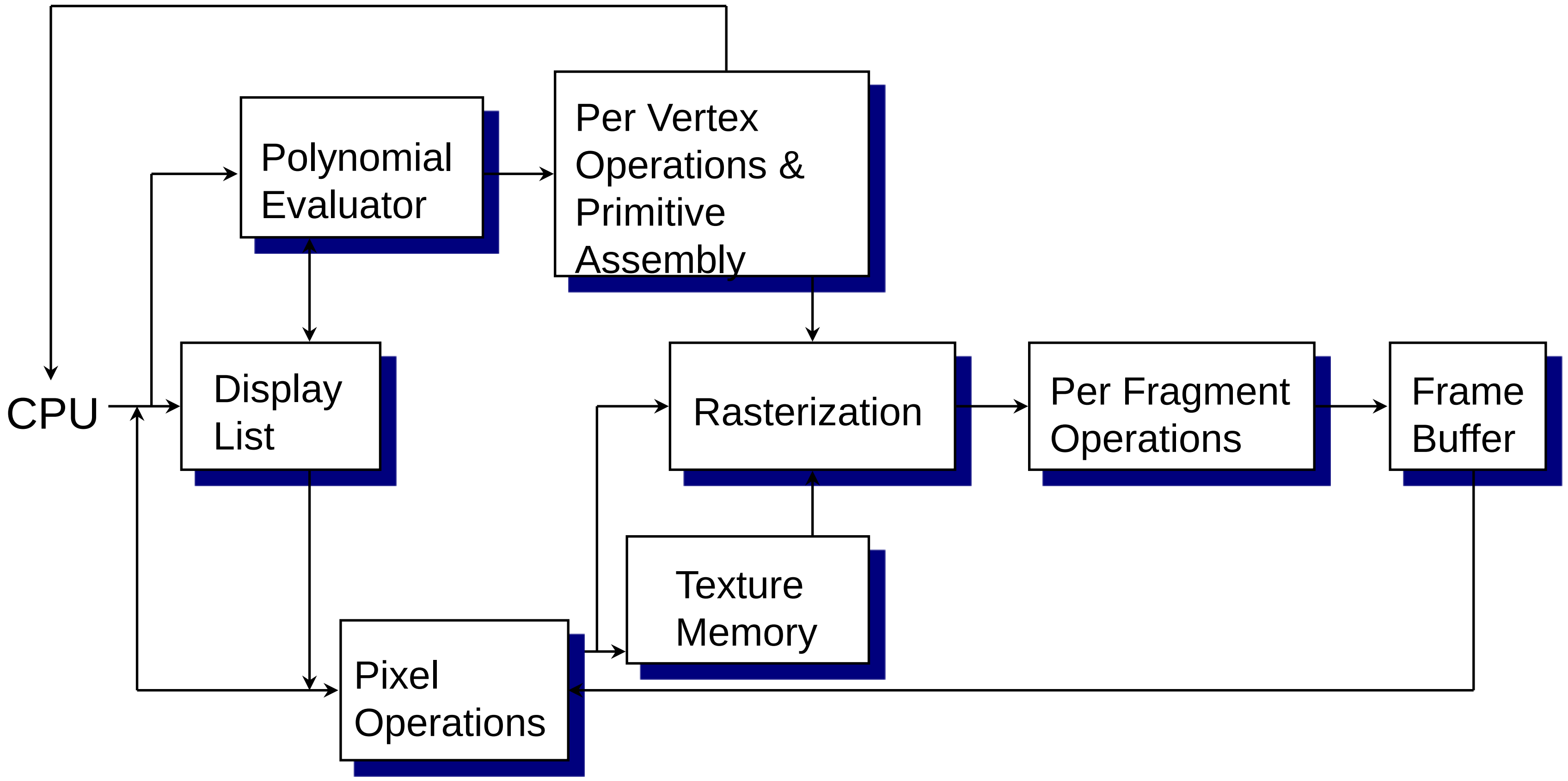
Introduction to Computer Graphics with OpenGL/GLUT

What is OpenGL?

