OpenGL basics compared to DirectX

Jin Li,

Jan 18th, 2005

Introduction to Computer Graphics

1. Review of the major differences between OpenGL 2.0 and DirectX 9.0

If we search online, we always see people asking questions about what they should use or learn for their application development: OpenGL or DirectX. Which one runs faster for the game or the CAD application? Such a question always results in endless discussions, which never reaches a definite answer. The easiest and also the best answer is: It depends on the requirements for your application. So to know a little more about similarities and major differences will help to make the right decision.

The most obvious difference is that DirectX is more than just a graphics API. DirectX contains tools to deal with sound, input, networking, and multimedia. On the other hand, OpenGL is strictly a graphics API. So the comparison is really between OpenGL and Direct3D.

Both APIs rely on the use of the traditional hardware rendering pipeline, which is based on the z buffer. Although this pipeline has been modified to adapt to advances in current hardware, the basic ideas remain valid.

Both OpenGL and DirectX describe vertices as a set of data consisting of coordinates in space that define the vertex location and possibly some other related data. Graphics primitives, such as points, lines, and triangles, are defined as an ordered set of vertices. There is a difference in how each API handles how vertices are combined to form primitives.

OpenGL(Open Graphics Library) has long been known to be easier to understand. OpenGL is cross-platform open standard. If you use OpenGL, you have a very good chance to be able to port your game to other platforms, like Windows, Linux, MacOS, etc. (But note that OpenGL itself does not include the programming interface to the graphical user interface. So for instance the code to create a 3D window is platform-specific.)

DirectX is a Windows-specific proprietary standard, known to be a bit harder to understand, but with Managed DirectX (which is DirectX wrapped into very handy .NET classes), it is much easier to use than before.

(The evolution of OpenGL is governed by the Architecture Review Board, an independent consortium of the leading graphics vendors. New functionalities are exposed through vendor-specific extensions (and there is quite a lot of them) whereas the base API changes only slowly.
DirectX is developed by Microsoft in cooperation with some vendors such as ATI and NVIDIA. New functionalities are exposed through API changes. . On top of that, Microsoft changed the API quite severely several times, mirroring some changes of

paradigms in graphics programming.

Learning some OpenGL basics helps us understand other people's previous work and also a large number of good online tutorials. Since the basic graphics knowledge are the same for both, I shall only focus on how basic 3D drawing is implemented in OpenGL with C++, compared to what we learned in Managed DirectX with C#.

2. How to start

To compile and link OpenGL programs, you'll need OpenGL header files and libraries.

Windows - If you are running Windows 98/NT/2000, then this library has already been installed in your system.

LINUX/BSD – There are several free implementations. Mesa is recommended by www.opengl.org. (http://www.opengl.org/resources/faq/getting_started.html)

More details about OpenGL on Linux can be read in the link below.
http://ls7-www.cs.uni-dortmund.de/~hinkenja/linux-opengl-faq.html#ChapterOpenGLandLinux

2.1 A fully installed Windows OpenGL development environment will look like this:

File	Location
GL.h glut.h GLU.h	[compiler]\include\GL (Microsoft Vc++ 6.0) [compiler]\include\gl (Microsoft Visual Studio .Net 2003)
Opengl32.lib glut32.lib glu32.lib	[compiler]\lib
Opengl32.dll glut32.dll glu32.dll	[system]

The other two often used libraries for OpenGL are GLU and GLUT for beginners. They can be downloaded from http://www.opengl.org.

For example, under Microsoft Visual Studio .NET, the installation path for GLUT is as following:
glut32.dll -> C:\Windows\System or C:\WinNT\System
glut32.lib -> C:\Program Files\Microsoft Visual Studio .NET\Vc7\PlatformSDK\lib
glut32.h -> C:\Program Files\Microsoft Visual Studio .NET\Vc7\PlatformSDK\Include\gl

2.2 For UNIX or UNIX-like operating systems:

Mesa 3D is recommended, which already includes GLUT in current version.

