CPSC/ECE 4780/6780

General-Purpose Computation on Graphical Processing Units (GPGPU)

Lecture 16: (OpenGL) Introduction to OpenGL

Recap of Last Lecture

- OpenCL Kernels
  - Work-item vs. Work-group
- OpenCL Memories
  - Address space: Global, local, constant, private
  - Memory objects (Buffer, image)
- Synchronization
  - Work-item synchronization
  - Kernel synchronization
- Events

What is OpenGL?

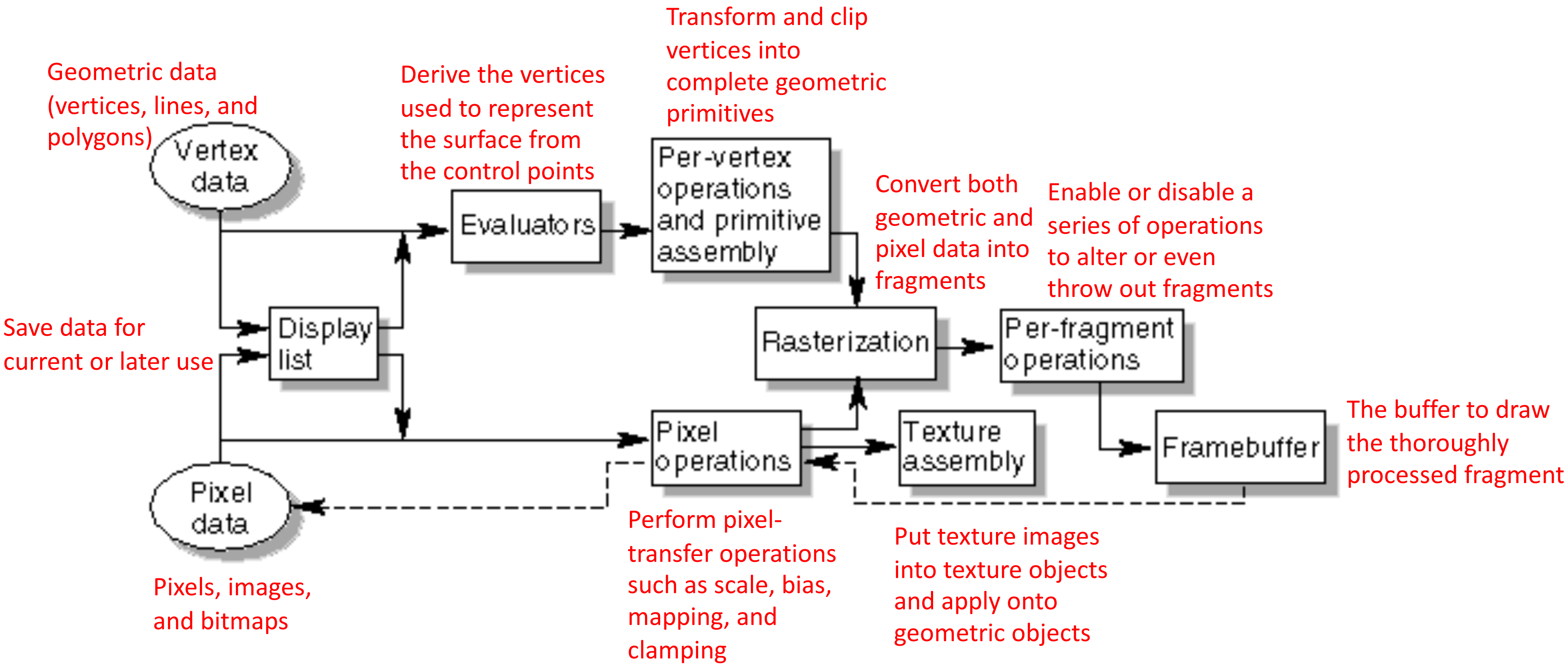
- **Open Graphics Library** (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics
- History
  - First came from an interface called GL developed for Silicon Graphics Incorporated (SGI) hardware in 1991
  - Released in January 1992
  - Currently managed by the Khronos Group and the OpenGL Architecture Review Board (ARB)

Features of OpenGL

- Contains more than 200 functions
  - Primitive functions (geometric primitives and discrete entities): define the elements that can produce images
  - Attribute functions: control the appearance of primitives
  - Viewing functions: determine the properties of the camera
  - Input functions: windows control, mouse and keyboard use
  - Control functions: start and terminate programs, turn on OpenGL features
- Concerned primarily with **rendering**
- Portable to any computer that supports the interface (Platform independent)
  - No input and windowing functions
  - Use **OpenGL Utility Toolkit (GLUT)** to interact with an operating system and the local windowing system
- It is a state machine
  - State variables: current color, current viewing, projection transformations, line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, material properties of the objects, etc.
  - The states you put into remain in effect until you change them
  - Enable or disable the mode of a state variable by **glEnable()** or **glDisable()**