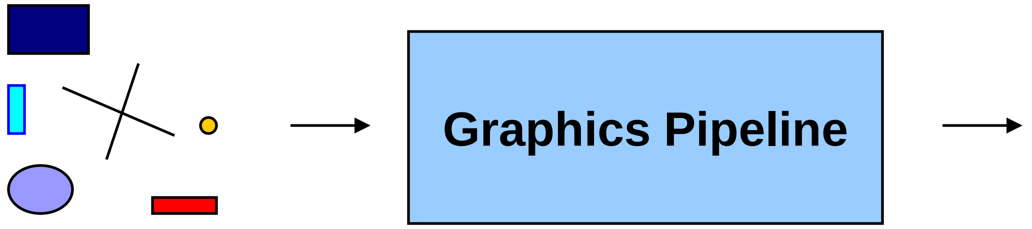
- A software interface to graphics hardware
- Graphics rendering API (Low Level)
  - High-quality color images composed of geometric and image primitives
  - Window system independent
  - Operating system independent

OpenGL Architecture



OpenGL Basics

- Rendering
  - Typically execution of OpenGL commands
  - Converting geometric/mathematical object descriptions into frame buffer values
- OpenGL can render:
  - Geometric primitives
    - Lines, points, polygons, etc…
  - Bitmaps and Images
    - Images and geometry linked through texture mapping



OpenGL and GLUT

- GL : Primitive, Shading & Color, Translation, Rotation & Scaling.
- GLU :Viewing, Image scaling.
- 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

How to install GLUT?

- Download GLUT
  - http://www.opengl.org/resources/libraries/glut.html
- Copy the files to following folders:
  - glut.h → MinGW/include/gl/
  - glut32.lib → MinGW/lib/
  - glut32.dll → windows/system32/
- Header Files:
  - #include <windows.h>
  - #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

Sample Program

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

void main(int argc, char** argv)
{
     int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow( “Simple” );
    init();
    glutDisplayFunc( display );
    glutKeyboardFunc( key );
    glutMainLoop();
}
```

Sample Program

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

void main(int argc, char** argv)
{
      int mode = GLUT_RGB|GLUT_DOUBLE;
     glutInitDisplayMode( mode );
     glutInitWindowSize( 500,500 );
     glutCreateWindow( "Simple" );
     init();
     glutDisplayFunc( display );
     glutKeyboardFunc( key );
     glutMainLoop();
}
```

$\leftarrow$ **Specify the display Mode – RGB or color Index, single or double Buffer**

Sample Program

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

void main(int argc, char** argv)
{
     int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow(  "Simple"  );
    init();
    glutDisplayFunc( display );
    glutKeyboardFunc( key );
    glutMainLoop();
}
```

$\longleftarrow$ **Create a window Named "simple" with resolution 500 x 500**

Sample Program

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

void main(int argc, char** argv)
{
     int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow( “Simple” );
    init();                                  ← Your OpenGL initialization
    glutDisplayFunc( display );                code (Optional)
    glutKeyboardFunc( key );
    glutMainLoop();
}
```

Sample Program

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

void main(int argc, char** argv)
{
     int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow( “Simple” );
    init();
    glutDisplayFunc( display );
     glutKeyboardFunc(key);
    glutMainLoop();
}
```

← **Register your call back functions**

glutMainLoop()

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

int main(int argc, char** argv)
{
     int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode(mode);
    glutInitWindowSize(500,500);
    glutCreateWindow( "Simple" );
    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glutMainLoop();
}
```

The program goes into an infinite loop waiting for events

OpenGL Initialization

- Set up whatever state you're going to use
  - Don't need this much detail unless working in 3D

```
void init( void )
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-10, 10, -10, 10, -10, 20);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glEnable( GL_LIGHT0 );
    glEnable( GL_LIGHTING );
    glEnable( GL_DEPTH_TEST );
}
```

GLUT Callback functions

- **Event-driven:** Programs that use windows
  - Input/Output
  - Wait until an event happens and then execute some pre-defined functions according to the user's input