If you don't find the header files and libraries that you need to use in standard locations, you need to point the compiler and linker to their location with the appropriate -I and -L options. The libraries you link with must be specified at link time with the -l option; -lglut -lGLU -lGL -lXmu -lX11 is typical.

3. Learning OpenGL basics through comparing simple examples

Illustrate how to do the following things through the simple examples, compared to a same example in Managed DirectX.

3.0 Setup an OpenGL project in Microsoft Visual Studio .NET 2003

3.1 Create and manage drawing windows through GLUT.

- glutInit(&argc, argv);

It will initialize the GLUT library and negotiate a session with the window system.

- glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
— sets the initial display mode.

  Double buffering allows for smooth animation by keeping the drawing in a back buffer and swapping the back with the front buffer (the visible one) when the rendering is complete. Using double buffering prevents flickering.

- glutInitWindowSize(1200,700);
- glutInitWindowPosition(10, 50);
- glutCreateWindow ("A simple OpenGL example");
- glutDisplayFunc (display); // sets the display callback for the current window.
- glutReshapeFunc (reshape); //sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established.
- glutKeyboardFunc (keyboard); // sets the keyboard callback, used when a user triggers a key in the keyboard
- glutIdleFunc (NULL); //sets the global idle callback.
- glutMainLoop ( ); // enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return.

3.2 Clearing the window (Clear color buffer and depth buffer)

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear Screen And Depth Buffer

glClearDepth (1.0); — specifies the depth value used by glClear to clear the depth buffer. Values specified by glClearDepth are clamped to the range [0,1].

3.3 Specifying a Color

glColor3f(). It takes three parameters, all of which are floating-point numbers between 0.0 and 1.0. The parameters are, in order, the red, green, and blue components of the color.

glClearColor() takes four parameters, the fourth is alpha value.

3.4 Forcing Completion of Drawing (glFlush() and glFinish())

OpenGL commands are not executed immediately. Instead, they are submitted to a command buffer that is then fed into to the hardware. The glFlush() and glFinish() commands are both used to force submission of the command buffer to the hardware for execution.

glFlush() causes all OpenGL commands currently queued to be submitted to the hardware for execution. This function returns immediately after having transferred the pending OpenGL command queue to the hardware (or software) renderer. These commands are queued for execution in some finite amount of time, but glFlush() does not block waiting for command completion.

glFinish() has the same effect as glFlush(), with the addition that glFinish() will block until all commands submitted have been executed. The entire command stream sent to the hardware (or software) renderer is guaranteed to finish execution before glFinish() will return.

3.5 Hidden-Surface Removal(Using a depth buffer)

OpenGL:

```
glEnable(GL_DEPTH_TEST);
glClear(GL_DEPTH_BUFFER_BIT);
```

DirectX:

```
presentParams.EnableAutoDepthStencil = true;
presentParams.AutoDepthStencilFormat = DepthFormat.D16;
device.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.CornflowerBlue, 1.0f, 0);
```

In both cases, there are more precision at the front of the depth buffer. This will cause the problem of less precision in large scene.

3.6 Describing Points, Lines, and Polygons

3.6.1 Using glBegin() and glEnd()

They are different from device.BeginScene() and device.EndScene() in DirectX.

glBegin/glEnd are used to specify the vertices of a primitive or group of like primitives, to group glVertex instructions together.,

Only a subset of GL subroutines can be used between the glBegin and glEnd

subroutines. For examples, glVertex, glColor, glNormal, glTexCoord, glMaterial and so on.

This is not necessary in Direct3D because you cannot specify a single vertex anyway. Note that specifying single vertices indeed is a bad idea to when it comes to generate large 3D meshes: You are going to issue tons of calls to the API and you cannot store the vertices on the graphics card (or AGP(Accelerated Graphics Port) memory, that is).

