Introduction OpenGL

What is OpenGL?

- A software interface to graphics hardware
- Graphics rendering API (Low Level)
  - High-quality color images composed of geometric and image primitives
  - Window system independent
  - Operating system independent

What is OpenGL? …

- The OpenGL API is defined as a state machine.
- Almost all of the OpenGL functions set or retrieve some state in OpenGL.

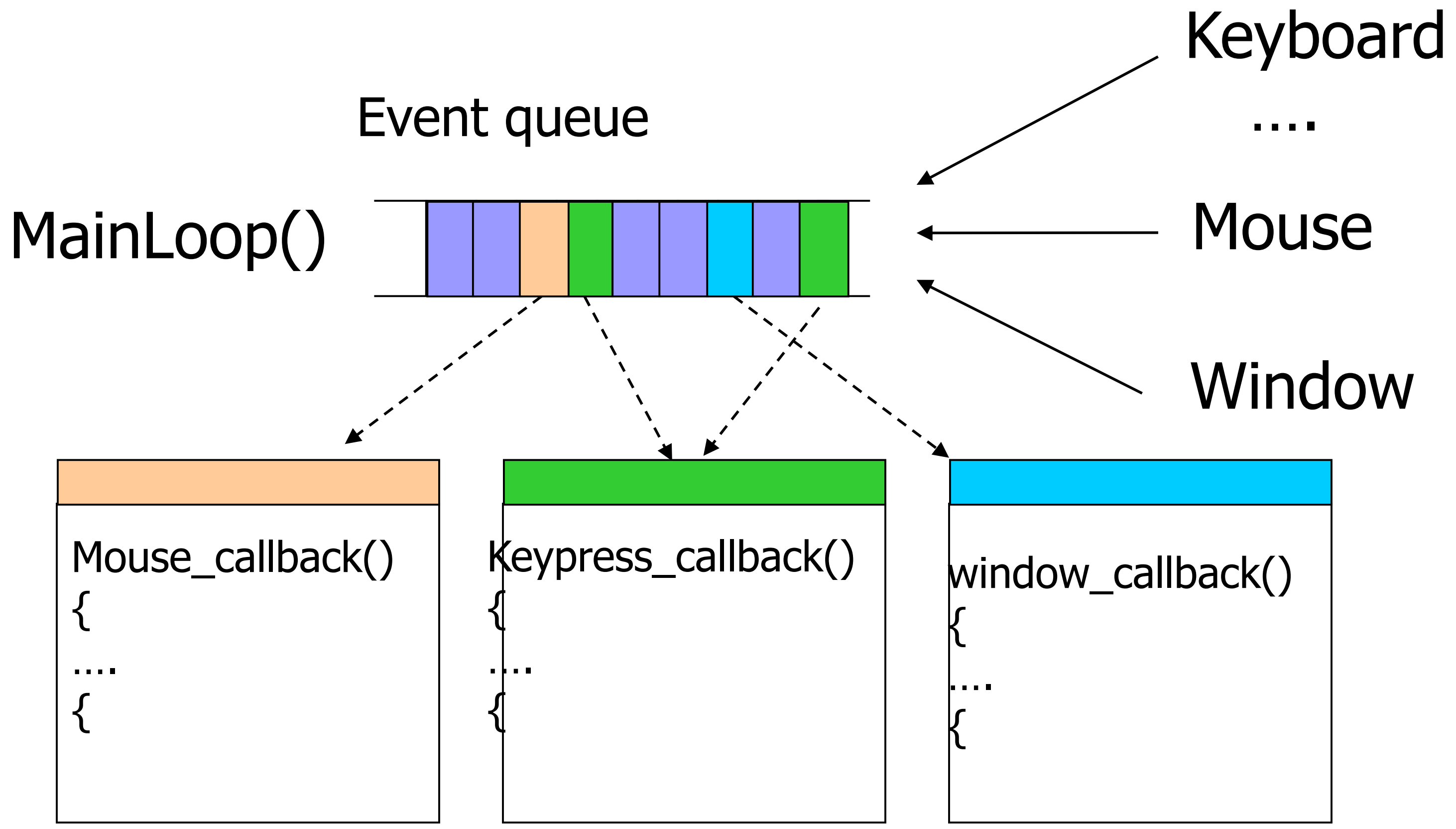
OpenGL and GLUT

- GLUT (OpenGL Utility Toolkit)
  - An auxiliary library
    - A portable windowing API
    - Easier to show the output of your OpenGL application
    - Not officially part of OpenGL
  - Handles:
    - Window creation,
    - OS system calls
      - Mouse buttons, movement, keyboard, etc…
    - Callbacks

GLUT Callback Functions

- **Events –** key press, mouse button press and release, window resize, etc.
- **Event-driven Programs** that use windows Input/Output
  - They wait until an event happens and then execute some pre-defined functions according to the user's input

Event Queue


Event queue
MainLoop()
Keyboard
....
Mouse
Window
Mouse_callback()
{
....
{
Keypress_callback()
{
....
{
window_callback()
{
....
{

GLUT with VS2010

- Download GLUT
  - http://www.opengl.org/resources/libraries/glut/glutdlls37beta.zip
- Copy the files to the following folders:
  - glut.h

  → C:\Program Files (x86)\MicrosoftSDKs\Windows\v7.0A\Include\gl
  - glut.dll, glut32.dll

  → C:\Windows\SysWOW64 (windows7 64 bit)

  → C:\Windows\System32 (windows7 32 bit)
  - glut.lib, glut32.lib

  → <D>:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\lib
    - (where <D> is VS2010 installed disk)

GLUT with VS2010 …

- Header Files:
  - #include <GL/glut.h>
  - #include <GL/gl.h>

  – Include glut automatically includes other header files

GLUT Basics

- Application Structure
  - Configure and open window
  - Initialize OpenGL state
  - Register input callback functions
    - render
    - resize
    - input: keyboard, mouse, etc.
  - Enter event processing loop

Your OpenGL program will be in infinite loop

GLUT Callback Functions

- **Callback function** : Routine to call when an event happens
  - Window resize or redraw
  - User input (mouse, keyboard)
  - Animation (render many frames)
- “Register” callbacks with GLUT
  - glutDisplayFunc(my_display_func);
  - glutIdleFunc(my_idle_func);
  - glutKeyboardFunc(my_key_events_func);
  - glutMouseFunc (my_mouse_events_func);

Getting Started

To start your own program in VC++ do the following.

