

Chapter 9 Introduction to Open GL

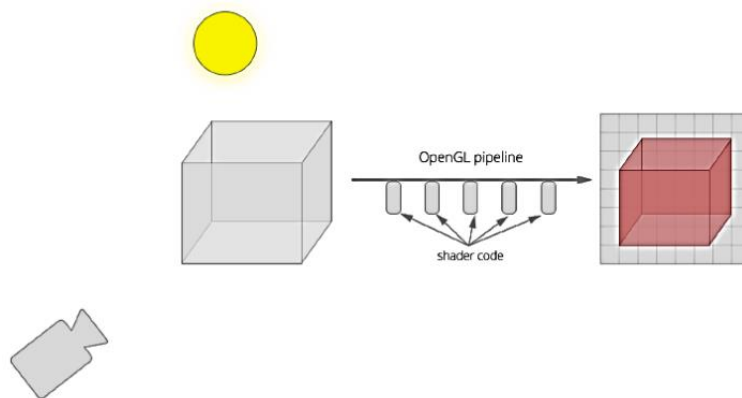
- **OpenGL (Open Graphics Library)**
 - Standard specification defining a cross language, cross platform API
 - For writing applications that produce 2D and 3D computer graphics.
- Provides a common set of commands that can be used
 - to manage graphics in different applications and on multiple platforms.
- Device independence
- Platform independence
- Abstractions (GL, GLU, GLUT)
- Open source
- Hardware-independent software interface
- Support of client-server protocol
- 3D Transformations - Rotations, scaling, translation
- Color models - Values: R, G, B, alpha
- Lighting
- Rendering
- Modeling

Basic OpenGL Syntax:

- Function names in the OpenGL basic library (also called the OpenGL core library) are prefixed with **gl**.
- Each component word within a function name has its **first letter capitalized**.
- E.g., glBegin, glClear, glCopyPixels, glPolygonMode

Chapter 9 Introduction to Open GL

- All constants begin with the uppercase letters GL_,
 - E.g., glBegin(GL_POLYGON);
 - In above example component words within a constant name are written in capital letters, and the underscore ('_') is used as a separator between all component words in the name.
- With OpenGL, an application can create the same effects in any operating system using any OpenGL-adhering graphics adapter.
- OpenGL specifies a set of "commands" or immediately executed functions.
- Each command directs a drawing action or causes special effects. A list of these commands can be created for repetitive effects.
- OpenGL is independent of the windowing characteristics of each operating system, but provides special "glue" routines for each operating system that enable OpenGL to work in that system's windowing environment.
- OpenGL comes with a large number of built-in capabilities request able through the API.
- These include hidden surface removal, alpha blending (transparency), antialiasing , texture mapping, pixel operations, viewing and modeling transformations, and atmospheric effects (fog, smoke, and haze).



Chapter 9 Introduction to Open GL

Header Files:

- For all OpenGL applications, we have to include the **gl.h** header file in every file.
- Almost all OpenGL applications use GLU, the OpenGL Utility Library, which requires inclusion of the **glu.h** header file. So almost every OpenGL source file begins with,

```
#include <GL/gl.h>
```

```
#include <GL/glu.h>
```

- If you are using GLUT for managing your window manager tasks, you should include

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

- Note that **glut.h** includes **gl.h**, **glu.h**, and **glx.h**, so including all 3 files is not required.

Callback Functions:

- A callback function is basically a function pointer that you can set that GLFW can call at an appropriate time. One of those callback functions that we can set is the KeyCallback function, which should be called whenever the user interacts with the keyboard.
- The prototype of this function is as follows:

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mode);
```

The key input function takes a GLFWwindow as its first argument, an integer that specifies the key pressed, an action that specifies if the key is pressed or released and an integer representing some bit flags to tell you if shift, control, alt or super keys have been pressed. Whenever a user pressed a key, GLFW calls this function and fills in the proper arguments for you to process.

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    // When a user presses the escape key, we set the WindowShouldClose
    // property to true,
    // closing the application
    if(key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, GL_TRUE);
}
```

Chapter 9 Introduction to Open GL

In our (newly created) `key_callback` function we check if the key pressed equals the escape key and if it was pressed (not released) we close GLFW by setting its `WindowShouldClose` property to true using `glfwSetWindowShouldClose`. The next condition check of the main while loop will then fail and the application closes.

- A callback function is a function which the library (GLUT) calls when it needs to know how to process something.
- E.g. when glut gets a key down event it uses the `glutKeyboardFunc` callback routine to find out what to do with a key press.
- GLUT supports a number of callbacks to respond to events. There are three types of callbacks: window, menu, and global.
 - Window callbacks indicate when to redisplay or reshape a window, when the visibility of the window changes, and when input is available for the window.
 - The menu callback is set by the `glutCreateMenu` call described already.
 - The global callbacks manage the passing of time and menu usage. The calling order of callbacks between different windows is undefined.
- Callbacks for input events should be delivered to the window the event occurs in. Events should not propagate to parent windows.