Rendering is the process of taking the specification of geometric objects and their properties and forming a picture of them with a virtual camera and lights

OpenGL Rendering Pipeline



Geometric data (vertices, lines, and polygons)

Derive the vertices used to represent the surface from the control points

Transform and clip vertices into complete geometric primitives

Save data for current or later use

Convert both geometric and pixel data into fragments

Enable or disable a series of operations to alter or even throw out fragments

The buffer to draw the thoroughly processed fragment

Pixels, images, and bitmaps

Perform pixel-transfer operations such as scale, bias, mapping, and clamping

Put texture images into texture objects and apply onto geometric objects

OpenGL-Related Libraries

- The core OpenGL Library (gl or GL)
    - Contains all the required OpenGL function
    - #include <GL/gl.h>
    - Routines use the prefix gl, e.g., glVertex3f()
- The OpenGL Utility Library (glu or GLU)
    - Contains functions that are written using the functions in the core library but are helpful for users to have available
    - #include <GL/glu.h>
    - Routines use the prefix glu, e.g., gluOrtho2D()
- The OpenGL Utility Toolkit (GLUT)
    - A window system-independent toolkit to hide the complexities of differing window system APIs
    - #include <GL/glut.h>

GLUT, the OpenGL Utility Toolkit

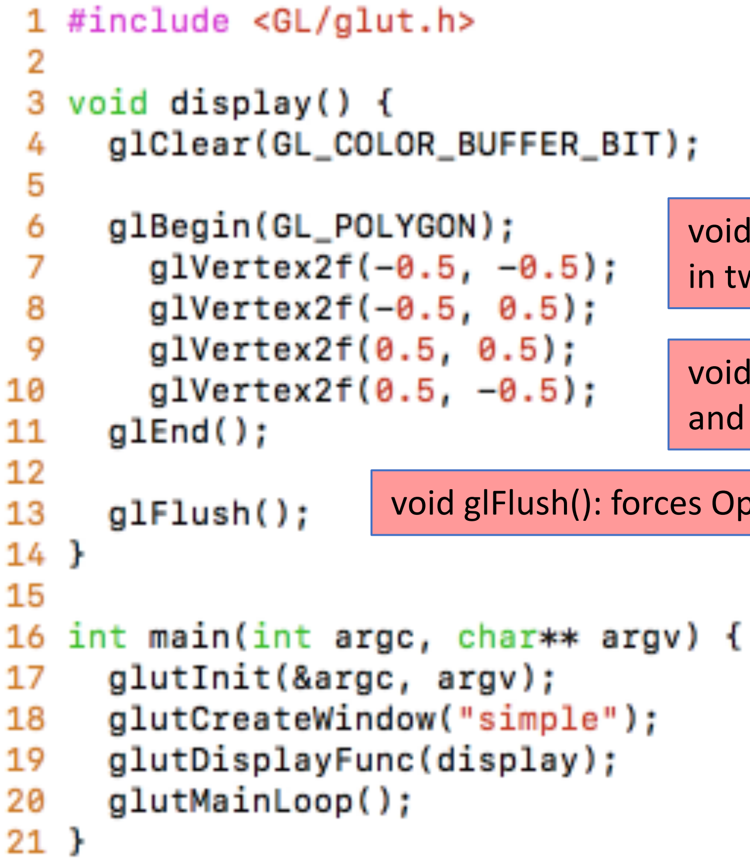
- A library of functions that are common to virtually all modern windowing systems
- Used to simplify opening windows, detecting input, and so on
- Includes several routines that create more complicated 3D objects (cone, cube, sphere, torus, teapot, icosahedron, octahedron, tetrahedron, dodecahedron) as wireframes or as solid shaded objects with surface normals defined, e.g.,
  - void **glutWireCube**(GLdouble *size*);
  - void **glutSolidCube**(GLdouble *size*);
  - void **glutWireSphere**(GLdouble *radius*, GLint *slices,* GLint *stacks*);
  - void **glutSolidSphere**(GLdouble *radius*, GLint *slices,* GLint *stacks*);

GLUT – Window Management Routines

- Five routines perform tasks necessary to initial a window:
  - **glutInit**(int **argc*, char ***argv*): initializes GLUT and processes any command line arguments
  - **glutInitDisplayMode**(unsigned int *mode*):
    - Specifies whether to use an *RGBA* or color-index color model
    - Specifies whether to use a single- or double-buffered window
    - Specifies whether the window has an associated depth, stencil, and/or accumulation buffer
    - E.g., Specify a window with double buffering, the RGBA color model, and a depth buffer by calling **glutInitDisplayMode**(*GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH*)
  - **glutInitWindowPosition**(int *x*, int *y*): specifies the screen location for the upper-left corner of your window
  - **glutInitWindowSize**(int *width*, int *size*): specifies the size, in pixels, of your window
  - int **glutCreateWindow**(char **string*): creates a window with an OpenGL context
- **glutMainLoop**(void): display the window
- **glutPostRedisplay**(void): gives glutMainLoop() a nudge to call the registered display callback at its next opportunity if your program changes the contents of the window

