
}
void box()
{
{
glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(50+cr+bi,270);
glVertex2i(50+cr+bi,250);
glVertex2i(70+cr+bi,250);
glVertex2i(70+cr+bi,270);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(150+cr+bi,360);
glVertex2i(150+cr+bi,340);
glVertex2i(170+cr+bi,340);
glVertex2i(170+cr+bi,360);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(270+cr+bi,270);
glVertex2i(270+cr+bi,250);
glVertex2i(290+cr+bi,250);
glVertex2i(290+cr+bi,270);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(300+cr+bi,270);
glVertex2i(300+cr+bi,250);
glVertex2i(320+cr+bi,250);
glVertex2i(320+cr+bi,270);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(400+cr+bi,270);
glVertex2i(400+cr+bi,250);
glVertex2i(420+cr+bi,250);
glVertex2i(420+cr+bi,270);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(440+cr+bi,270);
glVertex2i(440+cr+bi,250);
glVertex2i(460+cr+bi,250);
glVertex2i(460+cr+bi,270);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(550+cr+bi,350);
glVertex2i(550+cr+bi,330);
glVertex2i(570+cr+bi,330);
```

```
glVertex2i(570+cr+bi,350);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(600+cr+bi,350);
glVertex2i(600+cr+bi,330);
glVertex2i(620+cr+bi,330);
glVertex2i(620+cr+bi,350);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(700+cr+bi,270);
glVertex2i(700+cr+bi,250);
glVertex2i(720+cr+bi,250);
glVertex2i(720+cr+bi,270);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(830+cr+bi,270);
glVertex2i(830+cr+bi,250);
glVertex2i(850+cr+bi,250);
glVertex2i(850+cr+bi,270);
glEnd();
//box1();

glEnd();
glutPostRedisplay();
}

}

void redraw ()
{ if(s==0)
{
if(bi<40)
{

bi+=-0.1;
glutPostRedisplay();
}
else
{ bi=bi-1000;
glutPostRedisplay();
}
}
}

GLint i;
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POLYGON);
glColor3f(0.0,0.0,0.0);
glVertex2i(1,250);
glVertex2i(1,230);
glVertex2i(900,230);
glVertex2i(900,250);
glEnd();

redraw ();
box();

glutSwapBuffers();

glFlush();
glutPostRedisplay();
}
//GLint cr=0;

void main(int argc,char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
glutInitWindowSize(900,500);
glutCreateWindow ("hgfh");

glutDisplayFunc(display);

myinit();

glutMainLoop();
}
```

this a bit of prog….and i need to design a game where a man is standing at one end that is at the starting of the screen and on the background the blocks will be moving as u can see here i want to give it a keyboard use…plz help me……to do so by pressing "a" in the keyboard the man should go up…like this…plz help me….

Reply

3. **JW** says:
10/04/2012 at 3:54 PM

In case somebody wants a version that's easy to copy/paste and implement:

Seems like it will be useful since there was actually quite a lot changed since the last version.

```
#include
#include

#ifdef __APPLE__
#include
#else
#include
#endif

// actual vector representing the camera's direction
float lx=0.0f,lz=-1.0f;
// XZ position of the camera
float x=0.0f,z=5.0f;

// all variables initialized to 1.0, meaning
// the triangle will initially be white
float red=1.0f, blue=1.0f, green=1.0f;

// angle for rotating triangle
float angle = 0.0f;

//int dummy = 5;

void drawSnowMan()
{

glColor3f(1.0f, 1.0f, 1.0f);

// Draw Body
glTranslatef(0.0f ,0.75f, 0.0f);
glutSolidSphere(0.75f,20,20);

// Draw Head
glTranslatef(0.0f, 1.0f, 0.0f);
glutSolidSphere(0.25f,20,20);

// Draw Eyes
glPushMatrix();
glColor3f(0.0f,0.0f,0.0f);
glTranslatef(0.05f, 0.10f, 0.18f);
glutSolidSphere(0.05f,10,10);
glTranslatef(-0.1f, 0.0f, 0.0f);
glutSolidSphere(0.05f,10,10);
glPopMatrix();

// Draw Nose
glColor3f(1.0f, 0.5f , 0.5f);
glRotatef(0.0f,1.0f, 0.0f, 0.0f);
glutSolidCone(0.08f,0.5f,10,2);
}

void changeSize(int w, int h)
{

// Prevent a divide by zero, when window is too short
// (you cant make a window of zero width).
if (h == 0)
h = 1;
float ratio = w * 1.0 / h;

// Use the Projection Matrix
glMatrixMode(GL_PROJECTION);

// Reset Matrix
glLoadIdentity();

// Set the viewport to be the entire window
glViewport(0, 0, w, h);

// Set the correct perspective.
gluPerspective(45.0f, ratio, 0.1f, 100.0f);

// Get Back to the Modelview
glMatrixMode(GL_MODELVIEW);
}

void renderScene(void)
{

// Clear Color and Depth Buffers

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
// Reset transformations
glLoadIdentity();
// Set the camera
gluLookAt( x, 1.0f, z,
x+lx, 1.0f, z+lz,
0.0f, 1.0f, 0.0f);

// Draw ground
glColor3f(0.9f, 0.9f, 0.9f);
glBegin(GL_QUADS);
glVertex3f(-100.0f, 0.0f, -100.0f);
glVertex3f(-100.0f, 0.0f, 100.0f);
glVertex3f( 100.0f, 0.0f, 100.0f);
glVertex3f( 100.0f, 0.0f, -100.0f);
glEnd();

// Draw 36 SnowMen
for(int i = -3; i < 3; i++)
for(int j=-3; j < 3; j++) {
glPushMatrix();
glTranslatef(i*10.0,0,j * 10.0);
drawSnowMan();
glPopMatrix();
}

glutSwapBuffers();
}

void processNormalKeys(unsigned char key, int x, int y)
{

if (key == 27)
exit(0);
}

void processSpecialKeys(int key, int xx, int yy)
{

float fraction = 0.1f;

switch (key) {
case GLUT_KEY_LEFT :
angle -= 0.01f;
lx = sin(angle);
lz = -cos(angle);
break;
case GLUT_KEY_RIGHT :
angle += 0.01f;
lx = sin(angle);
lz = -cos(angle);
break;
case GLUT_KEY_UP :
x += lx * fraction;
z += lz * fraction;
break;
case GLUT_KEY_DOWN :
x -= lx * fraction;
z -= lz * fraction;
break;
}
}

int main(int argc, char **argv)
{

// init GLUT and create window

glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
glutInitWindowPosition(100,100);
glutInitWindowSize(320,320);
glutCreateWindow("Lighthouse3D - GLUT Tutorial");

// register callbacks
glutDisplayFunc(renderScene);
glutReshapeFunc(changeSize);
glutIdleFunc(renderScene);
glutKeyboardFunc(processNormalKeys);
glutSpecialFunc(processSpecialKeys);

// OpenGL init
glEnable(GL_DEPTH_TEST);

// enter GLUT event processing cycle
glutMainLoop();

return 1;
}
```

Reply

Vipin says:

10/09/2012 at 6:32 PM

Thanks a lot dude …

Reply

4. **sharma** says:
05/04/2012 at 8:20 AM

```
#include “stdafx.h”
#include
#include
// angle of rotation for the camera direction
float angle=0.0;
// actual vector representing the camera’s direction
float lx=0.0f,lz=-1.0f;
// XZ position of the camera
float x=0.0f,z=5.0f;

void drawTriangle()
{
glBegin(GL_TRIANGLES);
glColor3f(1.0,0,0);
glVertex3f(-0.5f,-0.5f, 0.0f);
glVertex3f( -0.5f, 0.0f, 0.0);
glVertex3f( 0.0f, -0.5f, 0.0);
glEnd();
}

void processSpecialKeys(int key, int xx, int yy) {

float fraction = 0.01f;
switch (key) {
case GLUT_KEY_LEFT :
angle -= 0.001f;
lx = sin(angle);
lz = -cos(angle);
break;

case GLUT_KEY_RIGHT :
angle += 0.001f;
lx = sin(angle);
lz = -cos(angle);
break;
case GLUT_KEY_UP :
lx += lx * fraction;
lz += lz * fraction;
break;
case GLUT_KEY_DOWN :
lx -= lx * fraction;
lz -= lz * fraction;
break;
}
}

void renderScene(void) {

gluLookAt( x, 1.0f, z,
x+lx, 1.0f, z+lz,
0.0f, 1.0f, 0.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();

glColor3f(0.2f, 0.1f, 0.4f);
glBegin(GL_QUADS);
glVertex3f(-100.0f, 0.0f, -100.0f);
glVertex3f(-100.0f, 0.0f, 100.0f);
glVertex3f( 100.0f, 0.0f, 100.0f);
glVertex3f( 100.0f, 0.0f, -100.0f);
glEnd();
for(int i = -3; i < 3; i++)
for(int j=-3; j < 3; j++) {
glPushMatrix();
glTranslatef(i*10,0,j * 10);
drawTriangle();
glPopMatrix();
}
glutSwapBuffers();
}
void reshape (int width, int height) {
glViewport(0, 0, (GLsizei)width, (GLsizei)height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(0, (GLfloat)width / (GLfloat)height, 1.0, 100.0);
glMatrixMode(GL_MODELVIEW);
}
```