Chapter 9 Introduction to Open GL

Color commands:

- There are many ways to specify a color in computer graphics, but one of the simplest and most widely used methods of describing a color is the RGB color model. RGB stands for the colors red, green and blue: the additive primary colors. Each of these colors is given a value, in OpenGL usually a value between 0 and 1. 1 means as much of that color as possible, and 0 means none of that color. We can mix these three colors together to give us a complete range of colors, as shown to on the left.
- For instance, pure red is represented as (1, 0, 0) and full blue is (0, 0, 1). White is the combination of all three, denoted (1, 1, 1), while black is the absence of all three, (0, 0, 0). Yellow is the combination of red and green, as in (1, 1, 0). Orange is yellow with slightly less green, represented as (1, 0.5, 0).

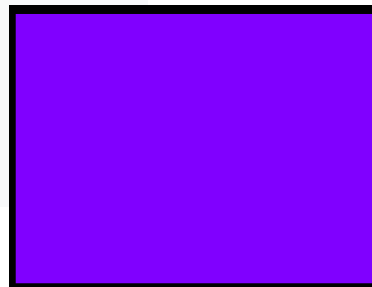
Using glColor3f:

glColor3f() takes 3 arguments: the red, green and blue components of the color you want. After you use glColor3f, everything you draw will be in that color.

For example

```
glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);
glColor3f(0.5f, 0.0f, 1.0f); // (0.5, 0, 1) is
half red and full blue, giving dark purple.
```

```
glBegin(GL_QUADS);
    glVertex2f(-0.75, 0.75);
    glVertex2f(-0.75, -0.75);
    glVertex2f(0.75, -0.75);
    glVertex2f(0.75, 0.75);
glEnd();
```

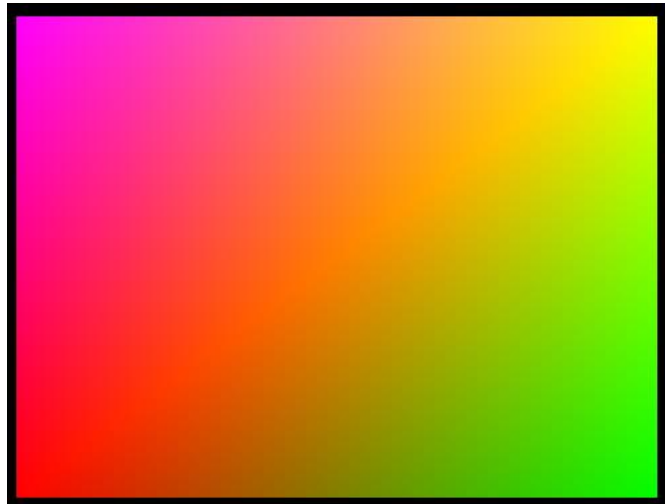


Giving Individual Vertices Different Colors

glColor3f can be called in between glBegin and glEnd. When it is used this way, it can be used to give each vertex its own color. The resulting rectangle is then shaded with an attractive color gradient, as shown on the right.

Chapter 9 Introduction to Open GL


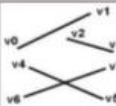
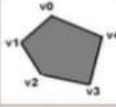


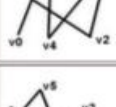
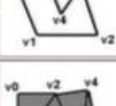
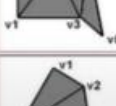
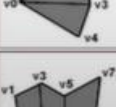

```
glClear(GL_COLOR_BUFFER_BIT |  
GL_DEPTH_BUFFER_BIT);  
    glBegin(GL_QUADS);  
        glColor3f(1.0f, 0.0f, 1.0f); // make this vertex purple  
        glVertex2f(-0.75, 0.75);  
        glColor3f(1.0f, 0.0f, 0.0f); // make this vertex red  
        glVertex2f(-0.75, -0.75);  
        glColor3f(0.0f, 1.0f, 0.0f); // make this vertex green  
        glVertex2f(0.75, -0.75);  
        glColor3f(1.0f, 1.0f, 0.0f); // make this vertex yellow  
        glVertex2f(0.75, 0.75);  
    glEnd();
```



Chapter 9 Introduction to Open GL

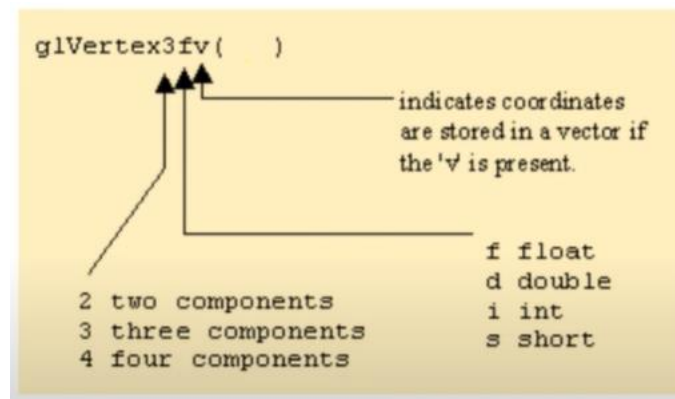
Drawings pixels:

- OpenGL provides only the lowest level of support for drawing strings of characters and manipulating fonts.
- The commands **glRasterPos*()** and **glBitmap()** position and draw a single bitmap on the screen.
- In addition, through the display-list mechanism, you can use a sequence of character codes to index into a corresponding series of bitmaps representing those characters.
- You'll have to write your own routines to provide any other support you need for manipulating bitmaps, fonts, and strings of characters.