A Simple Program ”simple.c”

- Draw a white rectangle on a black background



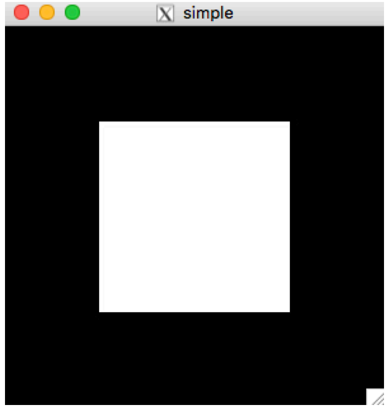
```c
#include <GL/glut.h>

void display() {
  glClear(GL_COLOR_BUFFER_BIT);

  glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -0.5);
  glEnd();

  glFlush();
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutCreateWindow("simple");
  glutDisplayFunc(display);
  glutMainLoop();
}
```

void glClear(GLbitfield mask): clears all buffers whose bits are set in mask. GL_COLOR_BUFFER refers to the color buffer

void glVertex{234}{sifd}(TYPE xcoordinate, TYPE ycoordinate, ….): specifies the location of a vertex in two, three, or four dimensions with the types short (s), int (i), float (f), or double (d)

void glBegin(Glenum mode) and void glEnd(): specifies the beginning of an object of type mode, and the end of a list of vertices

void glFlush(): forces OpenGL commands to execute

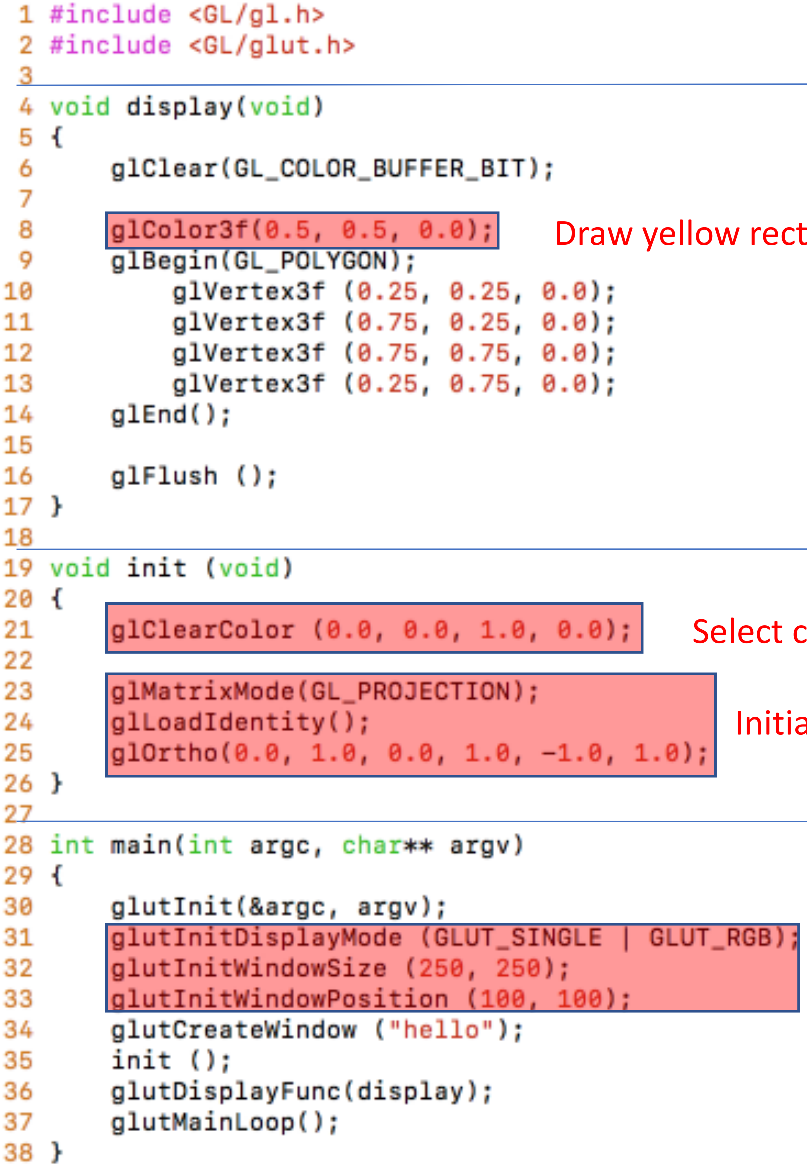


- Compiling: gcc simple.c -o simple -lGL -lGLU -lglut

Set up a More Sophisticate Program

- Changing the GLUT defaults to define a more customized window
- Setting colors for drawing objects
  - Two color models: RGBA or color-index mode
  - Default color for clearing the screen is black, or change by **glClearColor**(…)
  - Default color to fill an object is white, or change by **glColor***(…)
- Setting up coordinate systems transformations
  - Two coordinate systems: object coordinates (world coordinates) or window coordinates (screen coordinates)
  - void **glMatrixMode**(GLenum mode): specifies which matrix will be affect by subsequent transformation functions
  - void **glLoadIdentity**(): initializes the current matrix to an identity matrix
  - void **gluOrtho**(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top): specifies a two-dimensional rectangular clipping region whose lower-left corner is at (left, bottom) and whose upper-right corner is at (right, top).