```
int main(int argc, char **argv) {
#pragma comment( linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"" );
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA |GLUT_DEPTH);
glutInitWindowPosition(50,50);
glutInitWindowSize(500,500);
glutCreateWindow(argv[0]);
glutDisplayFunc(renderScene);
glutReshapeFunc(reshape);
glutIdleFunc(renderScene);
glutSpecialFunc(processSpecialKeys);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
return 1;
}
```

this code is not working sir plz help me .

Reply

5. %$&#YEAH says:
20/02/2012 at 8:32 AM

%$&# yeah… This is best tutorial I have ever got… THANKS MAAAAN!!!!

Reply

6. John Mieske says:
11/02/2012 at 6:05 PM

And now I see this site doesn't show the includes properly because of the greater then and less then brackets having a conflict. That I see is part of the problem here on this website. Hard for anyone else to learn to know what libraries are needed. Good thing I knew what you needed. For your code to work, make sure the following is ADDED to the top, but change what I spelled out with the actual symbol. Greater then and less then signs.

#ifdef __APPLE__
#include ( LESS THEN BRACKET ) GLUT/glut.h ( GREATER THEN BRACKET )
#else
#include ( LESS THEN BRACKET ) GL/glut.h ( GREATER THEN BRACKET )
#endif

#include ( LESS THEN BRACKET ) stdlib.h ( GREATER THEN BRACKET )
#include "math.h"

Reply

7. John Mieske says:
11/02/2012 at 5:55 PM

@ Zoe, Here you go, your code has been fixed. Turns out, you had more then one problem with this code. First off, your screen position was set way out to 8000+ for both X and Y. So I fixed that with 1024×768 as a screen resolution and set the window position to 1,1. The 1,1 is where on your screen it will show up, which in this case will be the top left.

The other problem is, you didn't supply the Math.H file or any includes. You just show # Include, and that's all you had. So that's been fixed.

Enjoy,
John

```
#ifdef __APPLE__
#include
#else
#include
#endif

#include
#include "math.h"

GLuint listID;
GLuint list2ID;
GLUquadric *sun, *mercury, *venus, *earth, *mars, *jupiter, *saturn, *uranus,
*neptune;
GLfloat zoomx=128000;
GLfloat zoomy=80000;
// angle of rotation for the camera direction
GLfloat angle=0.0;
// actual vector representing the camera's direction
GLfloat lx=0.0f,lz=-1.0f;
// XZ position of the camera
GLfloat x=0.0f,z=5.0f;
GLfloat fraction;

void init()
{
```

```
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
//store view matrix

glMatrixMode (GL_MODELVIEW);
glLoadIdentity();

angle=0;
glutInitWindowPosition(1,1);
glutInitWindowSize(1024,768);
glutCreateWindow("Solar System");

glClearColor(0,0,0,0);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-128000,128000,-80000,80000,1,5000);

glClearColor(0,0,0,1);

//glEnable(GL_DEPTH_TEST);

sun=gluNewQuadric();
mercury=gluNewQuadric();
venus=gluNewQuadric();
earth=gluNewQuadric();
mars=gluNewQuadric();
jupiter=gluNewQuadric();
saturn=gluNewQuadric();
uranus=gluNewQuadric();
neptune=gluNewQuadric();
}

void display()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

//glLoadIdentity();
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);

//draw sun
glLoadIdentity();
glColor3f(1,0,0);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
//glutPostRedisplay();
gluQuadricDrawStyle(sun, GLU_LINE);
gluSphere(sun,1400,400,400);

//draw mercury
angle=(angle+0.01);
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-397+1930*cos(angle/0.24)*cos(0.51)-1887*sin(angle/0.24)*sin(0.51),19
//glutPostRedisplay();
gluQuadricDrawStyle(mercury, GLU_LINE);
gluSphere(mercury,25,20,20);

//draw venus
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-24.4+3607*cos(angle/0.62)*cos(0.5)-3606*sin(angle/0.62)*sin(0.5),360
//glutPostRedisplay();
gluQuadricDrawStyle(venus, GLU_LINE);
gluSphere(venus,61,20,20);

//draw earth
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-83+4986*cos(angle)*cos(2)-4985*sin(angle)*sin(2),4986*cos(angle)*sin
//glutPostRedisplay();
gluQuadricDrawStyle(earth, GLU_LINE);
gluSphere(earth,64,10,10);

//draw mars
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-709+7598*cos(angle/1.9)*cos(5)-7560*sin(angle/1.9)*sin(5),7598*cos(a
//glutPostRedisplay();
gluQuadricDrawStyle(mars, GLU_LINE);
gluSphere(mars,34,10,10);

//draw jupiter
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-1266+25951*cos(angle/11.9)*cos(4.8)-25919*sin(angle/11.9)*sin(4.8),2
//glutPostRedisplay();
```

```
gluQuadricDrawStyle(jupiter, GLU_LINE);
gluSphere(jupiter,714,100,100);

//draw saturn
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-2664+47783*cos(angle/29.5)*cos(15)-47697*sin(angle/29.5)*sin(15),477
//glutPostRedisplay();
gluQuadricDrawStyle(saturn, GLU_LINE);
gluSphere(saturn,602,100,100);

//draw uranus
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-4258+95889*cos(angle/84.32)*cos(6)-95769*sin(angle/84.32)*sin(6),958
//glutPostRedisplay();
gluQuadricDrawStyle(uranus, GLU_LINE);
gluSphere(uranus,255,100,100);

//draw neptune
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-1683.4+150114*cos(angle/164.8)*cos(4.6)-150107*sin(angle/164.8)*sin(
glutPostRedisplay();
gluQuadricDrawStyle(neptune, GLU_LINE);
gluSphere(neptune,247,100,100);
//glLoadIdentity();
glutSwapBuffers();

}

void keysnormal(unsigned char nkey, int x2, int y2)
{
fraction = 0.1f;
switch (nkey) {
case 27 : //Esc key
exit(0);
case 65 : //A key
angle -= 0.01f;
lx = sin(angle);
lz = -cos(angle);
//glLoadIdentity();
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
//glutPostRedisplay();
break;
case 68 : //D key
angle += 0.01f;
lx = sin(angle);
lz = -cos(angle);
//glLoadIdentity();
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
//glutPostRedisplay();
break;
case 87 : //W key
x += lx * fraction;
z += lz * fraction;
//glLoadIdentity();
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
//glutPostRedisplay();
break;
case 83 : //S key
x -= lx * fraction;
z -= lz * fraction;
//glLoadIdentity();
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
//glutPostRedisplay();
break;
}

}

void keyspecial( int key, int x1, int y1 )
{
if (key==GLUT_KEY_UP)
{
zoomx=zoomx/2;
zoomy=zoomy/2;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1*zoomx,zoomx,-1*zoomy,zoomy,1,5000);
glutPostRedisplay();
}
if (key==GLUT_KEY_DOWN)
{
zoomx=zoomx*2;
zoomy=zoomy*2;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1*zoomx,zoomx,-1*zoomy,zoomy,1,5000);
```

```
glutPostRedisplay();
}
}

int main(int argc, char **argv)
{
glutInit(&argc,argv);

//Initializations
init();

glutDisplayFunc(display);
glutKeyboardFunc(keysnormal);
glutSpecialFunc(keyspecial);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
return 0;
}
```

Reply

8. zoe says:
27/07/2011 at 1:02 PM

If anyone feel like debugging please help! I am trying to make camera moves like this tutorial in a simple solar system model but it doesn't work. I've tried to put glulookat at different positions in my code but still nothing.
Here is my code (sorry for the wall of text, I couldn't find any way to hide it):

```
#include // Header file for standard file i/o.
#include // Header file for malloc/free.
#include
#include //include the gl header file
#include //include the glut header file /* this includes the necessary X
headers */

GLuint listID;
GLuint list2ID;
GLUquadric *sun, *mercury, *venus, *earth, *mars, *jupiter, *saturn, *uranus,
*neptune;
GLfloat zoomx=128000;
GLfloat zoomy=80000;
// angle of rotation for the camera direction
GLfloat angle=0.0;
// actual vector representing the camera's direction
GLfloat lx=0.0f,lz=-1.0f;
// XZ position of the camera
GLfloat x=0.0f,z=5.0f;
GLfloat fraction;

void init()
{ glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
angle=0;
glutInitWindowPosition(128000,80000);
glutInitWindowSize(1280,800);
glutCreateWindow("Solar System");

glClearColor(0,0,0,0);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-128000,128000,-80000,80000,1,5000);

glClearColor(0,0,0,1);

//glEnable(GL_DEPTH_TEST);

sun=gluNewQuadric();
mercury=gluNewQuadric();
venus=gluNewQuadric();
earth=gluNewQuadric();
mars=gluNewQuadric();
jupiter=gluNewQuadric();
saturn=gluNewQuadric();
uranus=gluNewQuadric();
neptune=gluNewQuadric();
}

void display()
{ glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

//glLoadIdentity();
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);

//draw sun
glLoadIdentity();
glColor3f(1,0,0);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
```

```
//glutPostRedisplay();
gluQuadricDrawStyle(sun, GLU_LINE);
gluSphere(sun,1400,400,400);

//draw mercury
angle=(angle+0.01);
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-397+1930*cos(angle/0.24)*cos(0.51)-1887*sin(angle/0.24)*sin(0.51),19
//glutPostRedisplay();
gluQuadricDrawStyle(mercury, GLU_LINE);
gluSphere(mercury,25,20,20);

//draw venus
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-24.4+3607*cos(angle/0.62)*cos(0.5)-3606*sin(angle/0.62)*sin(0.5),360
//glutPostRedisplay();
gluQuadricDrawStyle(venus, GLU_LINE);
gluSphere(venus,61,20,20);

//draw earth
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-83+4986*cos(angle)*cos(2)-4985*sin(angle)*sin(2),4986*cos(angle)*sin
//glutPostRedisplay();
gluQuadricDrawStyle(earth, GLU_LINE);
gluSphere(earth,64,10,10);

//draw mars
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-709+7598*cos(angle/1.9)*cos(5)-7560*sin(angle/1.9)*sin(5),7598*cos(a
//glutPostRedisplay();
gluQuadricDrawStyle(mars, GLU_LINE);
gluSphere(mars,34,10,10);