0) Start VC++

1) File->New->Project

2) Select the “Win32 Console Application” and pick a name and directory

3) Press “Next” -> Select "empty project" -> “Finish”

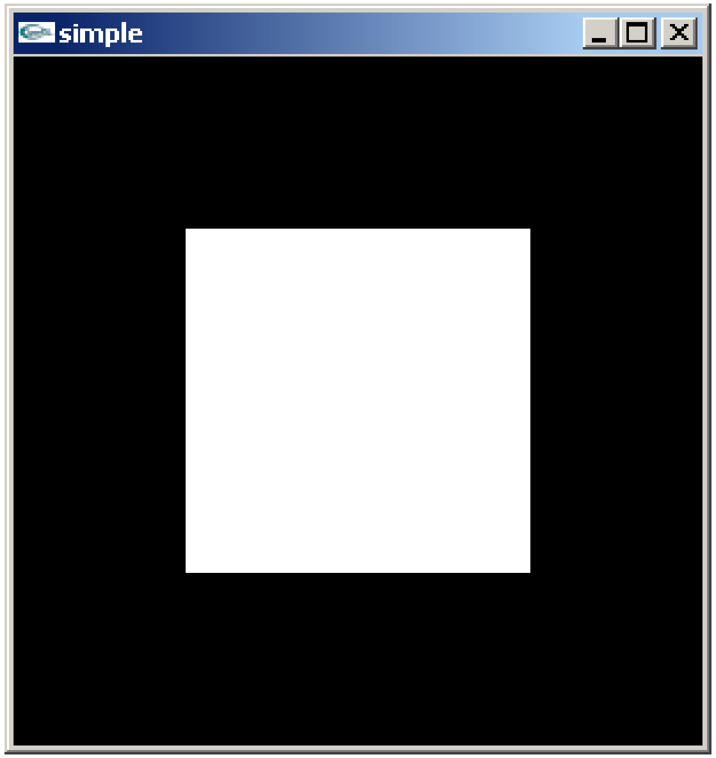
3) Project->Add new item->C++ File (pick a name and directory)

4) Copy and paste the first program (“Primitives.cpp”)

5) Compile and execute!

A Simple Program

Generate a square on a solid background



simple.cpp

```
#include <GL/glut.h>
void mydisplay(){
     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);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    init();
    glutMainLoop();
}
```

Closer Look at the main()

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

int main(int argc, char** argv)
{
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(0,0);
   glutCreateWindow("simple");
   glutDisplayFunc(mydisplay);

   init();