Second Version of “simple.c”



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

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.5, 0.5, 0.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();

    glFlush ();
}

void init (void)
{
    glClearColor (0.0, 0.0, 1.0, 0.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Draw yellow rectangle

Set up geometry

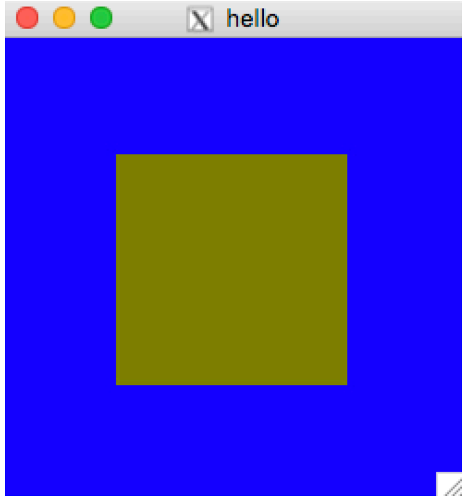
Select clearing (background) color to be blue

Initialize viewing values

Initialize states

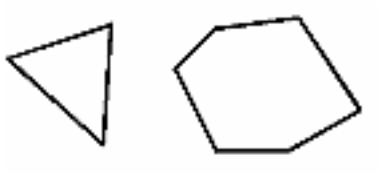
Declare initial window display mode (single buffer and RGBA), size, and position

Set up callbacks

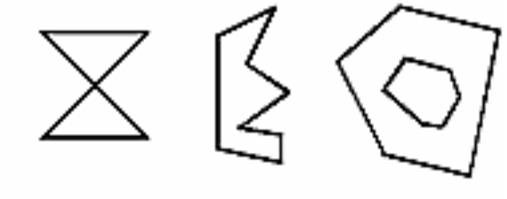


Geometric Primitives

- Three basic types: points, lines, and polygons
- Described in terms of their vertices: coordinates of points, endpoints of lines, corners of polygons
- Points: represented by a set of floating-point numbers called a vertex
- Lines: a line segment
- Polygons: areas enclosed by single closed loops of line segments
  - The edges can't intersect
  - Polygons must be convex (they cannot have indentations)



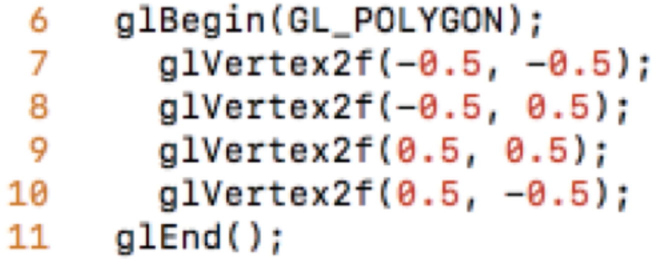
Valid



Invalid

Rectangles

- In the previous example, we drew a rectangle using:

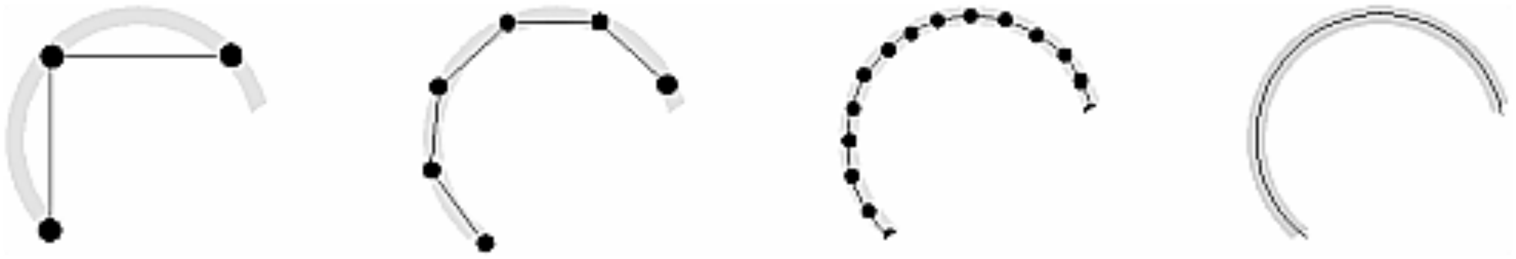


```
glBegin(GL_POLYGON);
  glVertex2f(-0.5, -0.5);
  glVertex2f(-0.5, 0.5);
  glVertex2f(0.5, 0.5);
  glVertex2f(0.5, -0.5);
glEnd();
```