//draw jupiter
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-1266+25951*cos(angle/11.9)*cos(4.8)-25919*sin(angle/11.9)*sin(4.8),2
//glutPostRedisplay();
gluQuadricDrawStyle(jupiter, GLU_LINE);
gluSphere(jupiter,714,100,100);

//draw saturn
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-2664+47783*cos(angle/29.5)*cos(15)-47697*sin(angle/29.5)*sin(15),477
//glutPostRedisplay();
gluQuadricDrawStyle(saturn, GLU_LINE);
gluSphere(saturn,602,100,100);

//draw uranus
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-4258+95889*cos(angle/84.32)*cos(6)-95769*sin(angle/84.32)*sin(6),958
//glutPostRedisplay();
gluQuadricDrawStyle(uranus, GLU_LINE);
gluSphere(uranus,255,100,100);

//draw neptune
glLoadIdentity();
glColor3f(0,0,1);
gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
glTranslatef(-1683.4+150114*cos(angle/164.8)*cos(4.6)-150107*sin(angle/164.8)*sin(
glutPostRedisplay();
gluQuadricDrawStyle(neptune, GLU_LINE);
gluSphere(neptune,247,100,100);

glutSwapBuffers();

}

void keysnormal(unsigned char nkey, int x2, int y2)
{
fraction = 0.1f;
switch (nkey) {
case 27 : //Esc key
exit(0);
case 65 : //A key
angle -= 0.01f;
lx = sin(angle);
lz = -cos(angle);
//glLoadIdentity();
```

```
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
//glutPostRedisplay();
break;
case 68 : //D key
angle += 0.01f;
lx = sin(angle);
lz = -cos(angle);
//glLoadIdentity();
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
//glutPostRedisplay();
break;
case 87 : //W key
x += lx * fraction;
z += lz * fraction;
//glLoadIdentity();
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
//glutPostRedisplay();
break;
case 83 : //S key
x -= lx * fraction;
z -= lz * fraction;
//glLoadIdentity();
//gluLookAt( x, 1.0f, z,x+lx, 1.0f, z+lz,0.0f, 1.0f, 0.0f);
//glutPostRedisplay();
break;
}

}

void keyspecial( int key, int x1, int y1 )
{

if (key==GLUT_KEY_UP)
{
zoomx=zoomx/2;
zoomy=zoomy/2;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1*zoomx,zoomx,-1*zoomy,zoomy,1,5000);
glutPostRedisplay();
}
if (key==GLUT_KEY_DOWN)
{
zoomx=zoomx*2;
zoomy=zoomy*2;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1*zoomx,zoomx,-1*zoomy,zoomy,1,5000);
glutPostRedisplay();

}
}

int main(int argc, char **argv)
{
glutInit(&argc,argv);

//Initializations
init();

glutDisplayFunc(display);
glutKeyboardFunc(keysnormal);
glutSpecialFunc(keyspecial);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
return 0;

}
```

Reply

9. **zoe** says:
27/07/2011 at 11:28 AM

Extremely helpful tutorials mate. Thanks a lot.

Reply

10. **Ozair** says:
25/06/2011 at 4:50 PM

hi, can you please explain about the calculations in processSpecialKeys function in more depth?

what if I want to change the starting direction of the camera? what should be the angle and the corresponding vector?

Reply

## Advanced Keyboard Features

Prev: Moving the Camera I Next: Moving the Camera II

In this section we're going to introduce four new functions to deal with the keyboard. These functions work hand in hand to release us from the auto repeat problem that causes a small delay when waiting for the autorepeat to be operated.

The first function we're going to present in here allows us to disable the keyboard repeat. The syntax is as follows:

```
int glutSetKeyRepeat(int repeatMode);
```

Parameters:

- repeatMode – Enables, disables or restores the auto repeat mode. See bellow for possible values.

Possible values for *repeatMode* are as follows:

- GLUT_KEY_REPEAT_OFF – disable auto repeat mode
- GLUT_KEY_REPEAT_ON – enable auto repeat mode
- GLUT_KEY_REPEAT_DEFAULT – reset the auto repeat mode to its default state.

Note that this function works in a global basis, i.e. it will affect the repeat mode in all windows, not just the ones from our applications. So beware, when using this function to disable the auto repeat mode, it is convinient to reset it to its default state before terminating the application.

GLUT provides us with a safer approach, disabling callbacks for keyboard when the key repeat occurs. This allows us to safely ignore key repeats in our application without affecting the other apps. The function that provides this functionality is presented next:

```
int glutIgnoreKeyRepeat(int repeatMode);
```
Parameters:

- repeatMode – zero enables auto-repeat, non-zero disables it.

In any case, we'll stop receiving callbacks when a key repeat occurs. However if you want to have an action performed while the key is being pressed, you'll need to know when the key is released. GLUT provides two functions that register callbacks when a key is released.

```
void glutKeyboardUpFunc(void (*func)(unsigned char key,int x,int y));

void glutSpecialUpFunc(void (*func)(int key,int x, int y));
```

Parameters:

- func – the name of the callback function.

The argument, a name of a function, will be the function that will handle these events. The parameters are the same as for when the user presses a key, so if a memory refresh is required look in the previous section.

In the next section we'll show how to use this features to improve the application by revisiting the last example.

Prev: Moving the Camera I Next: Moving the Camera II

**2 Responses to "Advanced Keyboard Features"**

1. point says:
21/04/2013 at 4:32 AM

thanks

Reply

2. suny kumar says:
09/11/2011 at 10:00 AM

just w ant to thank you.

Reply

**Leave a Reply**

Name (required)

E-mail (required)

URI

Your Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

Submit Comment

Notify me of follow-up comments by email

Notify me of new posts by email

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,203 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models w ith Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

**Links**

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

**Google Ads**

# Keyboard Example: Moving the Camera II

Prev: Advanced Keyboard Next: The Code So Far

In this section the last example is revisited. This time we'll use the advanced keyboard features.

The approach is to test when the key is pressed to start the motion, and only stop it when the key is released. Hence we are going to disable callbacks when the key repeat occurs with *glutIgnoreKeyRepeat*.

When a key is pressed we are going to set a variable to a non-zero value. Once the key is released the variable will be set to zero again.

Since no callbacks are active between key pressed and key released we have to check these variables in the render function and update the camera position and orientation accordingly.

In the initialization section we have two new variables: *deltaAngle* and *deltaMove*. These variables control the rotation and movement of the camera respectively. When non-zero some camera action will occur, when zero the camera is still. These two variables take an initial zero value, meaning that initially the camera is still.

In the beginning of our code we are going to add two variables to keep track of the key status, one for the orientation, *deltaAngle* and another for the position, *deltaMove*.

```
// the key states. These variables will be zero
//when no key is being presses
float deltaAngle = 0.0f;
float deltaMove = 0;
```

In the render function we will add some code at the beginning to check these vars and update the position and orientation accordingly.

```
void renderScene(void) {

	if (deltaMove)
		computePos(deltaMove);
	if (deltaAngle)
		computeDir(deltaAngle);
	...
```

Where *computePos* and *computeDir* are defined as:

```
void computePos(float deltaMove) {

	x += deltaMove * lx * 0.1f;
	z += deltaMove * lz * 0.1f;
}

void computeDir(float deltaAngle) {

	angle += deltaAngle;
	lx = sin(angle);
	lz = -cos(angle);
}
```

The functions that respond to the events of key pressed and key released are also going to suffer modifications. When a key is pressed we are only setting the respective variable to a non-zero value. When the key is released the variable will go back to zero.

```
void pressKey(int key, int xx, int yy) {

	switch (key) {
		case GLUT_KEY_LEFT : deltaAngle = -0.01f; break;
		case GLUT_KEY_RIGHT : deltaAngle = 0.01f; break;
		case GLUT_KEY_UP : deltaMove = 0.5f; break;
		case GLUT_KEY_DOWN : deltaMove = -0.5f; break;
	}
}

void releaseKey(int key, int x, int y) {

	switch (key) {
		case GLUT_KEY_LEFT :
		case GLUT_KEY_RIGHT : deltaAngle = 0.0f;break;
		case GLUT_KEY_UP :
		case GLUT_KEY_DOWN : deltaMove = 0;break;
	}
}
```

Finally, in the main function there are three new lines:

*glutIgnoreKeyRepeat* is called with a non-zero parameter to ask GLUT to stop reporting key repeats. Afterwards, *glutSpecialUpFunc* is called to register the callback function to process the key up event.

```
int main(int argc, char **argv) {

	// init GLUT and create window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D - GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);
	glutReshapeFunc(changeSize);
	glutIdleFunc(renderScene);
	glutSpecialFunc(pressKey);

	// here are the new entries
	glutIgnoreKeyRepeat(1);
	glutSpecialUpFunc(releaseKey);

	// OpenGL init
	glEnable(GL_DEPTH_TEST);

	// enter GLUT event processing cycle
	glutMainLoop();

	return 1;
}
```

Prev: Advanced Keyboard Next: The Code So Far

**4 Responses to "Keyboard Example: Moving the Camera II"**

1. Salem says:
07/11/2013 at 1:06 AM

modification to keep movement going when accidentally both opposite keys are pressed then one of them is released.

