Introduction to OpenGL and GLUT

General Information

When we work with 2D/3D applications, we need a way to interface with the graphics hardware in order to perform certain graphics operations. The two major programming interfaces for real-time rendering are OpenGL and Direct3D.

Direct3D (announced in 1996) is part of a multimedia collection of API's called DirectX for the Windows operating system. It has been developed and maintained by Microsoft. The version that the latest graphics cards support is DirectX 11 (actually 11.2 for Windows 8.1). DirectX is the major API used for the creation of video games.

OpenGL is a cross platform rendering API **and only that** (developed by SGI in 1992 and now managed by the non-profit industry consortium Khronos Group) and does not include any functionality for sound, input or anything unrelated to graphics rendering. OpenGL is being used for games (you can view a list here here), but is also used in other fields such as computer-aided design (CAD), scientific visualization, etc. The latest OpenGL version is 4.5. Academically, the API of choice is usually OpenGL.

The version we will use is OpenGL 3.3 core (released in March 2010). It is found in hardware capable of supporting Direct3D 10.

Strictly speaking, OpenGL is a specification defined by the ARB board (part of the Khronos Group). Whenever a new version of OpenGL is released, the ARB board releases a new specification, but not code. The code is written by the hardware developers for each platform. For example, in Windows, the drivers are written by the hardware vendors (NVidia, ATI). The development libraries (header files, etc.) are available with Visual Studio as part of the Windows SDK which also comes as part of Visual Studio.

The Way It Works

States and GLContext

OpenGL is defined as a state machine which operates on a GLContext. There are a number of function calls (more than 300!!) that change some state or pass data to the GPU. Whenever a state is changed, it remains at that state until the programmer decides to change it.

Example:

```
// Enable depth test
glEnable(GL_DEPTH_TEST);

// Disable depth test (if this is not called, depth testing will remain
enabled!)
glDisable(GL_DEPTH_TEST);
```

Be ALWAYS careful with the states.

You can view the glew.h header file to see all the available functions.

Naming and Data Types

OpenGL follows a naming convention to help identify gl Commands. These use the prefix gl and initial capital letters for each command (glClearColor, etc.). There are also some letters appended at the end of the name of some functions such as glUniform2f, glUniform3fv, etc. These specify the type of data OpenGL expects to receive. The following table shows the most common along with their typedefs as shown at the top of the glew.h header file:

Suffix	**Data Type**	**Typical Corresponding C-Language Type**	**OpenGL Type Definition**
b	8-bit (1 byte)	signed char	GLbyte
s	16-bit (2 bytes)	short	GLshort
i	32-bit (4 bytes)	int	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

Furthermore, we can pass a pointer to a function or a vector the same way. These commands include the suffix v at the end of their name.

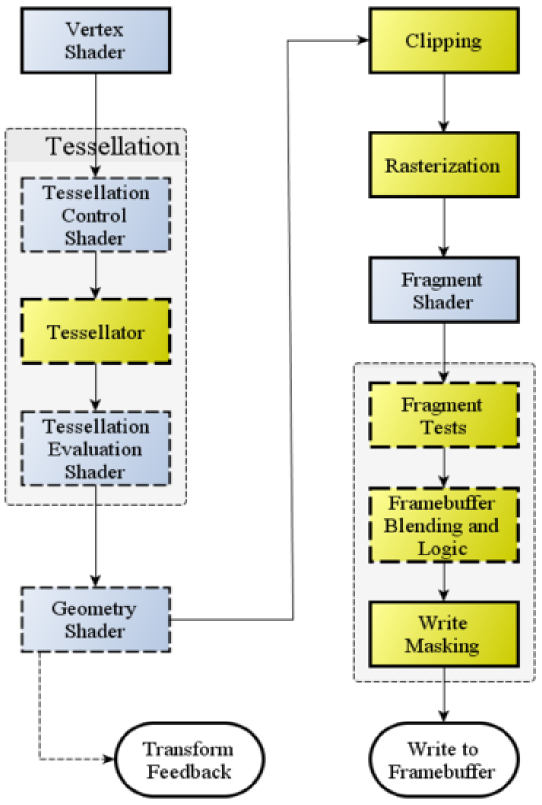
For example, glUniform3f and glUniform3fv are used as shown below:

```
// pass to shader uniform variable 0, the vec3 (0.0, 0.0, 1.0)
glUniform3f(0, 0.0f, 0.0f, 1.0f);
// can also pass as a pointer to a vector or array
GLfloat color_data[] = {0.0f, 0.0f, 1.0f};
glUniform3fv(0, color_data);
```

The Pipeline

The OpenGL rendering pipeline is shown in the following image

(Image taken from OpenGL wiki http://www.opengl.org/wiki/Rendering_Pipeline_Overview):



The general idea behind most basic OpenGL programs is the following:

On the CPU side:

- We set up our cameras, viewports and projection matrices which indicate how our data will be transformed in our scene and how it will be projected onto the screen.
- We set up our geometry information (vertex, normal, texturing, etc.) in data structures such as pointers to arrays. Our geometry information can be created either programmatically (e.g. hardcoded or procedurally) or imported by a 3D modeling software such as 3D Studio Max or Maya. These structures along with the transformations are sent to the OpenGL pipeline.

On the GPU side:

- The per-vertex operations are responsible for performing various calculations at the vertex level. For example, vertices (along with their associated data) are transformed from object space to eye or post-projective space. This part is programmable via a vertex shader. Vertex shaders are not optional.
- The primitive assembly stage receives vertices and groups them into primitives along with their adjacency information. The constructed primitives or patches (for tessellation shaders) are being sent down the pipeline for further processing.
- The geometry shader process primitives instead of vertices. They receive as input a list of vertices (and optional adjacency information) and output zero or more primitives. This stage is optional.

