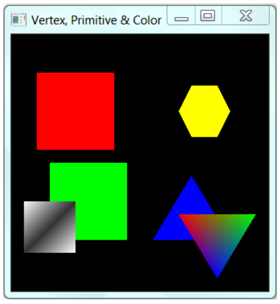
Lab 2.1: Drawing 2D Colored Shapes in OpenGL

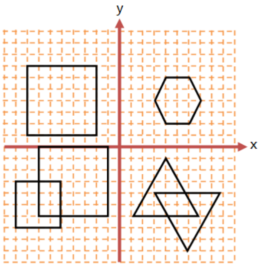
```
/*
 * GL02Primitive.cpp: Vertex, Primitive and Color
 * Draw Simple 2D colored Shapes: quad, triangle and polygon.
 */
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h

/* Initialize OpenGL
Graphics */void
initGL() {
   // Set "clearing" or background color
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque
}

/* Handler for window-repaint event. Call back when the window first
   appears andwhenever the window needs to be re-painted. */
void display() {
   glClear(GL_COLOR_BUFFER_BIT);     // Clear the color buffer with current
   clearing color

   // Define shapes enclosed within a pair of glBegin and glEnd
   glBegin(GL_QUADS);                // Each set of 4 vertices
      form a quadglColor3f(1.0f, 0.0f, 0.0f); // Red
      glVertex2f(-0.8f, 0.1f);       // Define vertices in counter-clockwise
      (CCW) orderglVertex2f(-0.2f, 0.1f); //  so that the normal (front-face)
      is facing you
      glVertex2f(-0.2f,  0.7f);
      glVertex2f(-0.8f,  0.7f); glColor3f(0.0f, 1.0f, 0.0f); // Green
      glVertex2f(-0.7f, -0.6f);
      glVertex2f(-0.1f,  -0.6f);
      glVertex2f(-0.1f,  0.0f);
      glVertex2f(-0.7f,  0.0f);

     glColor3f(0.2f,0.2f,0.2f);
     //Dark Gray
      glVertex2f(-0.9f, -0.7f);
      glColor3f(1.0f,1.0f, 1.0f);
      // White
      glVertex2f(-0.5f,-0.7f);
      glColor3f(0.2f, 0.2f, 0.2f); //
      Dark Gray
      glVertex2f(-0.5f, -0.3f);
      glColor3f(1.0f,1.0f, 1.0f);
      // White
      glVertex2f(-0.9f, -0.3f);
   glEnd();

    glBegin(GL_TRIANGLES);  // Each set of 3 vertices form a triangle
```

```
         glColor3f(0.0f, 0.0f, 1.0f); // Blue

         glVertex2f(0.1f,   -0.6f);

         glVertex2f(0.7f,   -0.6f);
         glVertex2f(0.4f,   -0.1f);

         glColor3f(1.0f, 0.0f, 0.0f); // Red
         glVertex2f(0.3f, -0.4f);
         glColor3f(0.0f, 1.0f, 0.0f); // Green
         glVertex2f(0.9f, -0.4f);
         glColor3f(0.0f, 0.0f, 1.0f); // Blue
         glVertex2f(0.6f, -0.9f);
      glEnd();

      glBegin(GL_POLYGON); // These vertices form a closed polygon
         glColor3f(1.0f, 1.0f, 0.0f); // Yellow
         glVertex2f(0.4f,   0.2f); glVertex2f(0.6f,0.2f); glVertex2f(0.7f,0.4f); glVertex2f(0.6f,0.6f);
         glVertex2f(0.4f,0.6f); glVertex2f(0.3f,   0.4f);
      glEnd();

      glFlush();   // Render now
   }
   /* Main function: GLUT runs as a console application starting at main()   */
   int main(int argc, char** argv) {
      glutInit(&argc, argv);                      // Initialize GLUT
      glutCreateWindow("Vertex, Primitive & Color");   // Create window with the given title
      glutInitWindowSize(320, 320);               // Set the window's initial width & height
      glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
      glutDisplayFunc(display);                   // Register callback handler for window re-paint event
      initGL();                                   // Our own OpenGL initialization
      glutMainLoop();                             // Enter the event-processing loop
      return (0);
}
```

Screen Shot of: 2.1_Drawing_2D_Colored_Shapes

The expected output and the coordinates are as follows. Take note that 4 shapes have pure color, and 2 shapes havecolor blending from their vertices.





OpenGL as a State Machine

OpenGL operates as a state machine, and maintain a set of state variables (such as the foreground color, background color, and many more). In a state machine, once the value of a state variable is set, the value persists until a new value is given.