```
void pressKey(int key, int xx, int yy) {

switch (key) {
case GLUT_KEY_LEFT : deltaAngle += -0.01f; break;
case GLUT_KEY_RIGHT : deltaAngle += 0.01f; break;
case GLUT_KEY_UP : deltaMove += 0.5f; break;
case GLUT_KEY_DOWN : deltaMove += -0.5f; break;
}
```

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,203 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models with Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

**Links**

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

```
}

void releaseKey(int key, int x, int y) {

switch (key) {
case GLUT_KEY_LEFT : deltaAngle -= -0.01f; break;
case GLUT_KEY_RIGHT : deltaAngle -= 0.01f; break;
case GLUT_KEY_UP : deltaMove -= 0.5f; break;
case GLUT_KEY_DOWN : deltaMove -= -0.5f; break;
}
}
```

Reply

2. **shriram** says:
01/01/2013 at 9:28 AM

How is the direction Calculated using sine and cos fn ??

Reply

> **ARF** says:
> 02/01/2013 at 11:38 PM
>
> Hi Shiram,
>
> I'm using polar coordinates to compute the camera direction.
>
> Reply

3. **Sandeep** says:
12/02/2012 at 9:23 PM

This tutorial is awesome! When changing position and angle the way you've mentioned in this page, movement becomes a lot smoother!

Reply

### Leave a Reply

Name (required)

E-mail (required)

URI

Your Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

Submit Comment

Notify me of follow-up comments by email

Notify me of new posts by email

# Lighthouse3d.com

MOSTLY ABOUT 3D GRAPHICS



## The Code So Far

Prev: Moving the Camera II Next: The Mouse

Here is the complete code for the previous example:

```
#include <stdlib.h>
#include <math.h>

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

// angle of rotation for the camera direction
float angle = 0.0f;
// actual vector representing the camera's direction
float lx=0.0f,lz=-1.0f;
// XZ position of the camera
float x=0.0f, z=5.0f;
// the key states. These variables will be zero
//when no key is being presses
float deltaAngle = 0.0f;
float deltaMove = 0;

void changeSize(int w, int h) {

        // Prevent a divide by zero, when window is too short
        // (you cant make a window of zero width).
        if (h == 0)
                h = 1;
        float ratio =  w * 1.0 / h;

        // Use the Projection Matrix
        glMatrixMode(GL_PROJECTION);

        // Reset Matrix
        glLoadIdentity();

        // Set the viewport to be the entire window
        glViewport(0, 0, w, h);

        // Set the correct perspective.
        gluPerspective(45.0f, ratio, 0.1f, 100.0f);

        // Get Back to the Modelview
        glMatrixMode(GL_MODELVIEW);
}

void drawSnowMan() {

        glColor3f(1.0f, 1.0f, 1.0f);

// Draw Body

        glTranslatef(0.0f ,0.75f, 0.0f);
        glutSolidSphere(0.75f,20,20);

// Draw Head
        glTranslatef(0.0f, 1.0f, 0.0f);
        glutSolidSphere(0.25f,20,20);
```

```
// Draw Eyes
        glPushMatrix();
        glColor3f(0.0f,0.0f,0.0f);
        glTranslatef(0.05f, 0.10f, 0.18f);
        glutSolidSphere(0.05f,10,10);
        glTranslatef(-0.1f, 0.0f, 0.0f);
        glutSolidSphere(0.05f,10,10);
        glPopMatrix();

// Draw Nose
        glColor3f(1.0f, 0.5f , 0.5f);
        glRotatef(0.0f,1.0f, 0.0f, 0.0f);
        glutSolidCone(0.08f,0.5f,10,2);
}

void computePos(float deltaMove) {

        x += deltaMove * lx * 0.1f;
        z += deltaMove * lz * 0.1f;
}

void computeDir(float deltaAngle) {

        angle += deltaAngle;
        lx = sin(angle);
        lz = -cos(angle);
}

void renderScene(void) {

        if (deltaMove)
                computePos(deltaMove);
        if (deltaAngle)
                computeDir(deltaAngle);

        // Clear Color and Depth Buffers
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Reset transformations
        glLoadIdentity();
        // Set the camera
        gluLookAt(      x, 1.0f, z,
                                x+lx, 1.0f,  z+lz,
                                0.0f, 1.0f,  0.0f);

// Draw ground

        glColor3f(0.9f, 0.9f, 0.9f);
        glBegin(GL_QUADS);
                glVertex3f(-100.0f, 0.0f, -100.0f);
                glVertex3f(-100.0f, 0.0f,  100.0f);
                glVertex3f( 100.0f, 0.0f,  100.0f);
                glVertex3f( 100.0f, 0.0f, -100.0f);
        glEnd();

// Draw 36 SnowMen

        for(int i = -3; i < 3; i++)
                for(int j=-3; j < 3; j++) {
                        glPushMatrix();
                        glTranslatef(i*10.0,0,j * 10.0);
                        drawSnowMan();
                        glPopMatrix();
                }

        glutSwapBuffers();
}


void pressKey(int key, int xx, int yy) {

        switch (key) {
                case GLUT_KEY_LEFT : deltaAngle = -0.01f; break;
                case GLUT_KEY_RIGHT : deltaAngle = 0.01f; break;
                case GLUT_KEY_UP : deltaMove = 0.5f; break;
                case GLUT_KEY_DOWN : deltaMove = -0.5f; break;
        }
}
```

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,203 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models w ith Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

```
void releaseKey(int key, int x, int y) {

	switch (key) {
		case GLUT_KEY_LEFT :
		case GLUT_KEY_RIGHT : deltaAngle = 0.0f;break;
		case GLUT_KEY_UP :
		case GLUT_KEY_DOWN : deltaMove = 0;break;
	}
}

int main(int argc, char **argv) {

	// init GLUT and create window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D - GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);
	glutReshapeFunc(changeSize);
	glutIdleFunc(renderScene);

	glutSpecialFunc(pressKey);

	// here are the new entries
	glutIgnoreKeyRepeat(1);
	glutSpecialUpFunc(releaseKey);

	// OpenGL init
	glEnable(GL_DEPTH_TEST);

	// enter GLUT event processing cycle
	glutMainLoop();

	return 1;
}
```

The code can be downloaded from this section.

Prev: Moving the Camera II Next: The Mouse

**3 Responses to "The Code So Far"**

1. Thomas says:
11/12/2013 at 3:21 PM

For which reason if i remove the 2 for loop to draw snowmen in the renderscene function the camera became much faster?

Reply

2. Morten says:
14/09/2011 at 5:37 PM

You implemented processNormalKeys, but you did not call glutKeyboardFunc(). (Which only means that Escape can't be used to exit the program in this example.)

Reply

ARF says:
16/09/2011 at 11:35 PM

Quite right! The processNormalKeys function was from the previous example and it was kept although, as you mention, not useful at all. The function has been removed from the source code. Thanks.

Reply

# The Mouse

Prev: The Code So Far Next: Moving the Camera III

In the previous section we saw how to add interactivity to an OpenGL application using GLUT's Keyboard functionality. Now its time to explore the mouse. GLUT's mouse interface provides a lot of options for adding mouse interactivy, namely detecting clicks and mouse motion.

**Detecting Mouse Clicks**

As in the keyboard version, GLUT provides a way for you to register the function that will be responsible for processing events generated by mouse clicks. The name of this function is *glutMouseFunc*,and it is commonly called in the initialization phase of the application. The syntax is as follows:

```
void glutMouseFunc(void (*func)(int button, int state, int x, int y));
```

Parameters:

- func – The name of the function that will handle mouse click events

As we can see from the signature of *glutMouseFunc*, the function that will handle the mouse click events must have four parameters.

The first relates to which button was pressed, or released. This argument can have one of three values:

- GLUT_LEFT_BUTTON
- GLUT_MIDDLE_BUTTON
- GLUT_RIGHT_BUTTON

The second argument relates to the state of the button when the callback was generated, i.e. pressed or released. The possible values are:

- GLUT_DOWN
- GLUT_UP

When a callback is generated with the state GLUT_DOWN, the application can assume that a GLUT_UP will come afterwards even if the mouse moves outside the window.

The remaining two parameters provide the (*x*,*y*) coordinates of the mouse relatively to the upper left corner of the client area of the window.

**Detecting Motion**

GLUT provides mouse motion detection capabilities to an application. There are two types of motion that GLUT handles: active and passive motion. Active motion occurs when the mouse is moved and a button is pressed. Passive motion is when the mouse is moving but no buttons are pressed. If an application is tracking motion, an event will be generated per frame during the period that the mouse is moving.

As usual you must register with GLUT the function that will be responsible for handling the motion events. GLUT allows us to specify two different functions: one for tracking passive motion, and another to track active motion.

The signatures for the GLUT functions are as follows:

```
void glutMotionFunc(void (*func) (int x,int y));
void glutPassiveMotionFunc(void (*func) (int x, int y));
```

Parameters:

- func – the function that will be responsible for the respective type of motion.

---

The parameters for the motion processing function are the (*x*,*y*) coordinates of the mouse relatively to the upper left corner of the window's client area.

**Detecting when the mouse enters or leaves the window**

GLUT is also able to detect when the mouse leaves or enters the window area. A callback function can be registered to handle these two events. The GLUT function to register this callback is *glutEntryFunc* and the syntax is as follows:

---

void glutEntryFunc(void (*func)(int state));

Parameters:

- func – the function that will handle these events.

---

The parameter of the function that will handle these events tells us if the mouse has entered of left the window region. GLUT defines two constants that can be used in the application:

- GLUT_LEFT
- GLUT_ENTERED

Note: This doesn't work exactly as it says in Microsoft Windows, this is because in Microsoft's OS the focus is changed with a mouse click. Although you can change this is your own system using some tools from Microsoft, others are likely to have the standard setting so its probably better if you don't use this feature in Microsoft Windows to detect when the mouse enters/leaves the window.

Prev: The Code So Far Next: Moving the Camera III

**11 Responses to "The Mouse"**

1. **Pavankumar** says:
20/07/2012 at 6:21 AM

Hi

Is there a way to handle the scroll wheel movements, like scroll up and scroll down??
Thank you

Reply

> **ARF** says:
> 20/07/2012 at 9:31 PM
>
> Yes, with freeglut.
>
> Reply

2. **Dougng** says:
05/12/2011 at 7:10 PM

This a good and well simplified tutorial. Thanks.
Please I need information about how I can add buttons to my GUI, and also make establish interactivity between the mouse click and the GLUT models in the GUI

Reply

3. **Narek** says:
27/05/2011 at 6:56 AM

Hi

thanks for this useful information

i have a problem with the mouse.i want to change the (x,y) of the mouse and make them change according to

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,203 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models with Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

Google Ads

# Lighthouse3d.com

MOSTLY ABOUT 3D GRAPHICS



## Mouse: Moving the Camera III

Prev: The Mouse Next: The Code So Far II

In the previous example we changed the orientation of the camera with the keyboard. In here we are going to use the mouse instead.

When the user presses the mouse left button we are going to record the X position of the mouse. As the mouse moves will check the new X position, and based on the difference we'll set a variable <I>deltaAngle</I>. This variable will be added to the initial angle to compute the direction of the camera.

A variable to store the X position where the mouse is clicked is also required.