For this reason, OpenGL also offers vertex arrays (which are not as efficient as Direct3D's vertex buffers). Only since last year OpenGL also offers Buffer Objects, which parallel Direct3D's vertex buffers in usage and efficency.

(The ARB_vertex_buffer_object extension was approved by the Architecture Review Board (ARB) on February 12th, 2003, This extension moves vertex array data onto the graphics card's memory, which allows for quick access.)

In professional OpenGL programming, glBegin/glEnd hardly show up.

3.6.2 Specifying Vertices

glVertex3f(), glVertex3d(), glVertex2f(), glColor3f.....

b: 8-bit integer , s: 16-bit integer , f :: 32-bit floating-point, d: 64-bit floating-point

3.6.3 Drawing Primitives

OpenGL:

glBegin(GLenum mode);

GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP,

GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP, and GL_POLYGON.

DirectX:

public void DrawPrimitives(PrimitiveType primitiveType, int startVertex, int primitiveCount);

PrimitiveType:

TriangleFan, TriangleStrip, TriangleList, LineStrip, LineList, PointList

3.6.4 Displaying Points, Lines, and Polygons

glPointSize(GLfloat size);, glBegin(GLenum mode); glLineWidth(GLfloat width); and so on.

3.6.4 Culling Polygon Faces

OpenGL:

glEnable(GL_CULL_FACE); .

glFrontFace(GLenum mode); Define the orientation of front-facing polygons

GL_CW and GL_CCW are accepted.

The default value is GL_CCW, which means polygons whose vertices appear in

counterclockwise order on the screen are called front-facing.

DirectX

This is the same here. The counterclockwise cull mode is the default for Direct3D

device.RenderState.CullMode = Cull.None;(Turn off culling)
`Cull.None, Cull.CounterClockwise, Cull.Clockwise`

3.7 Viewing and Transformations

Direct3D uses a left-handed coordinate system, OpenGL a right-handed coordinate system. The direction of the z axis is the opposite.

OpenGL has only two matrices: MODELVIEW and PROJECTION. In OpenGL viewing and modeling transformations are inextricably related in OpenGL and are in fact combined into a single modelview matrix.

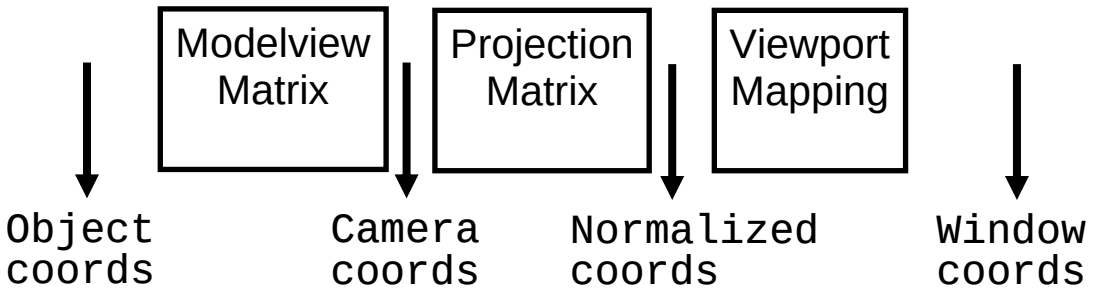
glMatrixMode(GL_MODELVIEW)
glMatrixMode(GL_PROJECTION);

Defining the Viewport

void **glViewport**(GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*);
You use the **glViewport()** command to choose a smaller drawing region; you can subdivide the window to create a split-screen effect for multiple views in the same window.