   glutMainLoop();
}
```

includes `gl.h`

define window properties

rendering callback

set OpenGL state

enter event loop

init.c

black clear color

opaque window

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

   glColor3f(1.0, 1.0, 1.0);

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

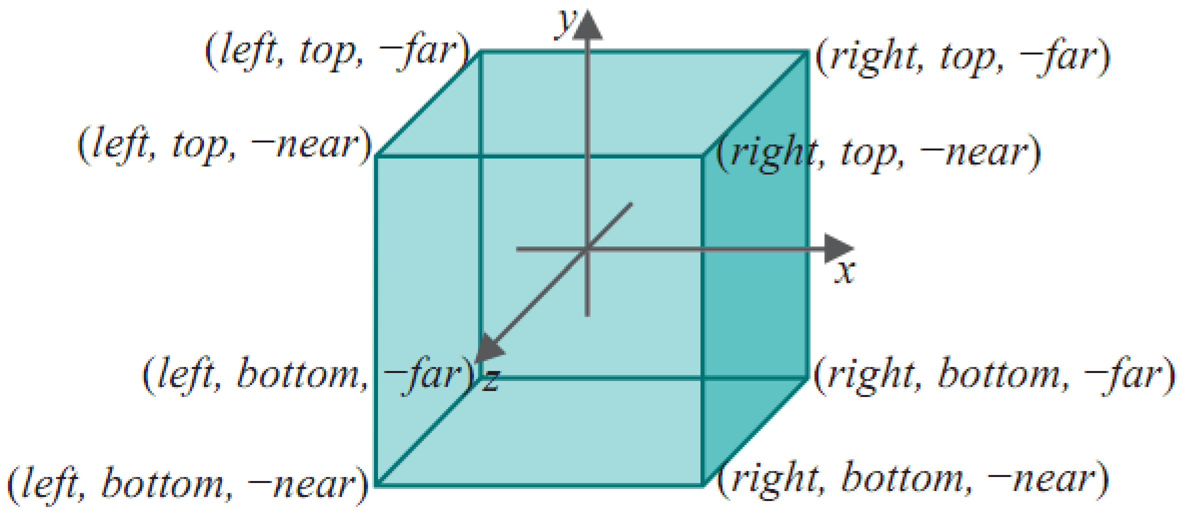
fill/draw with white

viewing volume

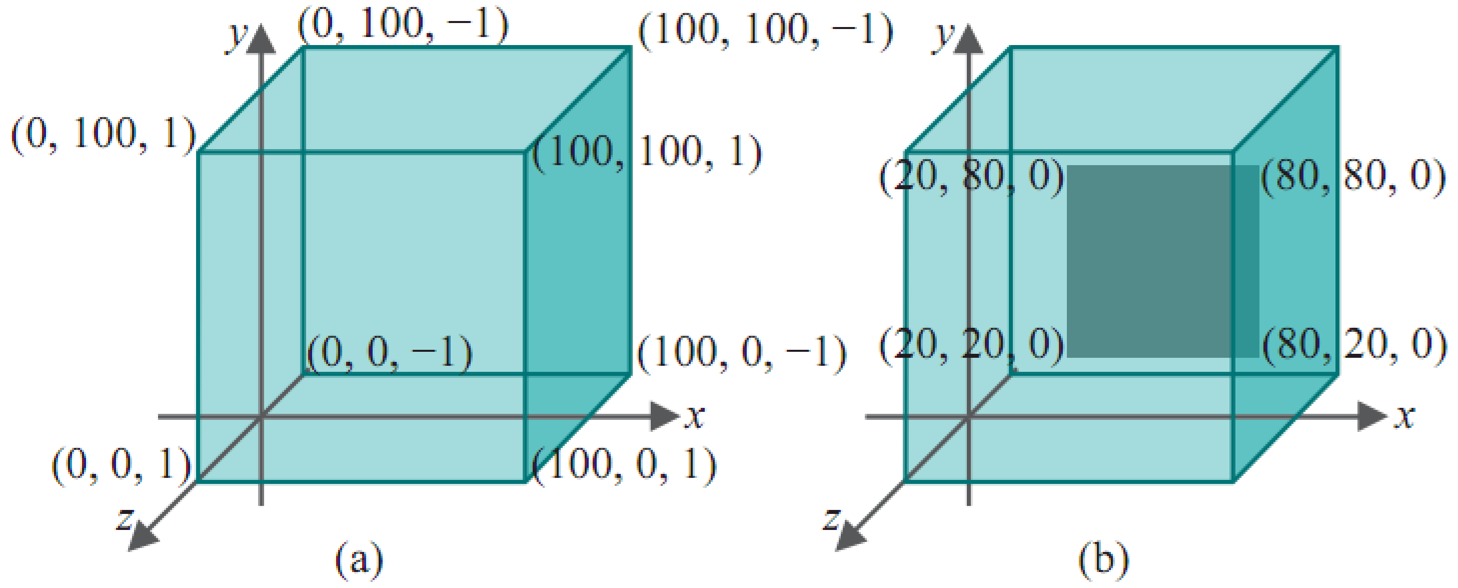
glOrtho(…)

- determines an imaginary viewing box inside which the programmer draws.

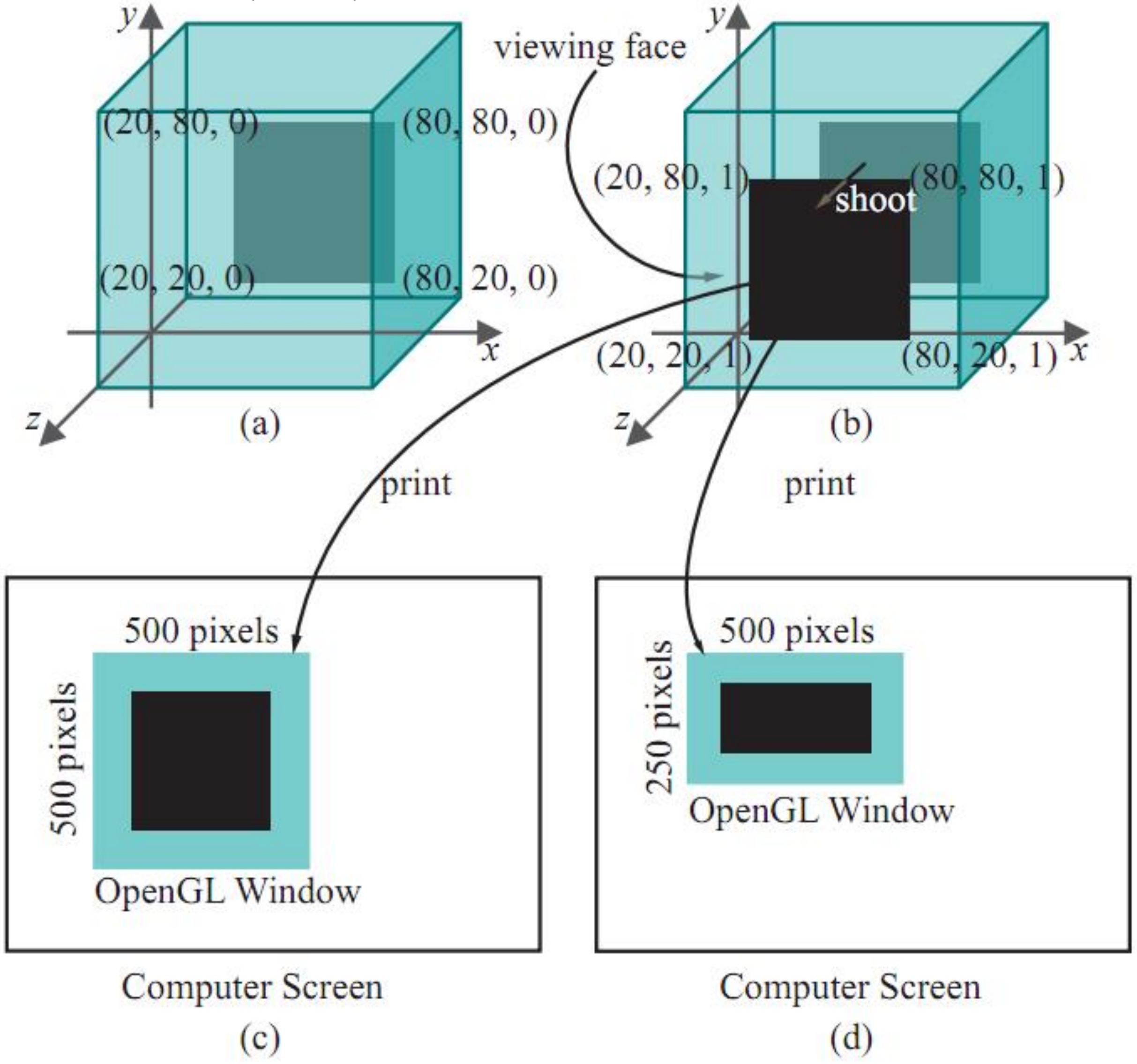
```
glOrtho(left, right, bottom, top, near, far)
```



glOrtho(…)