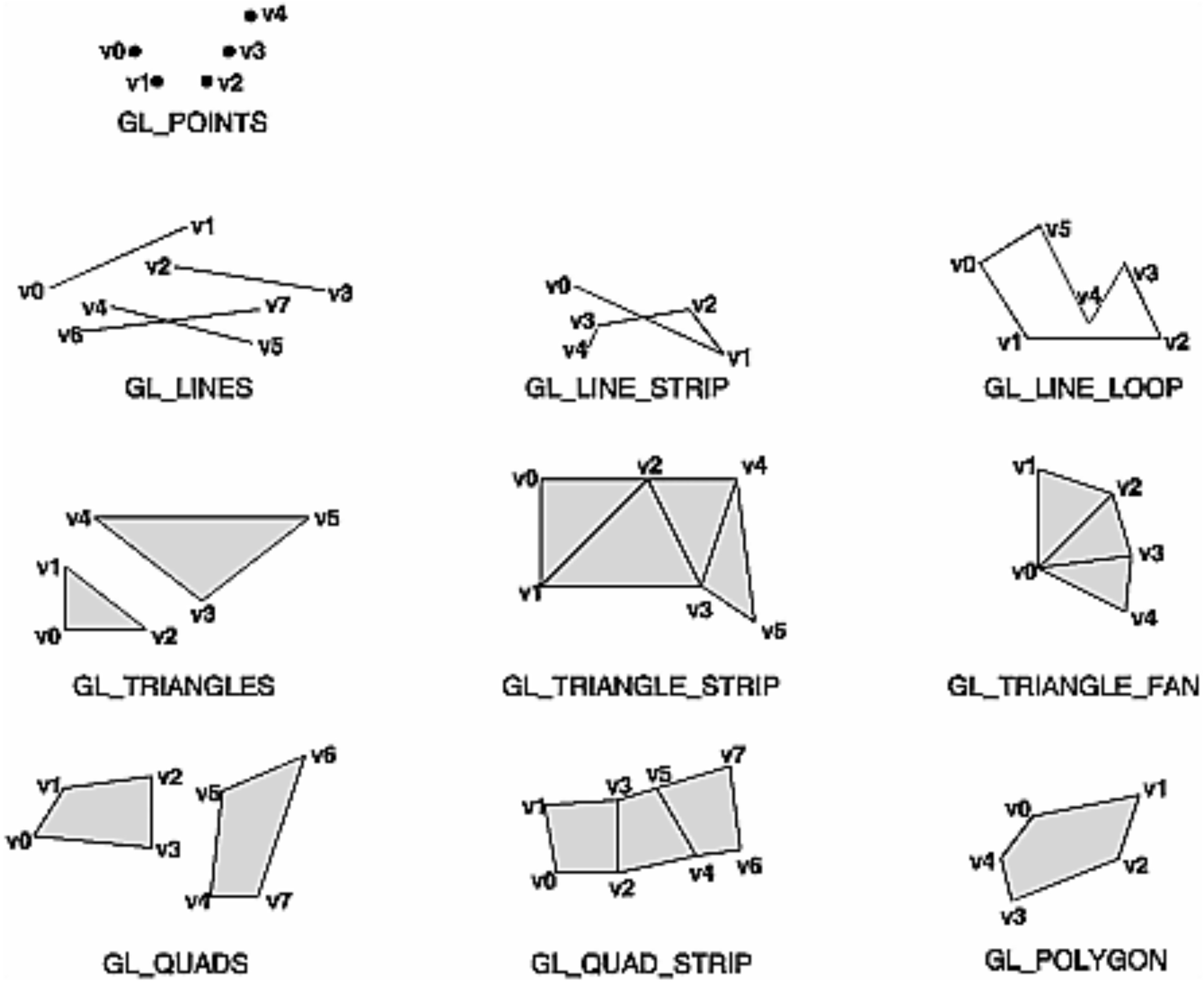
- Which can also be done by glRectf(-0.5, -0.5, 0.5, 0.5)
- void **glRect**{sifd}(TYPE x1, y1, x2, y2)
- void **glRect**{sifd}(Type *v1, TYPE *v2)
- Specifies a two-dimensional rectangle, using the standard data type by the x and y values of the corners or by pointers to arrays with these values
- glRect supports efficient specification of rectangles as two corner points

Curves and Curved Surfaces

- Approximate smoothly curved line or surface by short line segments or small polygonal regions