```
float deltaAngle = 0.0f;
int xOrigin = -1;
```

Note that xOrigin is initialized to a value that never occurs when the mouse is pressed (it must be at least zero). This will enable us to distinguish if the user is pressing the left button or any other button.

The next function is responsible for processing the button state changes:

```
void mouseButton(int button, int state, int x, int y) {

	// only start motion if the left button is pressed
	if (button == GLUT_LEFT_BUTTON) {

		// when the button is released
		if (state == GLUT_UP) {
			angle += deltaAngle;
			xOrigin = -1;
		}
		else  {// state = GLUT_DOWN
			xOrigin = x;
		}
	}
}
```

Notice that the var *xOrigin* is set to -1 when the left button is released.

The function to process the motion of the mouse is now presented:

```
void mouseMove(int x, int y) {

	// this will only be true when the left button is down
	if (xOrigin >= 0) {

		// update deltaAngle
		deltaAngle = (x - xOrigin) * 0.001f;

		// update camera's direction
		lx = sin(angle + deltaAngle);
		lz = -cos(angle + deltaAngle);
	}
}
```

In the main function we have to register the two new callback functions:

```
int main(int argc, char **argv) {
```

```
	// init GLUT and create window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D - GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);
	glutReshapeFunc(changeSize);
	glutIdleFunc(renderScene);

	glutIgnoreKeyRepeat(1);
	glutKeyboardFunc(processNormalKeys);
	glutSpecialFunc(pressKey);
	glutSpecialUpFunc(releaseKey);

	// here are the two new functions
	glutMouseFunc(mouseButton);
	glutMotionFunc(mouseMove);

	// OpenGL init
	glEnable(GL_DEPTH_TEST);

	// enter GLUT event processing cycle
	glutMainLoop();

	return 1;
}
```

Check the next page to see the full source code.

Prev: The Mouse

Next: The Code So Far II

**5 Responses to "Mouse: Moving the Camera III"**

1. intr says:
15/01/2012 at 9:34 PM

Hi, maybe you can post some examples how to move camera in all directions? in this tutorial you can move it only in x and z axis… but if i want it to move y axis too?
thank you.

Reply

2. Mario says:
28/06/2011 at 11:53 AM

Very useful tutorial, thanks!!
However, if I try to run this program (I got no errors from the compiler), it appears a window that says:
"The procedure entry point _glutIgnoreKeyRepeat@4 could not be located in the dynamic link library GULT32.dll"….
Do you have any idea of what does it means?
Thanks
mario

Reply

ARF says:
28/06/2011 at 12:36 PM

Hi, if you have translated the message as is then the culprit is "GULT32.dll". It should be GLUT32.dll. Otherwise can you provide more info on your setup please?

Reply

Mario says:
29/06/2011 at 6:53 AM

Sorry, the message was: "The procedure entry point _glutIgnoreKeyRepeat@4 could not be located in the dynamic link library GLUT32.dll"
I use Visual C++ 2008 Express edition, with the GLUT 3.7

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,203 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models with Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

# Lighthouse3d.com

MOSTLY ABOUT 3D GRAPHICS



## The Code So Far II

Prev: Moving the Camera III Next: Popup Menus

```
#include <stdlib.h>
#include <math.h>

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

// angle of rotation for the camera direction
float angle = 0.0f;

// actual vector representing the camera's direction
float lx=0.0f,lz=-1.0f;

// XZ position of the camera
float x=0.0f, z=5.0f;

// the key states. These variables will be zero
//when no key is being presses
float deltaAngle = 0.0f;
float deltaMove = 0;
int xOrigin = -1;

void changeSize(int w, int h) {

        // Prevent a divide by zero, when window is too short
        // (you cant make a window of zero width).
        if (h == 0)
                h = 1;

        float ratio =  w * 1.0 / h;

        // Use the Projection Matrix
        glMatrixMode(GL_PROJECTION);

        // Reset Matrix
        glLoadIdentity();

        // Set the viewport to be the entire window
        glViewport(0, 0, w, h);

        // Set the correct perspective.
        gluPerspective(45.0f, ratio, 0.1f, 100.0f);

        // Get Back to the Modelview
        glMatrixMode(GL_MODELVIEW);
}

void drawSnowMan() {

        glColor3f(1.0f, 1.0f, 1.0f);

// Draw Body
        glTranslatef(0.0f ,0.75f, 0.0f);
        glutSolidSphere(0.75f,20,20);

// Draw Head
```

```
        glTranslatef(0.0f, 1.0f, 0.0f);
        glutSolidSphere(0.25f,20,20);

// Draw Eyes
        glPushMatrix();
        glColor3f(0.0f,0.0f,0.0f);
        glTranslatef(0.05f, 0.10f, 0.18f);
        glutSolidSphere(0.05f,10,10);
        glTranslatef(-0.1f, 0.0f, 0.0f);
        glutSolidSphere(0.05f,10,10);
        glPopMatrix();

// Draw Nose
        glColor3f(1.0f, 0.5f , 0.5f);
        glRotatef(0.0f,1.0f, 0.0f, 0.0f);
        glutSolidCone(0.08f,0.5f,10,2);
}

void computePos(float deltaMove) {

        x += deltaMove * lx * 0.1f;
        z += deltaMove * lz * 0.1f;
}

void renderScene(void) {

        if (deltaMove)
                computePos(deltaMove);

        // Clear Color and Depth Buffers
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Reset transformations
        glLoadIdentity();
        // Set the camera
        gluLookAt(      x, 1.0f, z,
                        x+lx, 1.0f,  z+lz,
                        0.0f, 1.0f,  0.0f);

// Draw ground

        glColor3f(0.9f, 0.9f, 0.9f);
        glBegin(GL_QUADS);
                glVertex3f(-100.0f, 0.0f, -100.0f);
                glVertex3f(-100.0f, 0.0f,  100.0f);
                glVertex3f( 100.0f, 0.0f,  100.0f);
                glVertex3f( 100.0f, 0.0f, -100.0f);
        glEnd();

// Draw 36 SnowMen

        for(int i = -3; i < 3; i++)
                for(int j=-3; j < 3; j++) {
                     glPushMatrix();
                     glTranslatef(i*10.0,0,j * 10.0);
                     drawSnowMan();
                     glPopMatrix();
                }
        glutSwapBuffers();
}

void processNormalKeys(unsigned char key, int xx, int yy) {

        if (key == 27)
              exit(0);
}

void pressKey(int key, int xx, int yy) {

       switch (key) {
             case GLUT_KEY_UP : deltaMove = 0.5f; break;
             case GLUT_KEY_DOWN : deltaMove = -0.5f; break;
       }
}

void releaseKey(int key, int x, int y) {

        switch (key) {
```

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,203 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models with Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

Google Ads

```
			case GLUT_KEY_UP :
			case GLUT_KEY_DOWN : deltaMove = 0;break;
		}
}

void mouseMove(int x, int y) {

	 // this will only be true when the left button is down
	 if (xOrigin >= 0) {

		// update deltaAngle
		deltaAngle = (x - xOrigin) * 0.001f;

		// update camera's direction
		lx = sin(angle + deltaAngle);
		lz = -cos(angle + deltaAngle);
	}
}

void mouseButton(int button, int state, int x, int y) {

	// only start motion if the left button is pressed
	if (button == GLUT_LEFT_BUTTON) {

		// when the button is released
		if (state == GLUT_UP) {
			angle += deltaAngle;
			xOrigin = -1;
		}
		else  {// state = GLUT_DOWN
			xOrigin = x;
		}
	}
}

int main(int argc, char **argv) {

	// init GLUT and create window
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("Lighthouse3D - GLUT Tutorial");

	// register callbacks
	glutDisplayFunc(renderScene);
	glutReshapeFunc(changeSize);
	glutIdleFunc(renderScene);

	glutIgnoreKeyRepeat(1);
	glutKeyboardFunc(processNormalKeys);
	glutSpecialFunc(pressKey);
	glutSpecialUpFunc(releaseKey);

	// here are the two new functions
	glutMouseFunc(mouseButton);
	glutMotionFunc(mouseMove);

	// OpenGL init
	glEnable(GL_DEPTH_TEST);

	// enter GLUT event processing cycle
	glutMainLoop();

	return 1;
}
```

As usual the code is available in the download section.

Prev: Moving the Camera III Next: Popup Menus

**15 Responses to “The Code So Far II”**

1. **Peon** says:
23/05/2013 at 1:50 PM

Here is my code. Just some differences. And one problem: I can't reply on your page by Firefox.