For example, we set the "clearing" (background) color to black once in initGL(). We use this setting to clear the window in the display() repeatedly (display() is called back whenever there is a window re-paint request) - the clearing color is not changed in the entire program.

```
// In initGL(), set the "clearing" or background color
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);   // black and opaque

// In display(), clear the color buffer (i.e., set background) with the current "clearing" color glClear(GL_COLOR_BUFFER_BIT);
```

Another example: If we use glColor function to set the current foreground color to "red", then "red" will be used for all the subsequent vertices, until we use another glColor function to change the foreground color.

In a state machine, everything shall remain until you explicitly change it!

Naming Convention for OpenGL Functions

An OpenGL functions:

- begins with lowercase gl (for core OpenGL), glu (for OpenGL Utility) or glut (for OpenGL Utility Toolkit).
- followed by the purpose of the function, in camel case (initial-capitalized), e.g., glColor to specify the drawing color, glVertex to define the position of a vertex.
- followed by specifications for the parameters, e.g., glColor3f takes three float parameters. glVectex2i takestwo intparameters.
  (This is needed as C Language does not support function overloading. Different versions of the function need to bewritten for different parameter lists.)

The convention can be expressed as follows:

```
returnType glFunction[234][sifd] (type value, ...);   // 2, 3 or 4 parameters
returnType glFunction[234][sifd]v (type *value);   // an array parameter
```

The function may take 2, 3, or 4 parameters, in type of s (GLshort), i (GLint), f (GLfloat) or d (GLdouble). The 'v' (for vector) denotes that the parameters are kept in an array of 2, 3, or 4 elements, and pass into the function as an array pointer.

OpenGL defines its own *data types*:

- Signed Integers: GLbyte (8-bit), GLshort (16-
- bit), GLint (32-bit). Unsigned Integers:
  GLubyte (8-bit), GLushort (16-bit), GLuint (32-
  bit).
- Floating-point numbers: GLfloat (32-bit), GLdouble (64-bit), GLclampf and GLclampd (between 0.0 and 1.0).
- GLboolean (unsigned char with 0 for false and non-0 for true).
- GLsizei (32-bit non-negative integers).
- GLenum (32-bit enumerated integers).

The OpenGL types are defined via typedef in "gl.h" as follows:

```
typedef unsigned int           GLenum;
typedef unsigned char          GLboolean;
typedef unsigned int           GLbitfield;
typedef void              GLvoid;
typedef signed char       GLbyte;         /* 1-byte signed */
typedef short             GLshort;        /* 2-byte signed */
typedef int               GLint;          /* 4-byte signed */
typedef unsigned char     GLubyte;        /* 1-byte unsigned */

typedef unsigned short    GLushort;       /* 2-byte unsigned */
typedef unsigned int      GLuint;         /* 4-byte unsigned */
typedef int               GLsizei;        /* 4-byte signed */
typedef float             GLfloat;        /* single precision float */
typedef float             GLclampf;       /* single precision float in [0,1] */
typedef double            GLdouble;       /* double precision float */
typedef double            GLclampd;       /* double precision float in [0,1] */
```

OpenGL's *constants* begins with "GL_", "GLU_" or "GLUT_", in uppercase separated with underscores, e.g., GL_COLOR_BUFFER_BIT.

For examples,

```
glVertex3f(1.1f, 2.2f, 3.3f);              // 3 GLfloat parameters
glVertex2i(4, 5);                          // 2 GLint parameters
glColor4f(0.0f, 0.0f, 0.0f, 1.0f);         // 4 GLfloat parameters
GLdouble aVertex[] = {1.1, 2.2, 3.3};
glVertex3fv(aVertex);                      // an array of 3 GLfloat values
```

One-time Initialization initGL()

The initGL() is meant for carrying out one-time OpenGL initialization tasks, such as setting the clearing color.
initGL() is invoked once (and only once) in main().

Callback Handler display()

The function display() is known as a *callback event handler*. An event handler provides the *response* to a particular*event* (such as key-press, mouse-click, window-paint). The function display() is meant to be the handler for *window-paint* event. The OpenGL graphics system calls back display() in response to a window-paint request to re-paint the window (e.g., window first appears, window is restored after minimized, and window is resized). Callback means that the function is invoked by the system, instead of called by the your program.

The Display() runs when the window first appears and once per subsequent re-paint request. Observe that we included OpenGL graphics rendering code inside the display() function, so as to re-draw the entire window when the window first appears and upon each re-paint request.

Color

We use glColor function to set the *foreground color*, and glClearColor function to set the *background* (or *clearing*) color.

```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue)

void glColor3fv(GLfloat *colorRGB)
void glColor4f(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)
void glColor4fv(GLfloat*colorRGBA)

void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha) // GLclampf in the range of 0.0f to 1.0f
```

Notes:

- Color is typically specified in float in the range 0.0f and 1.0f.
- Color can be specified using RGB (Red-Green-Blue) or RGBA (Red-Green-Blue-Alpha) components. The 'A' (or alpha) specifies the transparency (or opacity) index, with value of 1 denotes opaque (non-transparent and cannot see-thru) and value of 0 denotes total transparent. We shall discuss alpha later.