y
(0, 100, −1)
(100, 100, −1)
(0, 100, 1)
(100, 100, 1)
(0, 0, −1)
(100, 0, −1)
(0, 0, 1)
(100, 0, 1)
x
z
(a)
y
(20, 80, 0)
(80, 80, 0)
(20, 20, 0)
(80, 20, 0)
x
z
(b)

glOrtho(…)


y
x
z
(20, 80, 0)
(80, 80, 0)
(20, 20, 0)
(80, 20, 0)
(a)
viewing face
(20, 80, 1)
(80, 80, 1)
shoot
(20, 20, 1)
(80, 20, 1)
(b)
print
print
500 pixels
500 pixels
OpenGL Window
Computer Screen
(c)
500 pixels
250 pixels
OpenGL Window
Computer Screen
(d)

Vertices and Primitives

- Primitives are specified using

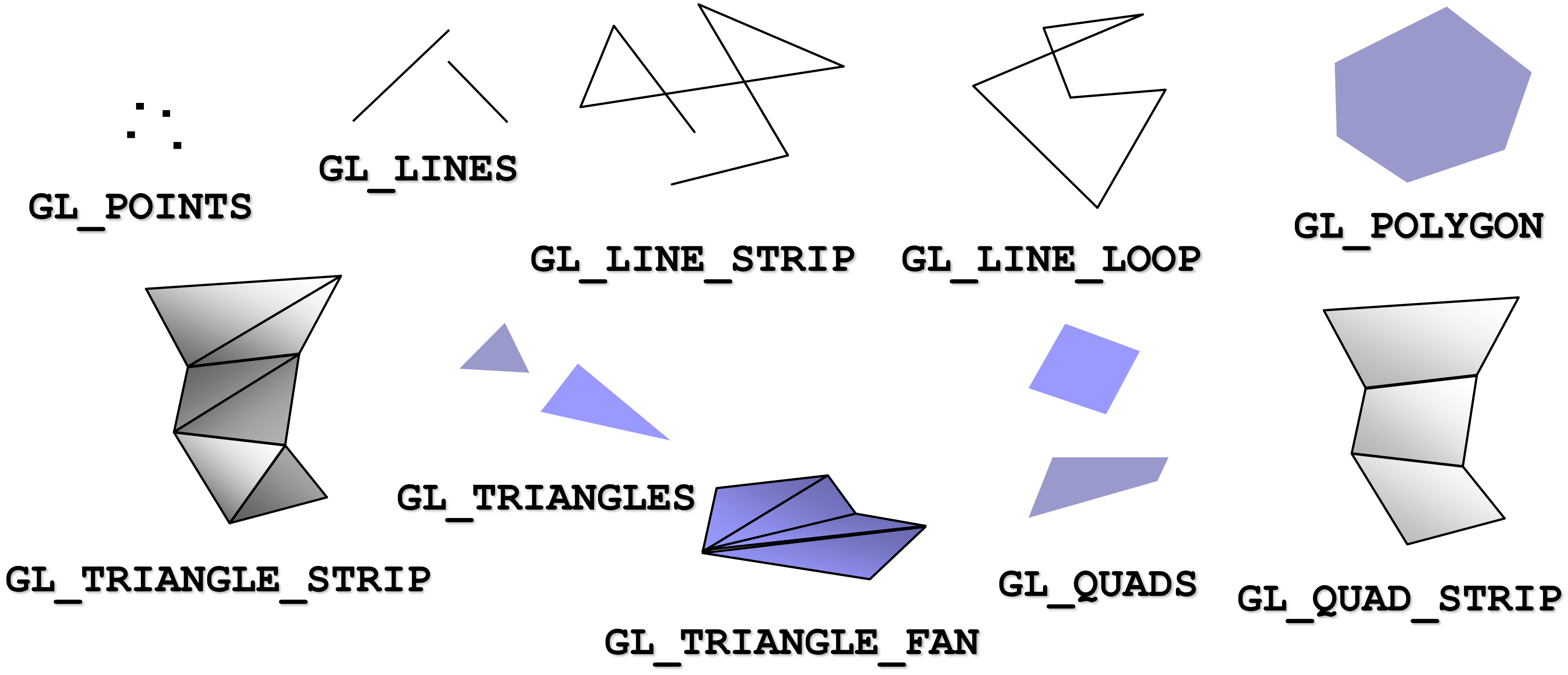
```
glBegin( primType );
…
glEnd();
```

  - *primType* determines how vertices are combined