Transformation Pipeline

Stages of vertex transformations:

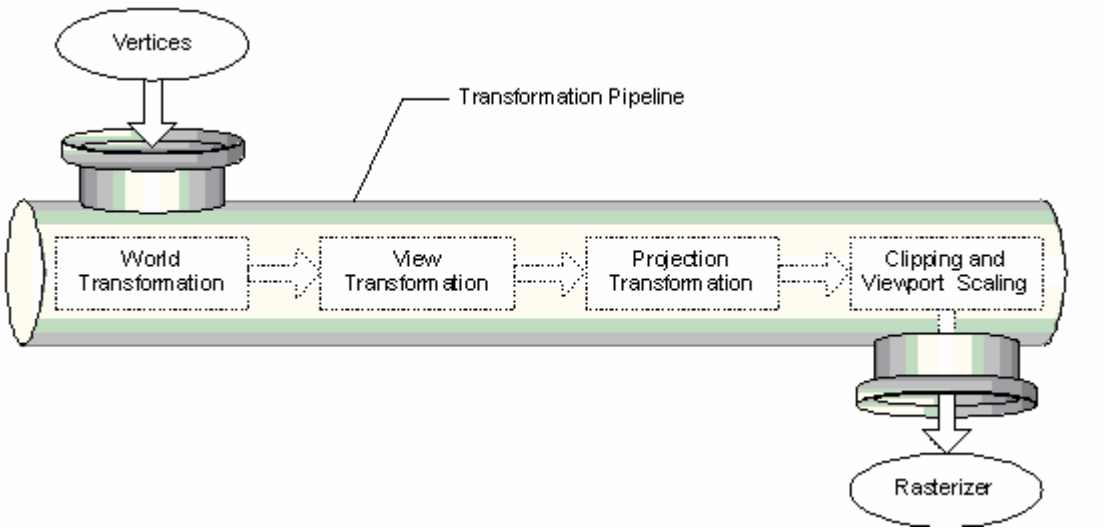


DirectX has three matrices: World, View, Perspective;

device.Transform.World =

device.Transform.Projection =

device.Transform.View =



Picture-1 Rendering pipeline for DirectX.

In OpenGL

3.7.1. Projection Transformations

Perspective Projection

gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar) –

device.Transform.Projection = Matrix.PerspectiveFovLH()

glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

Orthographic Projection-- parallel viewing volume

glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

3.7.2. Viewing Transformations

gluLookAt (eyeX , eyeY , eyeZ , centerX, centerY, centerZ, upX, upY, upZ)- view information ----

device.Transform.View = Matrix.LookAtLH()

3.7.3. Modeling transformations

glTranslate*(), glRotate*(), and glScale*(). ---

device.Transform.World = Matrix.RotationZ

3.7.4. Manipulating the Matrix Stacks

glPushMatrix()-- It copies the current matrix and adds the copy to the top of the stack.

glPopMatrix(), which discards the top matrix on the stack and uses this matrix as

the current matrix.

glLoadIdentity() command to clear the currently modifiable matrix for future transformation commands. Typically, you always call this command before specifying projection or viewing transformations And always remember to call glLoadIdentity before setting a matrix. Otherwise you are going to multiply its current value. (Which once seemed a good idea to do hierarchical things; compare glPushMatrix)

Most of the transformation commands multiply the current matrix by the specified matrix and then set the result to be the current matrix. If you don't clear the current matrix by loading it with the identity matrix, you continue to combine previous transformation matrices with the new one you supply.

An example in detail will be given here compared to DirectX.

3.8 Lighting

Create a light source and define the material characteristics of the object your are drawing.

Three main types of light source in OpenGL, directional, point and spot lights.

To directional lights, the 4th values of the lightPosition array to 0.0f.

GLfloat lightPosition[] = {0.7f, 0.7f, 0.7f, 0.0f} ;

To point lights, the 4th values of the lightPosition array to 1.0f.

GLfloat lightPosition[] = {0.7f, 0.7f, 0.7f, 1.0f} ;

light model:Ambient, Diffuse, and Specular Light

Also need to enable lighting and disable when finishing.

glEnable (GL_LIGHTING) ;

glDisable (GL_LIGHTING) ;

3.9 Blending

Alpha values can be specified with glColor*(), the 4th value.

glBlendFunc(GLenum sfactor, GLenum dfactor): supply two constants:

One specifies how the source factor should be computed, another one indicates how the destination factor should be computed.