- The transformed and projected data are going through clipping and culling operations. Perspective divide and viewport transformations also happen here.
- The primitives that have reached this stage are going under rasterization and interpolation operations where each primitive is broken down into discrete elements, called fragments. These fragments serve as the input to the fragment shader.
- The fragment shader receives fragments as input and writes color and depth information to the buffers in the current framebuffers. Stencil values can also be written here, but this is not programmable. The fragment stage is mandatory.
- Per-fragment operations are performed after the fragment stage such as scissor testing, stencil testing, depth testing and blending.
- The final image is stored into the framebuffer which is then rendered onto the screen.

Other Libraries

FREEGLUT

OpenGL is a rendering API and only that. It does not provide any functionality for window creation, audio, input handling, etc. FREEGLUT stands for Free Graphics Library Utility Toolkit and provides some simple windowing functionality, input handling and menu creation. It is only recommended for basic OpenGL programs, but not for advanced ones. FREEGLUT operates using callback functions. By passing our functions as a parameter onto a FREEGLUT event handling function we inform FREEGLUT that we need this function to get called when this event occurs.

Initialization of a FREEGLUT application

```
// glutInit initializes the GLUT library.
glutInit(&argc, argv);

// set OpenGL 3.3 Context as the default
glutInitContextVersion(3, 3);

// Do not include any deprecated features (meaningless for OpenGL3.2+ core)
glutInitContextFlags(GLUT_FORWARD_COMPATIBLE);

// Do not include any features removed in previous versions (fixed-function
pipeline, etc)
glutInitContextProfile(GLUT_CORE_PROFILE);

// inform FREEGLUT that we want to leave the main loop when the X button is
pressed
// this is CRITICAL since we need to release any allocated memory before
// exiting the application
glutSetOption(
       GLUT_ACTION_ON_WINDOW_CLOSE,
       GLUT_ACTION_GLUTMAINLOOP_RETURNS
       );

// glutInitDisplayMode tells GLUT what sort of rendering context we want.
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);

// glutInitWindowSize tells GLUT the dimensions of the window we want to create
glutInitWindowSize(480,480);

// glutInitWindowPosition sets the parameters for the place on our screen we
// want the window to appear on.
glutInitWindowPosition(50,50);

// glutCreateWindow will create a window based on the setting set above
// with the name passed as an argument
glutCreateWindow("Hello OpenGL!");

// The display function
glutDisplayFunc(Render);

// The resize function
glutReshapeFunc(Resize);

// The basic keyboard handler
glutKeyboardFunc(Keyboard);

// The keyboard handler for special keys (F1-F12, arrows, Home, End, Insert,
// etc)
glutSpecialFunc(KeyboardSpecial);

// The mouse handler
glutMouseFunc(Mouse);

// Start the application!!!
glutMainLoop();

// Redraw whenever needed in the main loop
glutPostRedisplay();

// At the end of the main loop swap buffers
glutSwapBuffers();
```

Base Structure of a Game

```
// Entering application
int main()
{
   // call Init
   initGame();

   // enter Loop
   enterLoop();
}

// Here we perform the game engine’s initializations
void initGame()
{
   // usually, the first steps involve analysing the computer hardware in order
   // to auto-adjust settings parameters later on
   // Also init error logging as early as possible

   // Initialize window and game engine
   initWindow();
   // load scene parameters (from an ini file, a db structure, etc.)
   loadSceneParameters();
   // Initialize models, textures, rendering settings, save games, etc.
   initModelData();
   // Initialize the graphics engine
   initGraphics();
   // Initialize everything else required for creating the 3D world (physics,
   // audio, input, networking, etc.)
}

// After initialization, we enter the main game loop
// This keeps on executing, captures events, until the user exits the game
void enterLoop()
{
   bool userExitRequest = false;

   while(!userExitRequest)
   {
      // check for any user input
      handleUserInput();
      // update timers required for physics, animations, audio, etc.
      updateTimers();
      // update camera(s), physics, audio, etc.
      // render frame
      renderEverything();
      // swap buffers
      swapBuffers();
   }
}
```

More Information

You can find online an incredible amount of information regarding computer graphics, games, OpenGL and DirectX.

- http://www.opengl.org (OpenGL official page)
- http://www.opengl.org/sdk/docs/ (OpenGL and GLSL Documentation)
- http://www.opengl.org/sdk/docs/man/ (Latest OpenGL Reference Pages)
- http://www.opengl.org/sdk/docs/man3/ (OpenGL 3.3 Reference Pages)
- http://www.opengl.org/sdk/docs/manglsl/ (Online GLSL Reference Pages)
- http://www.opengl.org/wiki/Main_Page (OpenGL Wiki)
- http://freeglut.sourceforge.net/ (FREEGLUT official webpage)
- http://www.microsoft.com/en-us/download/details.aspx?id=6812 (DirectX SDK)
- http://www.opengl-tutorial.org/ (OpenGL 3.3+ tutorials)
- http://www.g-truc.net/ (OpenGL samples, reviews and more)
- http://www.mbsoftworks.sk/index.php?page=tutorials&series=1 (More OpenGL tutorials)
- http://www.arcsynthesis.org/gltut/ (Learning Modern 3D Graphics Programming)
- http://www.gamedev.net/ (Game developer community)