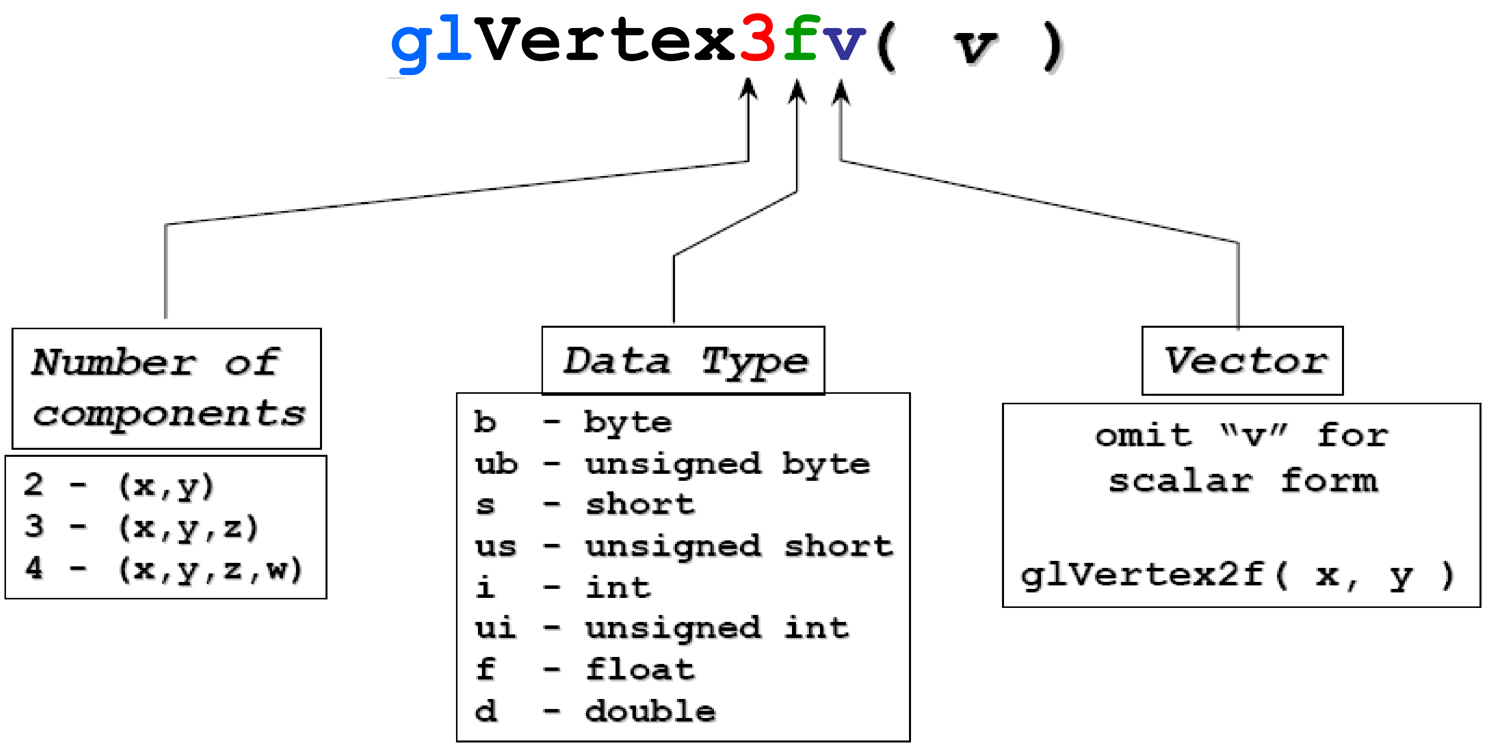
```
GLfloat red, green, blue;
Glfloat coords[nVerts][3];
/*Initialize coords and colors somewhere in program*/
glBegin( primType );
for ( i = 0; i < nVerts; ++i ) {
     glColor3f( red, green, blue );
     glVertex3fv( coords[i] );
}
glEnd();
```

Primitive Types

- All primitives are specified by vertices:



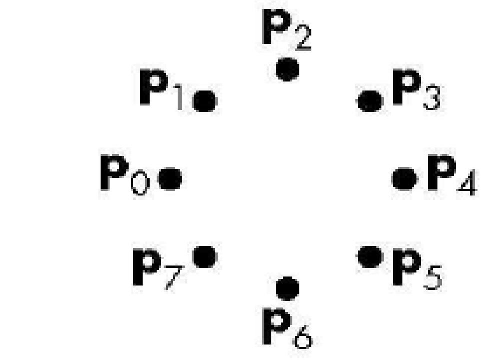
The glVertex command

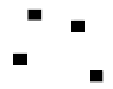


Vertices and Primitives

- Points, **`GL_POINTS`**
  - Individual points
  - Point size can be altered
    - ***glPointSize (float size)***