OpenGL Geometric Drawing Primitives



Displaying Points and Lines

- By default,
  - a point is drawn as a single pixel on the screen
  - a line is drawn solid and one pixel wide
  - polygons are drawn solidly filled in
- void **glPointSize**(GLfloat size);
  - Sets the width in pixels for rendered points; size must be greater than 0.0 and by default is 1.0
- void **glLineWidth**(GLfloat width);
  - Sets the width in pixels for rendered lines; width must be greater than 0.0 and by default is 1.0
- void **glLineStipple**(GLint factor, GLushort pattern);
  - Sets the current stippling pattern for lines
  - Must be enabled or disabled by passing GL_LINE_STIPPLE to **glEnable();** or **glDisable();**

Displaying Polygons

- void **glPolygonMode**(GLenum face, GLenum mode);
  - Controls the drawing mode for a polygon's front and back faces
  - The parameter face can be GL_FRONT_AND_BACK, GL_FRONT, or GL_BACK
  - Mode can be GL_POINT, GL_LINE, or GL_FILL to indicate whether the polygon should be drawn as points, outlined, or filled (by default)
- void **glFrontFace**(GLenum mode);
  - Controls how front-facing polygons are determined
  - Mode can be GL_CCW (by default) or GL_CW
- void **glCullFace**(GLenum mode);
  - Indicates which polygons should be discarded (culled) before they're converted to screen coordinates
  - Mode can be GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK to indicate front-facing, back-facing, or all polygons
  - Must be enabled or disabled by passing GL_CULL_FACE to **glEnable();** or **glDisable();**
- void **glPolygonStipple**(const GLubyte *mask);
  - Defines the current stipple pattern for filled polygons
  - Must be enabled and disabled by using **glEnable()** and **glDisable()** with GL_POLYGON_STIPPLE as the argument

Interaction by GLUT Event Loops and Callback Functions

- Interactive programs are based on the program’s reacting to a variety of discrete **events**:
  - Mouse events: moving the mouse or clicking a mouse button
  - Keyboard events: pressing a key
  - Window events: resizing a window or covering up a window by another
- Events are processed sequentially from the event queue
- **Callback functions** define how the program should react to specific events
  - **glutDisplayFunc**(void (**func*)(void)) is called each time there is a display callback
  - **glutReshapeFunc**(void (*func)(int w, int h)): indicates what action should be taken when the window is resized
  - **glutIdleFunc**(void (*func)(void)): executes function func() whenever no other events are to be handled
  - **glutKeyboardFunc**(void (*func)(unsigned char key, int x, int y)) and **glutMouseFunc**(void (*func)(int button, int state, int x, int y)): links a keyboard key or a mouse button with a routine that's invoked when the key or mouse button is pressed or released.
  - **glutMotionFunc**(void (*func)(int x, int y)): registers a routine to call back when the mouse is moved while a mouse button is also pressed.