GL_POINTS	Points	individual points	
GL_LINES	Lines	pairs of vertices interpreted as individual line segments	
GL_POLYGON	Polygon	boundary of a simple, convex polygon	
GL_TRIANGLES	Triangles	triples of vertices interpreted as triangles	
GL_QUADS	Quads	quadruples of vertices interpreted as four-sided polygons	
GL_LINE_STRIP	Line Strip	series of connected line segments	
GL_LINE_LOOP	Line Loop	same as above, with a segment added between last and first vertices	
GL_TRIANGLE_STRIP	Triangle Strip	linked strip of triangles	
GL_TRIANGLE_FAN	Triangle Fan	linked fan of triangles	
GL_QUADS_STRIP	Quad Strip	linked strip of quadrilaterals	

Chapter 9 Introduction to Open GL

Various forms of glVertex function calls



Drawing lines:

```
glBegin(GL_LINES);  
    glVertex2f(.25,0.25);  
    glVertex2f(.75,.75);  
glEnd();
```

/ Draws two horizontal lines */*

```
glBegin(GL_LINES);  
    glVertex2f(0.5f, 0.5f);  
    glVertex2f(-0.5f, 0.5f);  
    glVertex2f(-0.5f, -0.5f);  
    glVertex2f(0.5f, -0.5f);  
glEnd();
```



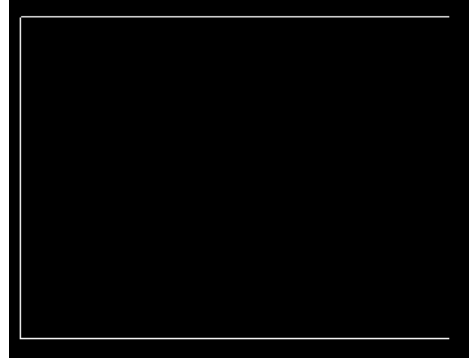
/ Draws a square */*

```
glBegin(GL_LINE_LOOP);  
    glVertex2f(0.5f, 0.5f);  
    glVertex2f(-0.5f, 0.5f);  
    glVertex2f(-0.5f, -0.5f);  
    glVertex2f(0.5f, -0.5f);  
glEnd();
```



Chapter 9 Introduction to Open GL

```
/* Draws a 'C' */
glBegin(GL_LINE_STRIP);
    glVertex2f(0.5f, 0.5f);
    glVertex2f(-0.5f, 0.5f);
    glVertex2f(-0.5f, -0.5f);
    glVertex2f(0.5f, -0.5f);
glEnd();
```



Polygons using OpenGL:

The graphics pipeline covers all of the steps that follow each other up on processing the input data to get to the final output image.

- Drawing a Triangle

```
glBegin (GL_TRIANGLES); //Start to draw a triangle
    glVertex2f( 2, 1);
    glVertex2f(1, 1);
    glVertex2f(0, 2);
glEnd() //End of drawing a triangle
```

Vertices

2- indicates parameter count
f- indicates float data type

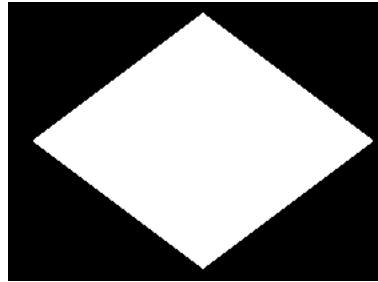
glBegin(GLenum mode);
- mode is the type of primitive we are drawing

It all begins with the vertices, these are the points from which shapes like triangles will later be constructed. Each of these points is stored with certain attributes and it's up to you to decide what kind of attributes you want to store. Commonly used attributes are 3D position in the world and texture coordinates

The vertex shader is a small program running on your graphics card that processes every one of these input vertices individually. This is where the perspective transformation takes place, which projects vertices with a 3D world position onto your 2D screen! It also passes important attributes like color and texture coordinates further down the pipeline.

Chapter 9 Introduction to Open GL

```
glBegin(GL_POLYGON);  
    glVertex2f(0.90 , 0.50);  
    glVertex2f(0.50 , 0.90);  
    glVertex2f(0.10 , 0.50);  
    glVertex2f(0.50 , 0.10);  
glEnd();
```



Viewing and Lighting:

As far as OpenGL is concerned, there is no camera. More specifically, the camera is always located at the eye space coordinate (0.0, 0.0, 0.0). To give the appearance of moving the camera, your OpenGL application must move the scene with the inverse of the camera transformation by placing it on the MODELVIEW matrix. This is commonly referred to as the viewing transformation.