```
#include
#include
#include

float angle = 0;
float xView = 0, yView = 2, zView = 7;
float lx = 0, ly = 0, lz = -1;

int xOrigin = -1, yOrigin = -1, zOrigin = -1;

float deltaAngle = 0;
//float deltaMove = 0;

void changeSize(int width, int height);
void drawSnowMan();
void display(void);
void processNormalKeys(unsigned char key, int x, int y);
void processSpecialKeys(int key, int x, int y);

void mouseMove(int x, int y);
void mouseButton(int button, int state, int x, int y);

void changeSize(int width, int height)
{
if (height == 0)
{
height = 1;
}

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glViewport(0, 0, width, height);
float aspect = width / height;
gluPerspective(45, aspect, 0.1, 100);
glMatrixMode(GL_MODELVIEW);
}

void drawSnowMan()
{
glColor3f(1, 1, 1);
glTranslatef(0, 0.75, 0);
glutWireSphere(0.75, 10, 10);
glTranslatef(0, 1, 0);
glutSolidSphere(0.25, 10, 10);
glPushMatrix();

//Draw eyes
glTranslatef(0.1, 0, 0.23);
glColor3f(0, 0, 1);
glutSolidSphere(0.03, 10, 10);
glTranslatef(-0.2, 0, 0);
glutSolidSphere(0.03, 10, 10);

glPopMatrix();
glTranslatef(0, -0.05, 0.25);
glColor3f(1, 0, 0);
glutSolidCone(0.025, 0.25, 4, 4);

}

void display(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

gluLookAt(xView, yView, zView,
xView + lx, yView + ly, zView + lz,
0, 1, 0);
glRotatef(angle, 0, 1, 0);

glColor3f(1, 0, 0);
glLineWidth(3);
glBegin(GL_LINES);
glVertex3f(0, 0, 0);
glVertex3f(1.5, 0, 0);
glVertex3f(0, 0, 0);
glVertex3f(0, 3, 0);
glVertex3f(0, 0, 0);
glVertex3f(0, 0, 1.5);
glEnd();

glColor3f(0.5, 0.5, 0.5);
glBegin(GL_QUADS);
glVertex3f(-100, 0, -100);
glVertex3f(-100, 0, 100);
```

```
glVertex3f(100, 0, 100);
glVertex3f(100, 0, -100);
glEnd();

drawSnowMan();

glutSwapBuffers();

}

void processNormalKeys(unsigned char key, int x, int y)
{
switch (key)
{
case 27:
exit(0);
break;
}
}

void processSpecialKeys(int key, int x, int y)
{
switch (key)
{
case GLUT_KEY_LEFT:
angle -= 0.5;
break;

case GLUT_KEY_RIGHT:
angle += 0.5;
break;

case GLUT_KEY_UP:
zView -= 0.5;
break;

case GLUT_KEY_DOWN:
zView += 0.5;
break;
}
}

void mouseMove(int x, int y)
{
if (xOrigin >= 0 || zOrigin >= 0)
{
deltaAngle = acos(((float)xOrigin * x + yOrigin * y) / (sqrt((float)xOrigin * xOrigin + yOrigin * yOrigin) * sqrt((float)x
* x + y * y)));
if (x <= xOrigin)
{
angle -= deltaAngle * 100;
xOrigin = x;
yOrigin = y;
}
else
{
angle += deltaAngle * 100;
xOrigin = x;
yOrigin = y;
}
}
}

void mouseButton(int button, int state, int x, int y)
{
if (button == GLUT_LEFT_BUTTON)
{
if (state == GLUT_DOWN)
{
xOrigin = x;
yOrigin = y;
}
else
{
xOrigin = -1;
yOrigin = -1;
}
}
}

int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGB);
glutInitWindowPosition(0, 0);
glutInitWindowSize(680, 680);
glutCreateWindow ("Example 2.4 – Mouse");

glutDisplayFunc(display);
glutReshapeFunc(changeSize);
glutIdleFunc(display);
glutKeyboardFunc(processNormalKeys);
```

```
glutSpecialFunc(processSpecialKeys);
//glutIgnoreKeyRepeat(1);
glutMouseFunc(mouseButton);
glutMotionFunc(mouseMove);

glEnable(GL_DEPTH_TEST);

glutMainLoop();

return 0;

}
```

Reply

2. **Cristi** says:
21/09/2012 at 4:04 PM

You have to be careful when you name your variables. I spent like one hour until i figure out that x from expresion (x – xOrigin) is the local x from the function and the same thing at xOrigin = x. Very good tutorial !! I also want to make the camera to move up and down and, as you said, I need another angle variable to move horizontaly

Reply

3. **Carl** says:
30/05/2012 at 8:58 PM

…has been removed from renderScene().

Sorry about that :/

Reply

4. **Carl** says:
30/05/2012 at 8:58 PM

Note that
:

```
if (deltaAngle)
computeDir(deltaAngle);
```

Reply

5. **Morten** says:
17/09/2011 at 10:48 AM

I would also like to thank for for this awesome guide. It has helped me immensely. By adopting the same techniques and functions I have been able to create a decent fractal generator.

Reply

6. **Morten** says:
14/09/2011 at 3:05 PM

It seems to me that, in the function mouseButton the line “angle += deltaAngle” is redundant, since nothing happens if I remove it. Is there another not-obvious reason for this?

Also, as it is written in the example the camera continues to rotate when I release the left mouse button at the same “speed” as before I released it. If I change the code under the second if, then I can make it stop rotating when I release the left mouse button:

```
void mouseButton(int button, int state, int x, int y) {

// only start motion if the left button is pressed
if (button == GLUT_LEFT_BUTTON) {

// when the button is released
if (state == GLUT_UP) {
//angle += deltaAngle;
deltaAngle = 0; // Stops the camera.
xOrigin = -1;
}
else {// state = GLUT_DOWN
xOrigin = x;
}
}
}
```

Reply

# Popup Menus

Prev: The Code So Far II Next: Sub Menus

Pop-up menus are also a part of GLUT. Although not all the features of the Pop-up menus usually found in windows systems are implemented, this part of GLUT does a great job. Adding menus to an application provides an easier way to interact and select options than the keyboard, avoiding having to remember all those keys.

The first thing we must do is to create a menu. GLUT's function *glutCreateMenu* has the following syntax:

```
int glutCreateMenu(void (*func)(int value));
```

Parameters:

- func – the function that will handle the menu events for the newly created menu.

The return value for this function is the menu identifier.

We can have as many menus as we want in our application. And for each menu a callback function is specified, although we can specify the same function for all our menus. Next we add some entries to the menu. The function to do this is *glutAddMenuEntry*.

```
void glutAddMenuEntry(char *name, int value);
```

Parameters:

- name – the string that will show up in the menu.
- value – this is the value that will be the returned to the callback function when the menu entry is selected.

This function appends the entry to the previously added entries, i.e. to the bottom of the menu. In GLUT there is no function to add an entry to the middle of the menu. Remember that GLUT doesn't pretend to be a complete API. GLUT is designed to make our lives easier when designing prototypes and it does an excellent job at that.

OK, so now you have a pop-up menu, but there's one last thing we must do: attach the menu to a mouse button, that is we must specify when the pop-up menu will appear. Using GLUT you can cause the menu to appear when a mouse button is pressed. The function to establish this relationship is *glutAttachMenu*.

```
void glutAttachMenu(int button);
```

Parameters:

- button – an integer that specifies which button the menu will be attached to.

The *button* should have one of the following values:

- GLUT_LEFT_BUTTON
- GLUT_MIDDLE_BUTTON
- GLUT_RIGHT_BUTTON

So here is a function that exemplifies the usage of all the above functions.

```
...
#define RED 1
#define GREEN 2
#define BLUE 3
#define ORANGE 4
...

void createGLUTMenus() {

	int menu;

	// create the menu and
	// tell glut that "processMenuEvents" will
	// handle the events
	menu = glutCreateMenu(processMenuEvents);

	//add entries to our menu
	glutAddMenuEntry("Red",RED);
	glutAddMenuEntry("Blue",BLUE);
	glutAddMenuEntry("Green",GREEN);
	glutAddMenuEntry("Orange",ORANGE);

	// attach the menu to the right button
	glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

Now we'll write the function to process the menu events. As you probably guessed by now, we're going to set a color using our menu.

Note that our function must be called *processMenuEvents*, thats the name we provided when we created the menu in the function above. Furthermore, looking at the syntax of *glutCreateMenu* we know that it will have a parameter representing the selected menu item.

```
void processMenuEvents(int option) {

	switch (option) {
		case RED :
			red = 1.0f;
			green = 0.0f;
			blue = 0.0f; break;
		case GREEN :
			red = 0.0f;
			green = 1.0f;
			blue = 0.0f; break;
		case BLUE :
			red = 0.0f;
			green = 0.0f;
			blue = 1.0f; break;
		case ORANGE :
			red = 1.0f;
			green = 0.5f;
			blue = 0.5f; break;
	}
}
```

The only thing left to do is to add a call *createGLUTMenus* in our main function.

Before we end this introduction to glut pop-up menus, we're going to look at two more functions. The first one allows you to break the relationship between a mouse button and a menu. Previously we attached the menu to a mouse button using the glut's function *glutAttachMenu*. In some applications it may be useful to break this association, i.e. when the user presses the mouse the menu no longer appears. In GLUT this is done with the function *glutDetachMenu*. This function causes GLUT to stop providing a menu when the mouse button is pressed. The syntax is as follows:

void glutDetachMenu(int button);

Parameters:

- button – the button to detach

The button parameter takes the same values as for the *glutAttachMenu*.

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,203 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models w ith Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

Google Ads

So in our previous example, if we want to free the mouse button, we could write:

```
...
glutDetachMenu(GLUT_RIGHT_BUTTON);
...
```

And finally, if we want to free the resources used by the menu then we can destroy it. The GLUT function to do this is *glutDestroyMenu* and it has the following syntax:

---

void glutDestroyMenu(int menuIdentifier);

Parameters:

- menuIdentifier – this is the id of the menu to destroy. It must be the same value as the id returned by the function *glutCreateMenu*.

---

So this is it, you now know the basics of building menus with GLUT. Next we'll explore some more features of the pop-up menus.

