#include <windows.h> // for MS Windows

#include <GL/glut.h> // GLUT, include glu.h and gl.h

/\* Handler for window-repaint event. Call back when the window first appears and

whenever the window needs to be re-painted. \*/

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque

glClear(GL\_COLOR\_BUFFER\_BIT); // Clear the color buffer (background)

glPointSize(5.0);

// Draw a Red 1x1 Square centered at origin

glBegin(GL\_TRIANGLES); // Each set of 4 vertices form a quad

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

glVertex2f(0.0f, 0.0f);

glVertex2f(0.3f, 0.0f);

glVertex2f(0.15f, 0.2f); // x, y

glEnd();

glBegin(GL\_POLYGON); // These vertices form a closed polygon

glColor3f(1.0f, 1.0f, 0.0f); // Yellow

glVertex2f(-0.9f, 0.0f);

glVertex2f(-0.5f, 0.0f);

glVertex2f(-0.5f, 0.3f);

glVertex2f(-0.9f, 0.3f);

glEnd();

glBegin(GL\_POLYGON); // These vertices form a closed polygon

glColor3f(1.0f, 0.0f, 0.0f); // red

glVertex2f(-0.5f, -0.5f);

glVertex2f(-0.3f, -0.3f);

glVertex2f(-0.5f, -0.1f);

glVertex2f(-0.7f, -0.3f);

glEnd();

glBegin(GL\_TRIANGLES); // Each set of 4 vertices form a quad

glColor3f(1.0f, 1.0f, 0.0f); // yellow

glVertex2f(0.3f, -0.5f);

glVertex2f(0.9f, -0.5f);

glVertex2f(0.3f, -0.2f); // x, y

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("OpenGL Setup Test"); // Create a window with the given title

glutInitWindowSize(320, 320); // Set the window's initial width & height

glutDisplayFunc(display); // Register display callback handler for window re-paint

glutMainLoop(); // Enter the event-processing loop

return 0;

}

-------------------------------------------------------------------------------------------

#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 and

whenever the window needs to be re-painted. \*/

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT); // Clear the color buffer with current clearing color

glBegin(GL\_QUADS); // Each set of 4 vertices form a quad

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

glVertex2f(0.0f, 0.0f); // x, y

glVertex2f(0.7f, 0.0f);

glVertex2f(0.7f, 0.1f); // x, y

glVertex2f(0.0f, 0.1f);

glEnd();

glBegin(GL\_QUADS); // Each set of 4 vertices form a quad

glColor3f(1.0f, 0.5f, 0.0f); // Red

glVertex2f(0.0f, 0.1f);

glVertex2f(0.7f, 0.1f);

glVertex2f(0.7f, 0.2f);

glVertex2f(0.0f, 0.2f); // x, y

glEnd();

glBegin(GL\_QUADS);

glColor3f(1.0f, 0.0f, 0.0f);

glVertex2f(0.0f, 0.2f);

glVertex2f(0.7f, 0.2f);

glVertex2f(0.7f, 0.3f);

glVertex2f(0.0f, 0.3f);

glEnd();

glBegin(GL\_QUADS);

glColor3f(1.0f, 1.0f, 0.0f);

glVertex2f(0.0f, 0.3f);

glVertex2f(0.7f, 0.3f);

glVertex2f(0.7f, 0.4f);

glVertex2f(0.0f, 0.4f);

glEnd();

glBegin(GL\_QUADS);

glColor3f(0.0f, 0.0f, 1.0f);

glVertex2f(0.0f, 0.4f);

glVertex2f(0.7f, 0.4f);

glVertex2f(0.7f, 0.5f);

glVertex2f(0.0f, 0.5f);

glEnd();

glBegin(GL\_QUADS);

glColor3f(1.0f, 0.0f, 1.0f);

glVertex2f(0.0f, 0.5f);

glVertex2f(0.7f, 0.5f);

glVertex2f(0.7f, 0.6f);

glVertex2f(0.0f, 0.6f);

glEnd();

glBegin(GL\_QUADS);

glColor3f(1.0f, 1.0f, 1.0f); // White

glVertex2f(0.0f, 0.6f);

glVertex2f(0.7f, 0.6f);

glVertex2f(0.7f, 0.7f);

glVertex2f(0.0f, 0.7f);

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(420, 420); // Set the window's initial width & height

glutDisplayFunc(display); // Register callback handler for window re-paint event

initGL(); // Our own OpenGL initialization

glutMainLoop(); // Enter the event-processing loop

return 0;

}