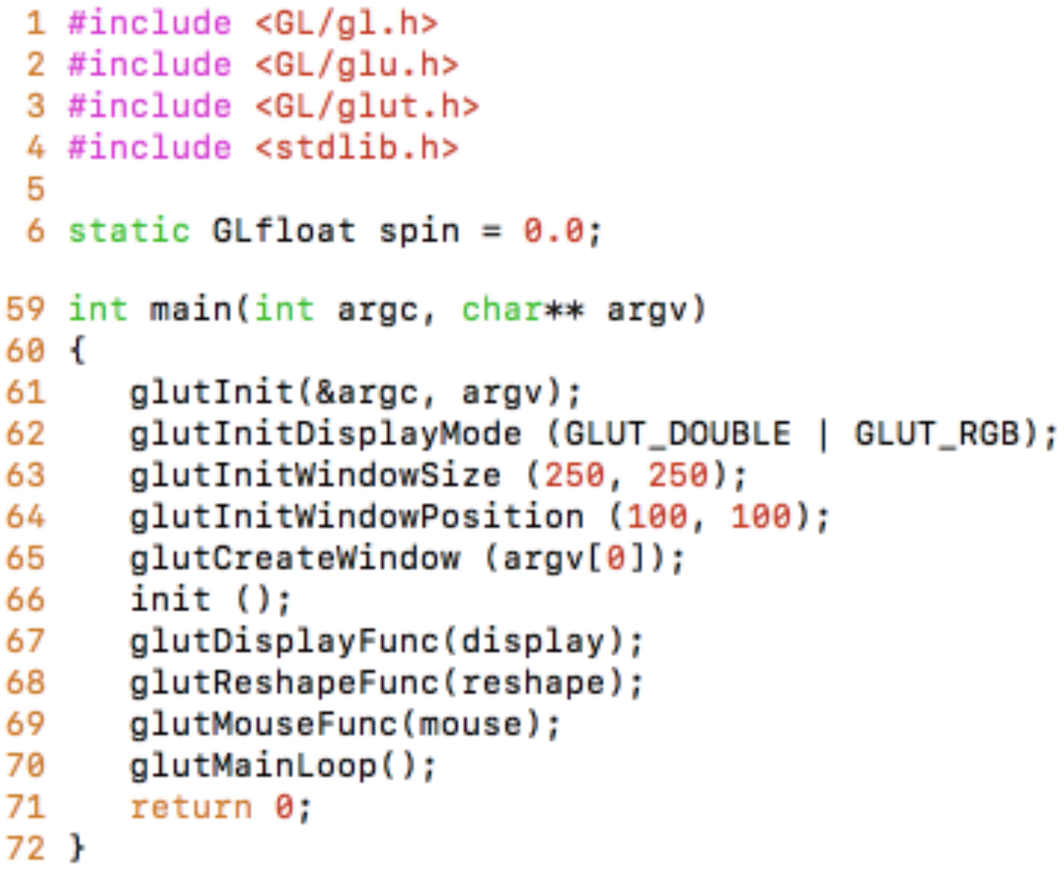
Animation

- Motion is typically achieved by taking a sequence of pictures and projecting them on the screen
- The key for motion picture projection to work is that each frame is complete when it is displayed
- Double buffering ensure that we will display a single fully drawing
    - Front buffer: the color buffer that is displayed by the display hardware
    - Back buffer: the color buffer that the application draws into
    - Swap the two buffers after each drawing by **glutSwapBuffers**()
        - Must require double buffering in the GLUT initialization: glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

Motion = Redraw + Swap

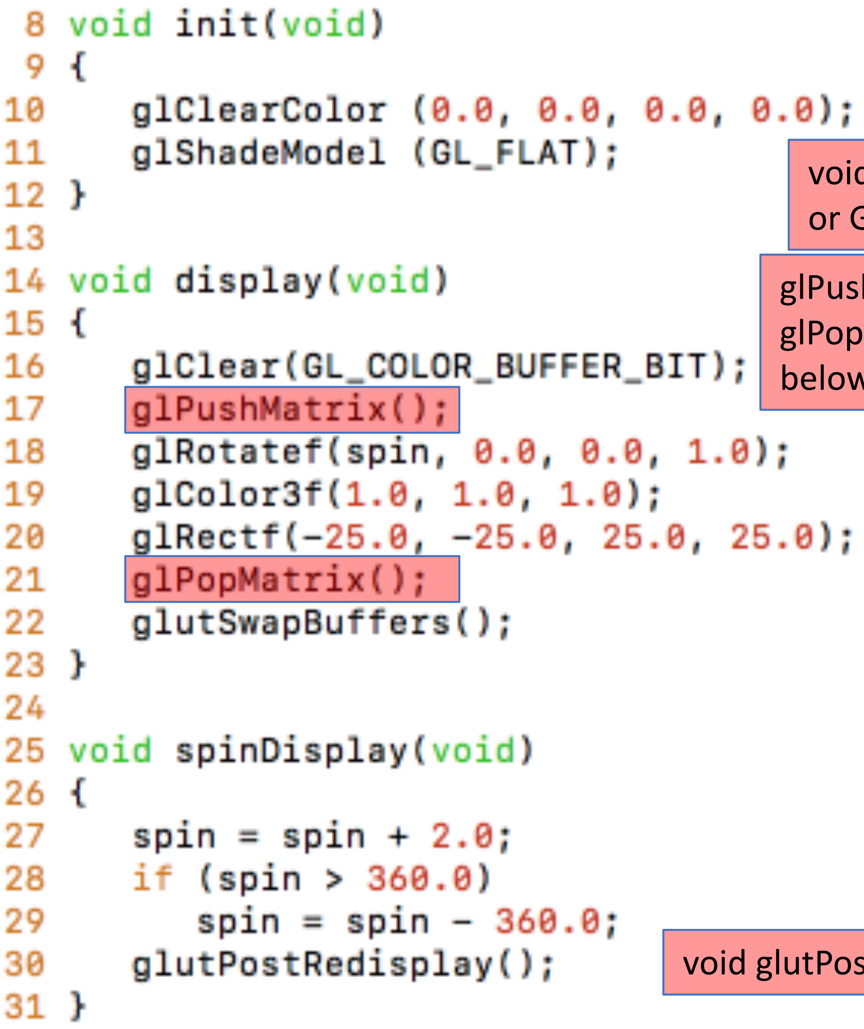
Example: Double-Buffered Rotating Square

- Main()



```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdlib.h>

static GLfloat spin = 0.0;

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMouseFunc(mouse);
   glutMainLoop();
   return 0;
}
```

Example: Double-Buffered Rotating Square



```
void init(void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel (GL_FLAT);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glRectf(-25.0, -25.0, 25.0, 25.0);
    glPopMatrix();
    glutSwapBuffers();
}

void spinDisplay(void)
{
    spin = spin + 2.0;
    if (spin > 360.0)
        spin = spin - 360.0;
    glutPostRedisplay();
}
```