Prev: The Code So Far II Next: Sub Menus

**Leave a Reply**

Name (required)

E-mail (required)

URI

Your Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

Submit Comment

Notify me of follow-up comments by email

Notify me of new posts by email

## Sub Menus

Prev: Popup Menus Next: Modifying a Menu

In the previous section we saw how to build a simple menu, and process the events generated by a user selection. Now we're going to see how to add a cascade submenu.

A submenu is created with the same function as a menu. Therefore we use the function *glutCreateMenu*, see the previous section, to create submenus as well as menus. Then we must add the submenu as an entry to the menu. This is done in GLUT using *glutAddSubMenu*.

void glutAddSubMenu(char *entryName, int menuIndex);

Parameters:

- entryName – The name of the submenu entry in the menu
- menuIndex – The index of the submenu, this is the value that we get as a return value when calling *glutCreateMenu* for the submenu.

This function adds an entry to the end of the menu. When selected, the new entry will open a submenu.

The following piece of code illustrates the usage of the above function:

```
void createPopupMenus() {

        shrinkMenu = glutCreateMenu(processShrinkMenu);
        glutAddMenuEntry("Shrink",SHRINK);
        glutAddMenuEntry("NORMAL",NORMAL);

        fillMenu = glutCreateMenu(processFillMenu);
        glutAddMenuEntry("Fill",FILL);
        glutAddMenuEntry("Line",LINE);

        colorMenu = glutCreateMenu(processColorMenu);
        glutAddMenuEntry("Red",RED);
        glutAddMenuEntry("Blue",BLUE);
        glutAddMenuEntry("Green",GREEN);
        glutAddMenuEntry("Orange",ORANGE);

        mainMenu = glutCreateMenu(processMainMenu);
        glutAddSubMenu("Polygon Mode", fillMenu);
        glutAddSubMenu("Color", colorMenu);

        // attach the menu to the right button
        glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

Using the code above, when the user pressed the right mouse button a menu with two options would be presented: "Polygon Mode" and "Color". Pressing on "Color", a submenu appears with three items: "Red", "Blue", "Green", and "Orange".

Prev: Popup Menus Next: Modifying a Menu

### Leave a Reply

Name (required)

E-mail (required)

URI

Your Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

Submit Comment

Notify me of follow-up comments by email

Notify me of new posts by email

Shader Examples 0 comments | 11,288 views

Keyboard Example: Moving the Camera 0 comments | 11,203 views

OpenGL Framebuffer Objects 0 comments | 10,785 views

Importing 3D Models with Assimp 0 comments | 10,671 views

Keyboard 0 comments | 10,306 views

### Links

- AMD Developer Central
- Codeflow
- Flipcode
- G-Truc Creation
- GameDev.net
- Geeks3D
- Hugo Elias' Site
- Humus 3D
- Iñigo Quilez Site
- Khronos Group
- Nehe
- nVidia Developer
- OpenGL.org
- Paul Bourke's Site
- Real-Time Rendering

### Google Ads

# Lighthouse3d.com

MOSTLY ABOUT 3D GRAPHICS



## Modifying a Menu

Prev: Sub Menus Next: Swapping Menus

In certain situations a change of a menu entry may be desirable. GLUT allows us to change and delete menu entries. To alter a menu entry use:

---

void glutChangeToMenuEntry(int entry, char *name, int value);

Parameters:

- entry – the index of the entry, this must be between 1 and the total number of entries
- name – the name of the new entry
- value – The value that will be return to the callback function when the entry is selected.

---

To swap a submenu use:

---

void glutChangeToSubMenu(int entry, char *name, int menu);

Parameters:

- entry – the index of the entry, this must be between 1 and the total number of entries
- name – the name of the new entry
- menu – The menu index to be used.

---

To following function deletes an item.

---

void glutRemoveMenuItem(int entry);

Parameters:

- entry – the index of the entry, this must be between 1 and the total number of entries

---

One last thing, you can query at any time the number of items of the current menu with *glutGet*.

The next example shows an example of changing a menu:

```
void processMenuEvents(int option) {

        red = 0.0;
        green = 0.0;
        blue = 0.0;

        switch (option) {
                case RED :
                        red = 1.0; break;
                case GREEN :
                        green = 1.0; break;
                case BLUE :
                        blue = 1.0; break;
                case WHITE :
                        red = 1.0;
                        green = 1.0;
                        blue = 1.0; break;
        }
```

```
}

void processKeys(unsigned char c, int x, int y) {

	int num = glutGet(GLUT_MENU_NUM_ITEMS);
	switch (c) {
		case 'a':
			glutChangeToMenuEntry(1,"Blue",BLUE);
			glutChangeToMenuEntry(3,"Red",RED);
			break;
		case 'b':
			glutChangeToMenuEntry(3,"Blue",BLUE);
			glutChangeToMenuEntry(1,"Red",RED);
			break;
		case 'c':
			if (num > 3)
				glutRemoveMenuItem(num);
			break;
		case 'd': if (num == 3)
				glutAddMenuEntry("White",WHITE);
			break;
	}
	glutSetMenu(menu);
}

void createGLUTMenus() {

	menu = glutCreateMenu(processMenuEvents);
	glutAddMenuEntry("Red",RED);
	glutAddMenuEntry("Green",GREEN);
	glutAddMenuEntry("Blue",BLUE);
	glutAddMenuEntry("White",WHITE);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

Note that we changed the menu in the keyboard callback function as opposed to the menu callback function. This is because we shouldn't do any changes to a menu while it is in use. A menu is in use until the callback is over, so we couldn't change the menu's structure inside the menu's own callback.

As mentioned before, when a menu is in use it can't, or at least it shouldn't, be altered. In order to prevent messing up we must make sure if a menu is not in use before we change the menu entries. GLUT allows us to register a callback function that will ba called whenever a menu pops-up, and when it goes away. The function to register the callback is *glutMenuStatusFunc*.

---

void glutMenuStatusFunc(void (*func)(int status, int x, int y);

Parameters:

- func – the name of the callback function

---

This function can be called in our main function, so we'll just add it there.

As seen by the signature of *glutMenuStatusFunc* the callback function must take three parameters. These are:

- status – one of GLUT_MENU_IN_USE or GLUT_MENU_NOT_IN_USE
- x – The left coordinate of the menu relative to the window client area.
- y – The top coordinate of the menu relative to the window client area.

Bellow an example function is presented where a flag is set when the menu is in use.

```
void processMenuStatus(int status, int x, int y) {

	if (status == GLUT_MENU_IN_USE)
		flag = 1;
	else
		flag = 0;
}
```

We can now use this flag when processing keyboard events as in the next example:

```
void processKeys(unsigned char c, int x, int y) {
```

```
	if (!flag) {
		int num = glutGet(GLUT_MENU_NUM_ITEMS);
		switch (c) {
			case 'a':
				glutChangeToMenuEntry(1,"Blue",BLUE);
				glutChangeToMenuEntry(3,"Red",RED);
				break;
			case 'b':
				glutChangeToMenuEntry(3,"Blue",BLUE);
				glutChangeToMenuEntry(1,"Red",RED);
				break;
			case 'c':
				if (num > 3)
					glutRemoveMenuItem(num);
				break;
			case 'd': if (num == 3)
					glutAddMenuEntry("White",WHITE);
				break;
		}
	}
}
```

Prev: Sub Menus Next: Swapping Menus

**One Response to “Modifying a Menu”**

1. Hatem says:
   27/01/2014 at 7:50 AM

   Thanks, this is really helpful

   Reply

**Leave a Reply**

Name (required)

E-mail (required)

URI

Your Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

Submit Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.

## Swapping Menus

Prev: Modifying a Menu Next: The Code So Far III

GLUT even allows us to change an entire menu in the middle of our application. Two functions are provided: *glutSetMenu* and *glutGetMenu*. The syntax for the former is:

void glutSetMenu(int menu);

Parameters:

- menu – the index of a previously created menu

This function allows us to swap a menu, for instance if there is a change in the context of the application.

The syntax for *glutGetMenu* is as follows:

int glutGetMenu(void);

This function returns the index of the current menu. Next we present an example where we use two menus that are swapped when the user presses F1.

```
void processSpecialKeys(int c, int x, int y) {

	if (!flag) {
		if (c == GLUT_KEY_F1) {
			int x = glutGetMenu();
			if (x == menu1)
				glutSetMenu(menu2);
			else
				glutSetMenu(menu1);
			// don't forget to attach the menu!!!
			glutAttachMenu(GLUT_RIGHT_BUTTON);
		}
	}
}

void createGLUTMenus() {

	menu2 = glutCreateMenu(processMenuEvents);
	glutAddMenuEntry("Blue",BLUE);
	glutAddMenuEntry("Green",GREEN);
	glutAddMenuEntry("Red",RED);

	menu1 = glutCreateMenu(processMenuEvents);
	glutAddMenuEntry("Red",RED);
	glutAddMenuEntry("Green",GREEN);
	glutAddMenuEntry("Blue",BLUE);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

Prev: Modifying a Menu Next: The Code So Far III

### Leave a Reply

Name (required)

E-mail (required)

URI

Your Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

Submit Comment

Notify me of follow-up comments by email

Notify me of new posts by email

Shader Examples 0 comments | 11,288 views

Keyboard Example: Moving the Camera 0 comments | 11,203 views

OpenGL Framebuffer Objects 0 comments | 10,785 views

Importing 3D Models with Assimp 0 comments | 10,671 views

Keyboard 0 comments | 10,306 views

**Links**

- AMD Developer Central
- Codeflow
- Flipcode
- G-Truc Creation
- GameDev.net
- Geeks3D
- Hugo Elias' Site
- Humus 3D
- Iñigo Quilez Site
- Khronos Group
- Nehe
- nVidia Developer
- OpenGL.org
- Paul Bourke's Site
- Real-Time Rendering

**Google Ads**

## The Code So Far III

Prev: Swapping Menus Next: Bitmap Fonts

In here we're going to try to include some of the stuff we've presented before in our little application. We're going to add menus to the application, submenus, and menu swapping.

Just copy and paste the following code into your project, or go to the download section, and try it. The right mouse button should open a menu. The keys 's' and 'c' will affect the menu options.