Also, to have blending take effect, you need to enable it:

glEnable(GL_BLEND);

GL_ZERO source or destination (0, 0, 0, 0)

GL_ONE source or destination (1, 1, 1, 1)

GL_DST_COLOR, GL_SRC_COLOR ,GL_ONE_MINUS_DST_COLOR ,GL_ONE_MINUS_SRC_COLOR ,GL_SRC_ALPHA…….

In DirectX, for example

```
device.RenderState.AlphaBlendEnable = true;
device.RenderState.DestinationBlend = Blend.One;
device.RenderState.SourceBlend =
Blend.SourceAlpha;device.RenderState.BlendOperation=BlendOperation.Max;
```

4.0 Texture mapping

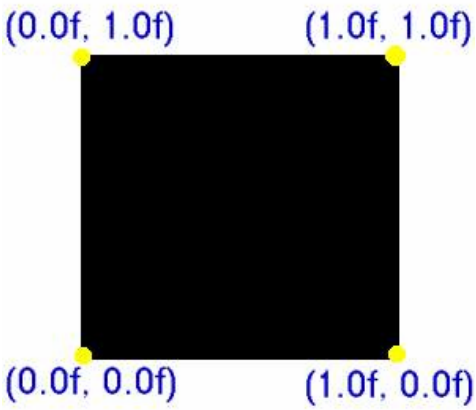
Steps to apply a texture:

- Create a texture object and specify a texture for that object

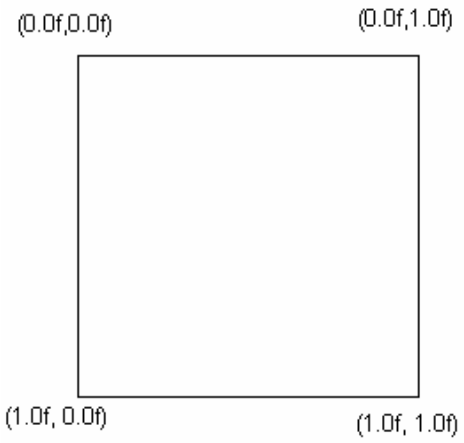
  Loading a texture from file is not a built-in feature of OpenGL. In the example used in this presentation, glaux.lib is used to load an external image(which might not be an good solution, there are many other libraries for this problem)

  1. Generate texture names
  2. Bind texture objects to texture data & parameters

     void glBindTexture(GLenum target, GLuint textureName);
- Indicate how the texture is to be applied to each pixel
- Enable texture mapping) glEnable(GL_TEXTURE_2D)
- Draw the scene, supplying both texture and geometric coordinates



Texture Coordinate System in Open



Texture Coordinate System in DirectX

An example will be given.

Good references for OpenGL online:

1. OpenGL.org

   http://www.opengl.org/

2. NeHe tutorials

http://nehe.gamedev.net/

3. Nate Robins - OpenGL - Tutors

http://www.xmission.com/~nate/tutors.html

4. Siggraph 1997 OpenGL Course Examples

http://www.opengl.org/resources/tutorials/advanced/advanced97/programs/programs.html

5. OpenGL Project Examples

http://www.sulaco.co.za/opengl.htm

6. RedBook (OpenGL Programming Guide)

http://fly.cc.fer.hr/~unreal/theredbook/

7. Examples from RedBook

http://www.opengl.org/resources/code/basics/redbook/index.html

8. GameDev.Net:
9. Gamasutra.com
10. Setting up Microsoft Visual Studio .NET for OpenGL/GLUT

http://www.students.uwosh.edu/~bradfj23/visualstudio/

http://www.cosc.brocku.ca/Offerings/3P98/course/MSVC/msvcnetglut.html

Setting up Microsoft Visual C++ 6.0 for OpenGL

http://www.airport1.de/opengl.htm