```
glBegin(GL_POINTS);
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```



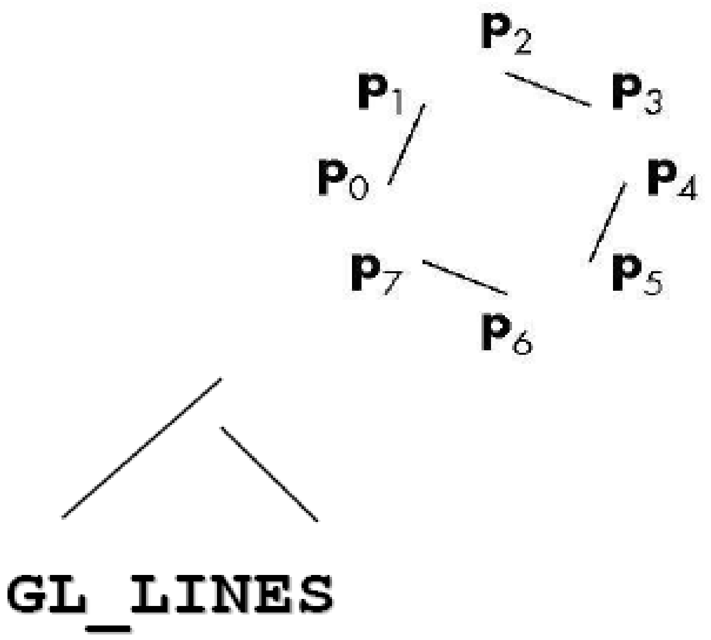


GL_POINTS

Vertices and Primitives

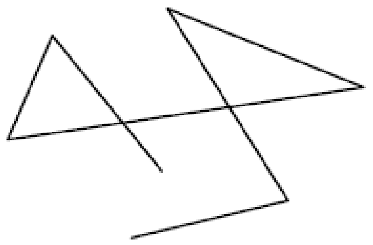
- Lines, **`GL_LINES`**
  - Pairs of vertices interpreted as individual line segments
  - Can specify line width using:
    - ***glLineWidth (float width)***

```
glBegin(GL_LINES);
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```

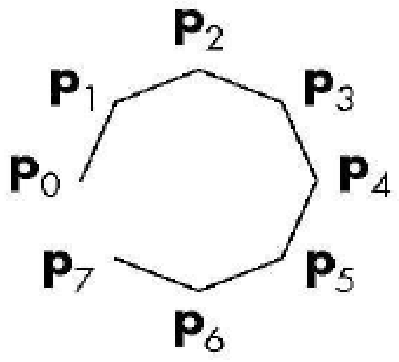


Vertices and Primitives

- Line Strip, **`GL_LINE_STRIP`**
  - series of connected line segments

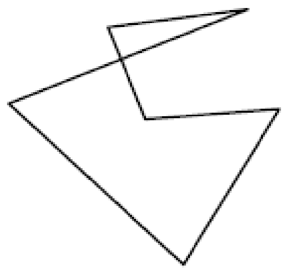


`GL_LINE_STRIP`

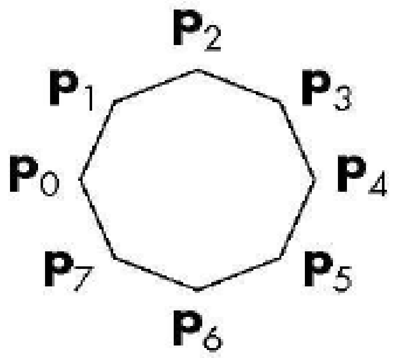


Vertices and Primitives

- Line Loop, **`GL_LINE_LOOP`**
  - Line strip with a segment added between last and first vertices

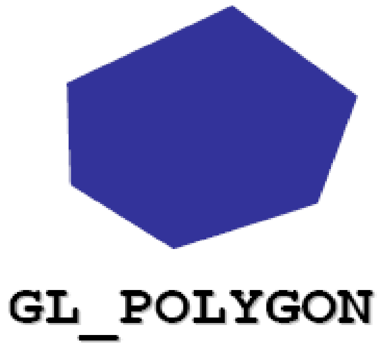


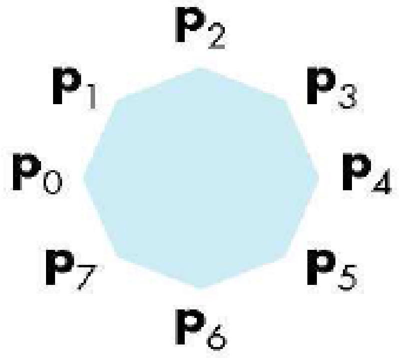
GL_LINE_LOOP



Vertices and Primitives

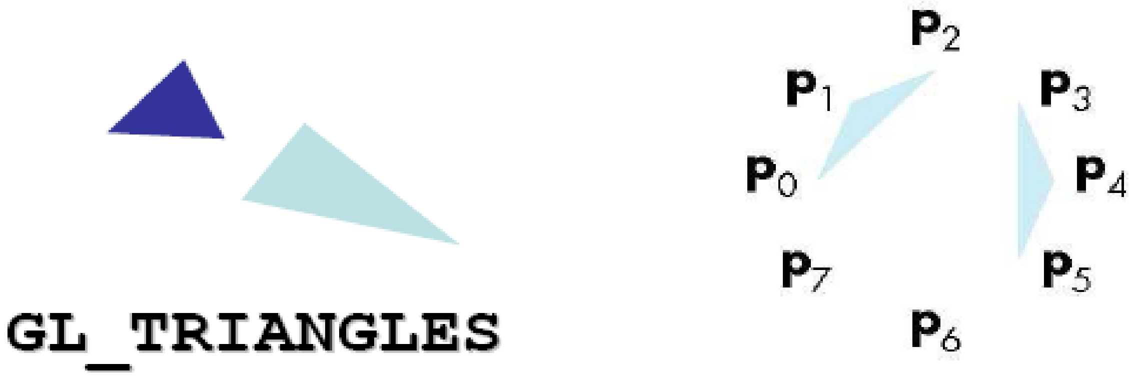
- Polygon , **`GL_POLYGON`**
  - boundary of a simple, convex polygon





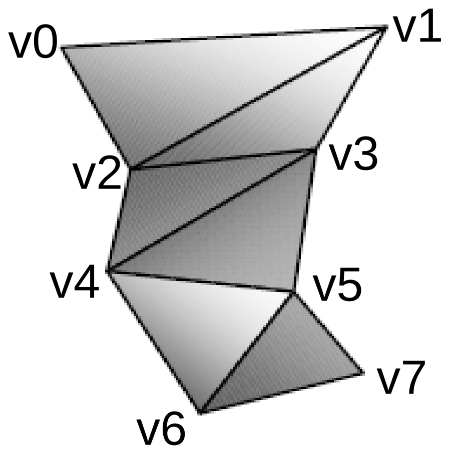
Vertices and Primitives

- Triangles , **`GL_TRIANGLES`**
  - triples of vertices interpreted as triangles



Vertices and Primitives

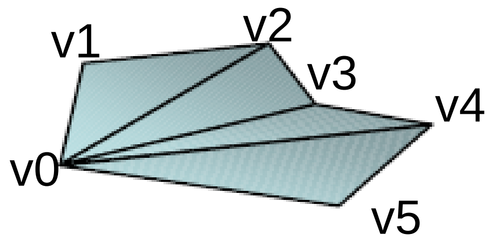
- Triangle Strip , **`GL_TRIANGLE_STRIP`**
  - linked strip of triangles



`GL_TRIANGLE_STRIP`