In the above example, we set the background color via glClearColor in initGL(), with R=0, G=0, B=0 (black) and A=1 (opaque and cannot see through).

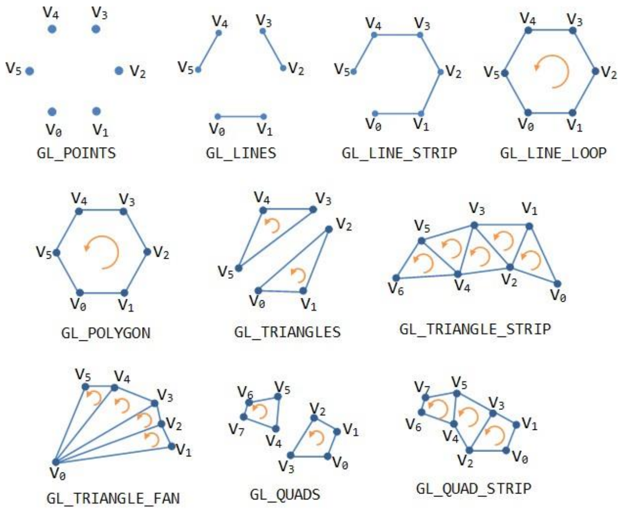
```
// In initGL(), set the "clearing" or background color
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);   // Black and opaque
```

In display(), we set the vertex color via glColor3f for subsequent vertices. For example, R=1, G=0, B=0 (red).

```
// In display(), set the foreground color of the pixel

glColor3f(1.0f, 0.0f, 0.0f); // Red
```

Geometric Primitives

In OpenGL, an object is made up of geometric primitives such as triangle, quad, line segment and point. A primitive is made up of one or more vertices. OpenGL supports the following primitives:



OpenGL Primitives

A geometric primitive is defined by specifying its vertices via glVertex function, enclosed within a pair glBegin and glEnd.

glBegin specifies the type of geometric object, such as GL_POINTS, GL_LINES, GL_QUADS, GL_TRIANGLES, and GL_POLYGON. For types that end with 'S', you can define multiple objects of the same type in each glBegin/glEnd pair.For example, for GL_TRIANGLES, each set of three glVertex's defines a triangle.

The vertices are usually specified in float precision. It is because integer is not suitable for trigonometric operations (needed to carry out transformations such as rotation). Precision of

```
void glBegin(GLenum shape)
    void glVertex[234][sifd] (type x, type y, type z, ...)
    void glVertex[234][sifd]v (type *coords)
void glEnd()
```

float is sufficient for carrying out intermediate operations, and render the objects finally into pixels on screen (with resolution of says 800x600, integral precision). double precision is often not necessary.

In the above example:

```
glBegin(GL_QUADS);
     .... 4 quads with 12x glVertex() ....
glEnd();
```

we define 3 color quads (GL_QUADS) with 12x glVertex() functions.

```
glColor3f(1.0f, 0.0f, 0.0f);
glVertex2f(-0.8f, 0.1f);
glVertex2f(-0.2f, 0.1f);
glVertex2f(-0.2f, 0.7f);
glVertex2f(-0.8f, 0.7f);
```

We set the color to red (R=1, G=0, B=0). All subsequent vertices will have the color of red. Take note that in OpenGL, color (and many properties) is applied to vertices rather than primitive shapes. The color of the a primitive shape is *interpolated* from its vertices.

We similarly define a second quad in green.

For the third quad (as follows), the vertices have different color. The color of the quad surface is interpolated from its vertices, resulting in a shades of white to dark gray, as

```
glColor3f(0.2f, 0.2f, 0.2f);                 // Dark Gray
glVertex2f(-0.9f, -0.7f);
glColor3f(1.0f, 1.0f, 1.0f);                // White
glVertex2f(-0.5f, -0.7f);
glColor3f(0.2f, 0.2f, 0.2f);                // Dark Gray
glVertex2f(-0.5f, -0.3f);
glColor3f(1.0f, 1.0f, 1.0f);                // White
glVertex2f(-0.9f, -0.3f);
```

shown in the output.

Lab 2.2: Viewport Resize & Clipping

glMatrixMode (GL_PROJECTION) and glLoadIdentity ( ) functions are used to set up the projection identity matrix used by the routine gluOrtho2D (0.0, (GLdouble) width, 0.0, (GLdouble) height) to project the object coordinates onto the screen from (left, bottom) position to the (right, top) position of the screen.