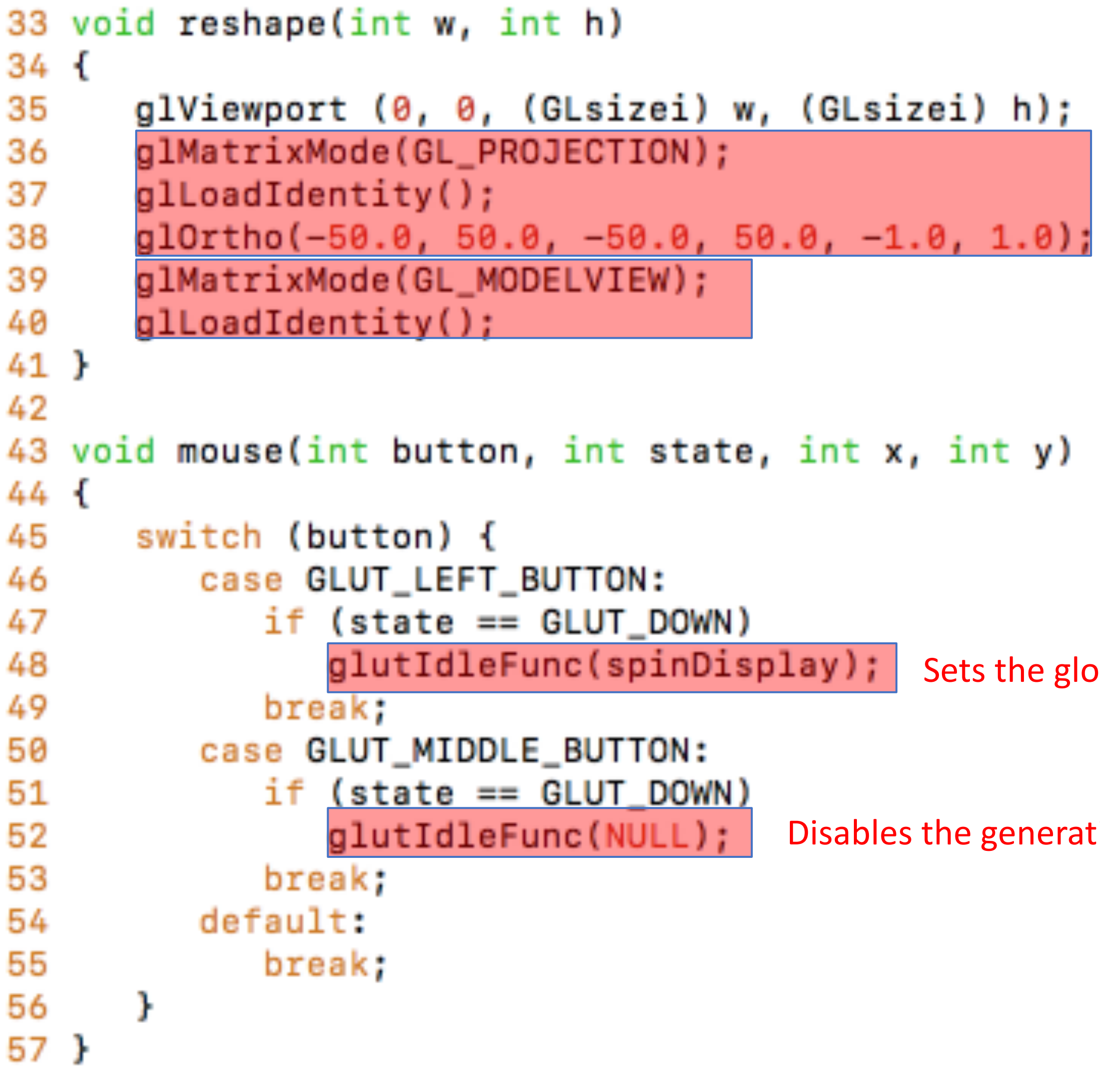
void glShadeModel(GLenum mode): selects flat of smooth shading. Mode can be GL_FLAT or GL_SMOOTH

glPushMatrix() pushes the current matrix stack down by one, duplicating the current matrix
glPopMatrix() pops the current matrix stack, replacing the current matrix with the one below it on the stack

void glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z): multiplies the current matrix by a rotation matrix.
angle: specifies the angle of rotation, in degrees
x, y, z: specify the x, y, and z coordinates of a vector, respectively

void glutPostRedisplay(void): marks the current window as needing to be redisplayed

Example: Double-Buffered Rotating Square



```
void reshape(int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void mouse(int button, int state, int x, int y)
{
    switch (button) {
        case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN)
                glutIdleFunc(spinDisplay);
            break;
        case GLUT_MIDDLE_BUTTON:
            if (state == GLUT_DOWN)
                glutIdleFunc(NULL);
            break;
        default:
            break;
    }
}
```

void glViewport(GLint x, Glint y, Glsizei width, Glsizei height); sets the viewport
x, y: specify the lower left corner of the viewport rectangle, in pixels
width, height: specify the width and height of the viewport

Sets the global idle callback to be spinDisplay

Disables the generation of the idle callback