Vertices and Primitives

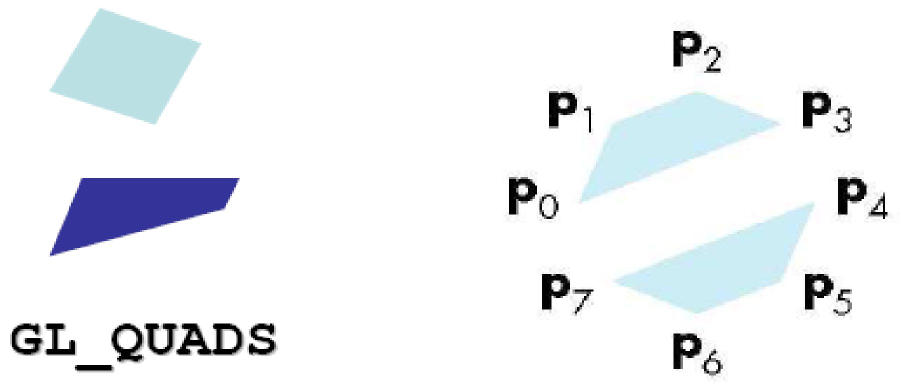
- Triangle Fan , **`GL_TRIANGLE_FAN`**
  - linked fan of triangles



`GL_TRIANGLE_FAN`

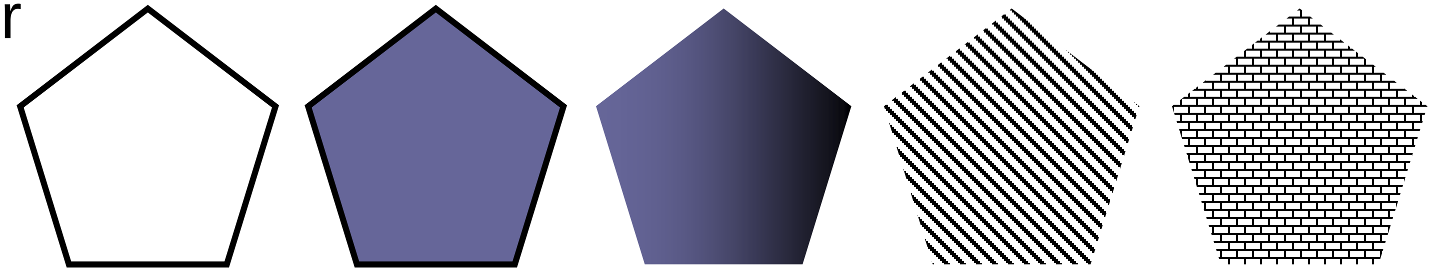
Vertices and Primitives

- Quads , **`GL_QUADS`**
  - quadruples of vertices interpreted as four-sided polygons

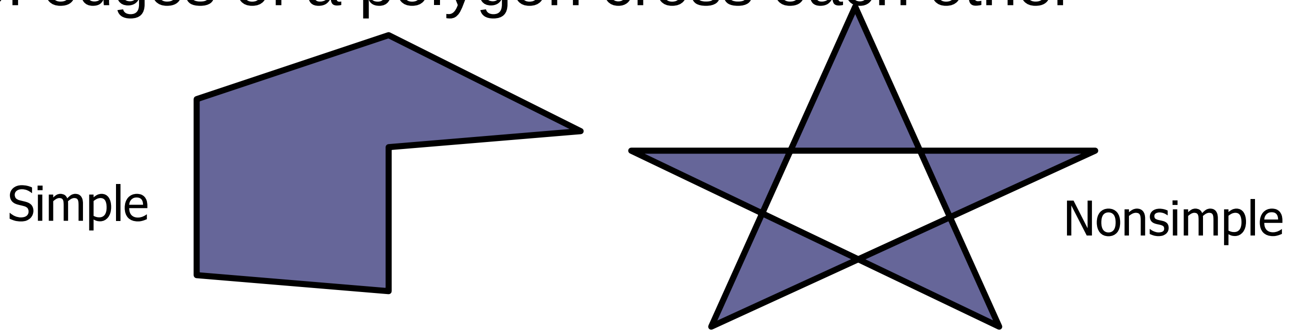


Polygons (1/2)

- Polygon - Definition
  - Object that is closed as in a line loop, but that has an interior

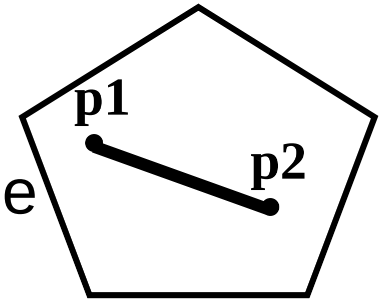


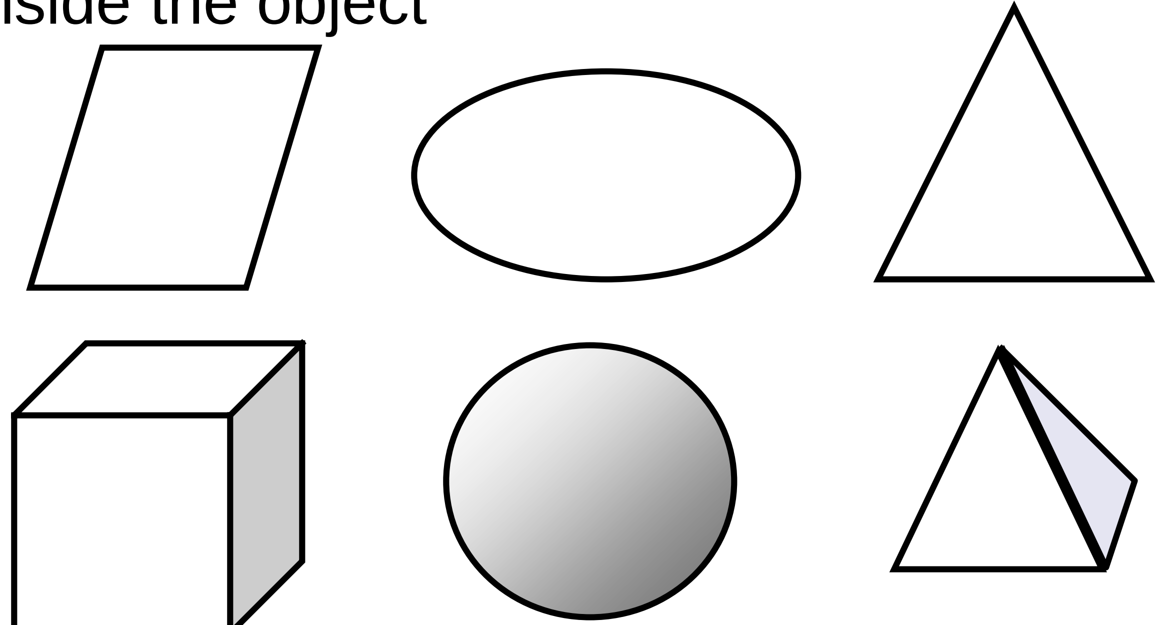
- Simple Polygon
  - No pair of edges of a polygon cross each other



Polygons (2/2)

- Convexity
  - If all points on the line segment between any two points inside the object, or on its boundary, are inside the object




Convex Objects

Polygon Issues

- OpenGL will only display polygons correctly that are
  - *Simple:* edges cannot cross
  - Convex: All points on line segment between two points in a polygon are also in the polygon
  - *Flat:* all vertices are in the same plane
- User program can check if above true
  - OpenGL will produce output if these conditions are violated but it may not be what is desired

Rendering Callback

- It’s a callback function where all our drawing is done
- Every GLUT program must have a display callback
- glutDisplayFunc(***my_display_func***); /* this part is in main.c */