`glViewport` (GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*) ;

x, y: Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0,0).

width, height: Specify the width and height of the viewport.

```
/*
*          Viewport.cpp: Clipping-area and Viewport
*          Implementing resize to ensure same aspect ratio between the
*           clipping-area and the viewport.
*/
#include <windows.h> // for MS Windows
#include <GL/glut.h> // GLUT, include glu.h and gl.h
/* Initialize OpenGL Graphics */
void initGL() {
// Set "clearing" or background color
            glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque 13          }

void display() {
            glClear(GL_COLOR_BUFFER_BIT);// Clear the color buffer with current clearing color
// Define shapes enclosed within a pair of glBegin and glEnd
  glBegin(GL_QUADS);      // Each set of 4 vertices form a quad                         glColor3f(1.0f, 0.0f,
0.0f); // Red
            glVertex2f(-0.8f, 0.1f);        // Define vertices in counter-clockwise (CCW) order
            glVertex2f(-0.2f, 0.1f);        // so that the normal (front-face) is facing you            glVertex2f(-
0.2f, 0.7f);
            glVertex2f(-0.8f, 0.7f);


            glColor3f(0.0f, 1.0f, 0.0f);    //          Green
            glVertex2f(-0.7f, -0.6f);
            glVertex2f(-0.1f, -0.6f);
            glVertex2f(-0.1f, 0.0f);
            glVertex2f(-0.7f, 0.0f);

            glColor3f(0.2f, 0.2f, 0.2f);    //          Dark Gray
            glVertex2f(-0.9f, -0.7f);
            glColor3f(1.0f, 1.0f, 1.0f);    //          White
            glVertex2f(-0.5f, -0.7f);
            glColor3f(0.2f, 0.2f, 0.2f);    //          Dark Gray
            glVertex2f(-0.5f, -0.3f);
            glColor3f(1.0f, 1.0f, 1.0f);    //          White
            glVertex2f(-0.9f, -0.3f);
  glEnd();

  glBegin(GL_TRIANGLES);  //          Each set of 3 vertices form a triangle
            glColor3f(0.0f, 0.0f, 1.0f);    //          Blue
            glVertex2f(0.1f, -0.6f);
            glVertex2f(0.7f, -0.6f);
            glVertex2f(0.4f, -0.1f);

            glColor3f(1.0f, 0.0f, 0.0f);    //          Red
            glVertex2f(0.3f, -0.4f);
            glColor3f(0.0f, 1.0f, 0.0f);    //          Green
            glVertex2f(0.9f, -0.4f);
            glColor3f(0.0f, 0.0f, 1.0f);    //          Blue
            glVertex2f(0.6f, -0.9f);
 glEnd();

  glBegin(GL_POLYGON);   //           These vertices form a closed polygon
            glColor3f(1.0f, 1.0f, 0.0f);    //          Yellow
            glVertex2f(0.4f, 0.2f);
            glVertex2f(0.6f, 0.2f);
            glVertex2f(0.7f, 0.4f);
            glVertex2f(0.6f, 0.6f);
            glVertex2f(0.4f, 0.6f);
            glVertex2f(0.3f, 0.4f);
```

```
   glEnd();

            }
glFlush(); // Render now

/* Handler for window re-size event. Called back when the window first appears and
            whenever the window is re-sized with its new width and height */

void reshape(GLsizei width, GLsizei height) { // GLsizei for non-negative integer
// Compute aspect ratio of the new window
if (height == 0) height = 1;// To prevent divide by 0

            GLfloat aspect = (GLfloat)width / (GLfloat)height;

// Set the viewport to cover the new window
            glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport
            glMatrixMode(GL_PROJECTION); // To operate on the Projection matrix
            glLoadIdentity();   // Reset the projection matrix
            if (width >= height) {          // aspect >= 1, set the height from -1 to 1, with larger width
                        gluOrtho2D(-1.0 * aspect, 1.0 * aspect, -1.0, 1.0);
                        } else {
// aspect < 1, set the width to -1 to 1, with larger height
                        gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);
                        }
            }

            /*          Main function: GLUT runs as a console application starting at main() */
            int main(int argc, char** argv) {
            glutInit(&argc, argv);          // Initialize GLUT
            glutInitWindowSize(640, 480);          // Set the window's initial width & height - non-square
            glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
            glutCreateWindow("Viewport Transform"); // Create window with the given title
            glutDisplayFunc(display);    // Register callback handler for window re-paint event
            glutReshapeFunc(reshape); // Register callback handler for window re-size event
            initGL();  // Our own OpenGL initialization
            glutMainLoop();    // Enter the infinite event-processing loop
            return 0;
            }
```