```
#include <stdlib.h>
#include <math.h>

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

// angle of rotation for the camera direction
float angle = 0.0f;

// actual vector representing the camera's direction
float lx=0.0f,lz=-1.0f;

// XZ position of the camera
float x=0.0f, z=5.0f;

// the key states. These variables will be zero
//when no key is being presses
float deltaAngle = 0.0f;
float deltaMove = 0;
int xOrigin = -1;

// Constant definitions for Menus
#define RED 1
#define GREEN 2
#define BLUE 3
#define ORANGE 4

#define FILL 1
#define LINE 2

#define SHRINK 1
#define NORMAL 2

// Pop up menu identifiers
int fillMenu, shrinkMenu, mainMenu, colorMenu;

// color for the nose
float red = 1.0f, blue=0.5f, green=0.5f;

// scale of snowman
float scale = 1.0f;

// menu status
int menuFlag = 0;

void changeSize(int w, int h) {

        // Prevent a divide by zero, when window is too short
        // (you cant make a window of zero width).
        if (h == 0)
```

```
                h = 1;

        float ratio =  w * 1.0 / h;

        // Use the Projection Matrix
        glMatrixMode(GL_PROJECTION);

        // Reset Matrix
        glLoadIdentity();

        // Set the viewport to be the entire window
        glViewport(0, 0, w, h);

        // Set the correct perspective.
        gluPerspective(45.0f, ratio, 0.1f, 100.0f);

        // Get Back to the Modelview
        glMatrixMode(GL_MODELVIEW);
}

void drawSnowMan() {

        glScalef(scale, scale, scale);
        glColor3f(1.0f, 1.0f, 1.0f);

// Draw Body
        glTranslatef(0.0f ,0.75f, 0.0f);
        glutSolidSphere(0.75f,20,20);

// Draw Head
        glTranslatef(0.0f, 1.0f, 0.0f);
        glutSolidSphere(0.25f,20,20);

// Draw Eyes
        glPushMatrix();
        glColor3f(0.0f,0.0f,0.0f);
        glTranslatef(0.05f, 0.10f, 0.18f);
        glutSolidSphere(0.05f,10,10);
        glTranslatef(-0.1f, 0.0f, 0.0f);
        glutSolidSphere(0.05f,10,10);
        glPopMatrix();

// Draw Nose
        glColor3f(red, green, blue);
        glRotatef(0.0f,1.0f, 0.0f, 0.0f);
        glutSolidCone(0.08f,0.5f,10,2);

        glColor3f(1.0f, 1.0f, 1.0f);
}

void computePos(float deltaMove) {

        x += deltaMove * lx * 0.1f;
        z += deltaMove * lz * 0.1f;
}

void renderScene(void) {

        if (deltaMove)
                computePos(deltaMove);

        // Clear Color and Depth Buffers
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Reset transformations
        glLoadIdentity();
        // Set the camera
        gluLookAt(      x, 1.0f, z,
                        x+lx, 1.0f,  z+lz,
                        0.0f, 1.0f,  0.0f);

// Draw ground

        glColor3f(0.9f, 0.9f, 0.9f);
        glBegin(GL_QUADS);
                glVertex3f(-100.0f, 0.0f, -100.0f);
                glVertex3f(-100.0f, 0.0f,  100.0f);
                glVertex3f( 100.0f, 0.0f,  100.0f);
```

Shader Examples 0 comments | 11,288 views
Keyboard Example: Moving the Camera 0 comments | 11,203 views
OpenGL Framebuffer Objects 0 comments | 10,785 views
Importing 3D Models w ith Assimp 0 comments | 10,671 views
Keyboard 0 comments | 10,306 views

Links

AMD Developer Central
Codeflow
Flipcode
G-Truc Creation
GameDev.net
Geeks3D
Hugo Elias' Site
Humus 3D
Iñigo Quilez Site
Khronos Group
Nehe
nVidia Developer
OpenGL.org
Paul Bourke's Site
Real-Time Rendering

Google Ads

```
                glVertex3f( 100.0f, 0.0f, -100.0f);
        glEnd();

// Draw 36 SnowMen

        for(int i = -3; i < 3; i++)
                for(int j=-3; j < 3; j++) {
                        glPushMatrix();
                        glTranslatef(i*10.0f, 0.0f, j * 10.0f);
                        drawSnowMan();
                        glPopMatrix();
                }
        glutSwapBuffers();
}

// ------------------------------------
//             KEYBOARD
// ------------------------------------

void processNormalKeys(unsigned char key, int xx, int yy) {

        glutSetMenu(mainMenu);
        switch (key) {
                case 27:
                        glutDestroyMenu(mainMenu);
                        glutDestroyMenu(fillMenu);
                        glutDestroyMenu(colorMenu);
                        glutDestroyMenu(shrinkMenu);
                        exit(0);
                        break;

                case 's':
                        if (!menuFlag)
                          glutChangeToSubMenu(2,"Shrink",shrinkMenu);
                        break;
                case 'c':
                        if (!menuFlag)
                                glutChangeToSubMenu(2,"Color",colorMenu);
                        break;
        }
        if (key == 27)
                exit(0);
}

void pressKey(int key, int xx, int yy) {

        switch (key) {
                case GLUT_KEY_UP : deltaMove = 0.5f; break;
                case GLUT_KEY_DOWN : deltaMove = -0.5f; break;
        }
}

void releaseKey(int key, int x, int y) {

        switch (key) {
                case GLUT_KEY_UP :
                case GLUT_KEY_DOWN : deltaMove = 0;break;
        }
}

// ------------------------------------
//             MOUSE
// ------------------------------------

void mouseMove(int x, int y) {

        // this will only be true when the left button is down
        if (xOrigin >= 0) {

                // update deltaAngle
                deltaAngle = (x - xOrigin) * 0.001f;

                // update camera's direction
                lx = sin(angle + deltaAngle);
                lz = -cos(angle + deltaAngle);
        }
}
```

```
void mouseButton(int button, int state, int x, int y) {

	// only start motion if the left button is pressed
	if (button == GLUT_LEFT_BUTTON) {

		// when the button is released
		if (state == GLUT_UP) {
			angle += deltaAngle;
			xOrigin = -1;
		}
		else  {// state = GLUT_DOWN
			xOrigin = x;
		}
	}
}

// ------------------------------------
//             MENUS
// ------------------------------------

void processMenuStatus(int status, int x, int y) {

	if (status == GLUT_MENU_IN_USE)
		menuFlag = 1;
	else
		menuFlag = 0;
}

void processMainMenu(int option) {

	// nothing to do in here
	// all actions are for submenus
}

void processFillMenu(int option) {

	switch (option) {

		case FILL: glPolygonMode(GL_FRONT, GL_FILL); break;
		case LINE: glPolygonMode(GL_FRONT, GL_LINE); break;
	}
}

void processShrinkMenu(int option) {

	switch (option) {

		case SHRINK: scale = 0.5f; break;
		case NORMAL: scale = 1.0f; break;
	}
}

void processColorMenu(int option) {

	switch (option) {
		case RED :
			red = 1.0f;
			green = 0.0f;
			blue = 0.0f; break;
		case GREEN :
			red = 0.0f;
			green = 1.0f;
			blue = 0.0f; break;
		case BLUE :
			red = 0.0f;
			green = 0.0f;
			blue = 1.0f; break;
		case ORANGE :
			red = 1.0f;
			green = 0.5f;
			blue = 0.5f; break;
	}
}

void createPopupMenus() {

	shrinkMenu = glutCreateMenu(processShrinkMenu);
```

```
        glutAddMenuEntry("Shrink",SHRINK);
        glutAddMenuEntry("NORMAL",NORMAL);

        fillMenu = glutCreateMenu(processFillMenu);

        glutAddMenuEntry("Fill",FILL);
        glutAddMenuEntry("Line",LINE);

        colorMenu = glutCreateMenu(processColorMenu);
        glutAddMenuEntry("Red",RED);
        glutAddMenuEntry("Blue",BLUE);
        glutAddMenuEntry("Green",GREEN);
        glutAddMenuEntry("Orange",ORANGE);

        mainMenu = glutCreateMenu(processMainMenu);

        glutAddSubMenu("Polygon Mode", fillMenu);
        glutAddSubMenu("Color", colorMenu);
        // attach the menu to the right button
        glutAttachMenu(GLUT_RIGHT_BUTTON);

        // this will allow us to know if the menu is active
        glutMenuStatusFunc(processMenuStatus);
}

// ------------------------------------
//             MAIN
// ------------------------------------

int main(int argc, char **argv) {

        // init GLUT and create window
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(320,320);
        glutCreateWindow("Lighthouse3D - GLUT Tutorial");

        // register callbacks
        glutDisplayFunc(renderScene);
        glutReshapeFunc(changeSize);
        glutIdleFunc(renderScene);

        glutIgnoreKeyRepeat(1);
        glutKeyboardFunc(processNormalKeys);
        glutSpecialFunc(pressKey);
        glutSpecialUpFunc(releaseKey);

        // here are the two new functions
        glutMouseFunc(mouseButton);
        glutMotionFunc(mouseMove);

        // OpenGL init
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_CULL_FACE);

        // init Menus
        createPopupMenus();

        // enter GLUT event processing cycle
        glutMainLoop();

        return 1;
}
```

Prev: Swapping Menus Next: Bitmap Fonts

**9 Responses to “The Code So Far III”**

1. **Marcus** says:
   23/12/2012 at 7:03 PM