```
void my_display_func (void )
{
   glClear( GL_COLOR_BUFFER_BIT );
   glBegin( GL_TRIANGLE );
    glVertex3fv( v[0] );
    glVertex3fv( v[1] );
    glVertex3fv( v[2] );
   glEnd();
   glFlush();
}
```

Idle Callback

- Use for animation and continuous update
  - Can use *glutTimerFunc* or *timed callbacks* for animations
- glutIdleFunc(***idle***);

```
void idle( void )
{
  /* change something */
  t += dt;
  glutPostRedisplay();
}
```

User Input Callbacks

- Process user input
- glutKeyboardFunc(my_key_events);

```
void my_key_events (char key, int x, int y )
{
   switch ( key ) {
    case ‘q’ : case ‘Q’ :
        exit ( EXIT_SUCCESS);
        break;
    case ‘r’ : case ‘R’ :
        rotate = GL_TRUE;
        break;
   }
}
```

Mouse Callback

- Captures mouse press and release events
- glutMouseFunc(my_mouse);

```
void myMouse(int button, int state, int x, int y)
{
      if (button == GLUT_LEFT_BUTTON && state ==
        GLUT_DOWN)
      {
      …
      }
}
```

Events in OpenGL

Event	**Example**	**OpenGL Callback Function**
Keypress	KeyDown KeyUp	`glutKeyboardFunc`
Mouse	leftButtonDown leftButtonUp	`glutMouseFunc`
Motion	With mouse press Without	`glutMotionFunc` `glutPassiveMotionFunc`
Window	Moving Resizing	`glutReshapeFunc`
System	Idle Timer	`glutIdleFunc` `glutTimerFunc`
Software	What to draw	`glutDisplayFunc`

Try…

- Write a program to draw polygons using the mouse.

you may use keyboard to specify the number of sides

References

1. *http://www.opengl.org/documentation/spec.html*

1. *http://www.opengl.org/documentation/red_book_1.0/*

1. *http://www.cs.rit.edu/~jdb/cg1/openGLIntro.pdf*

1. *http://www.**ceng**.**metu**.edu.tr/courses/**ceng**477/2005/documents/recitations/**opengl**.ppt*