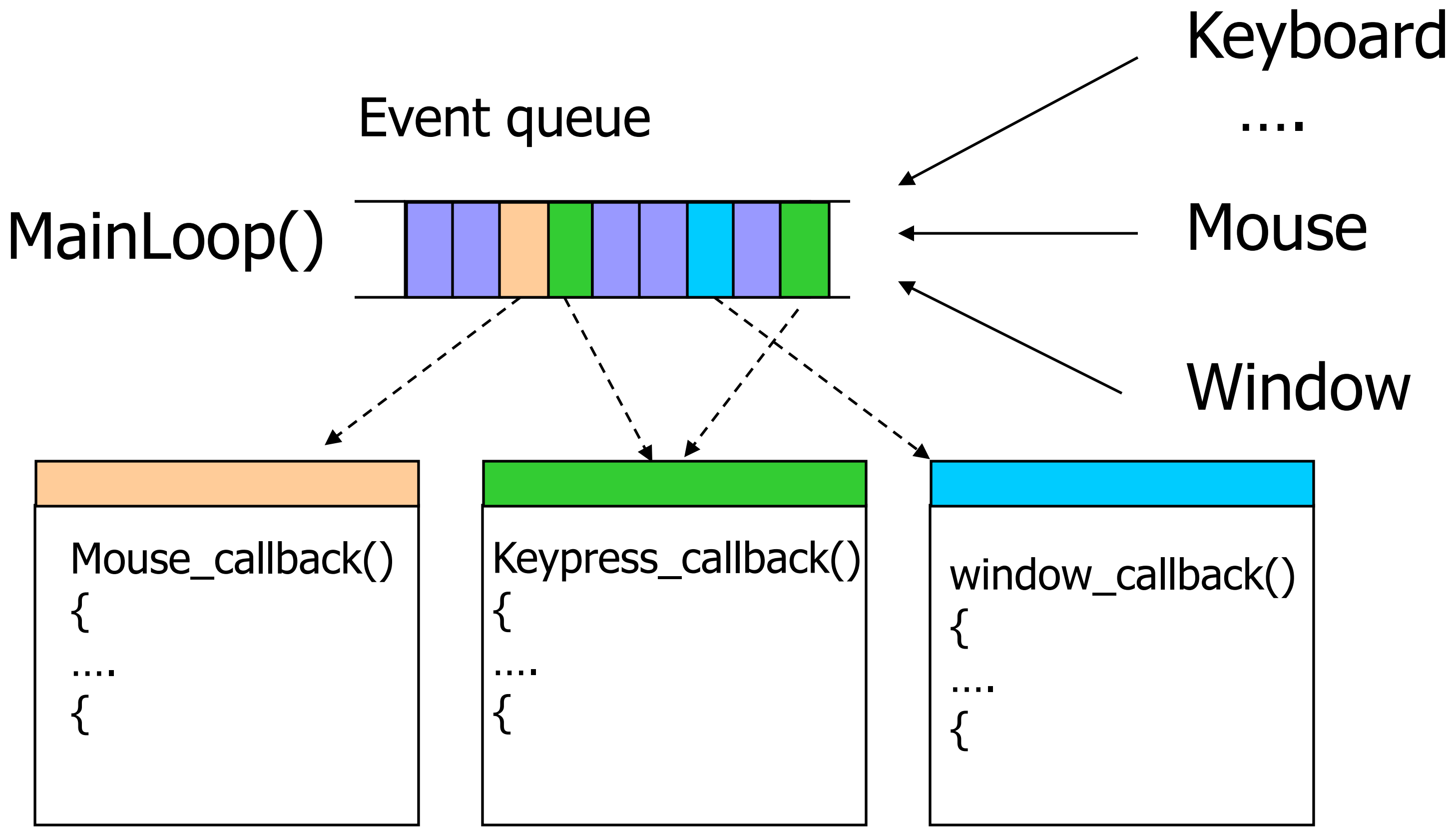
- **Events** – key press, mouse button press and release, window resize, etc.
- *Your OpenGL program will be in infinite loop*

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);

Event Queue


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

Rendering Callback

- 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)
    {
    …
    }
```

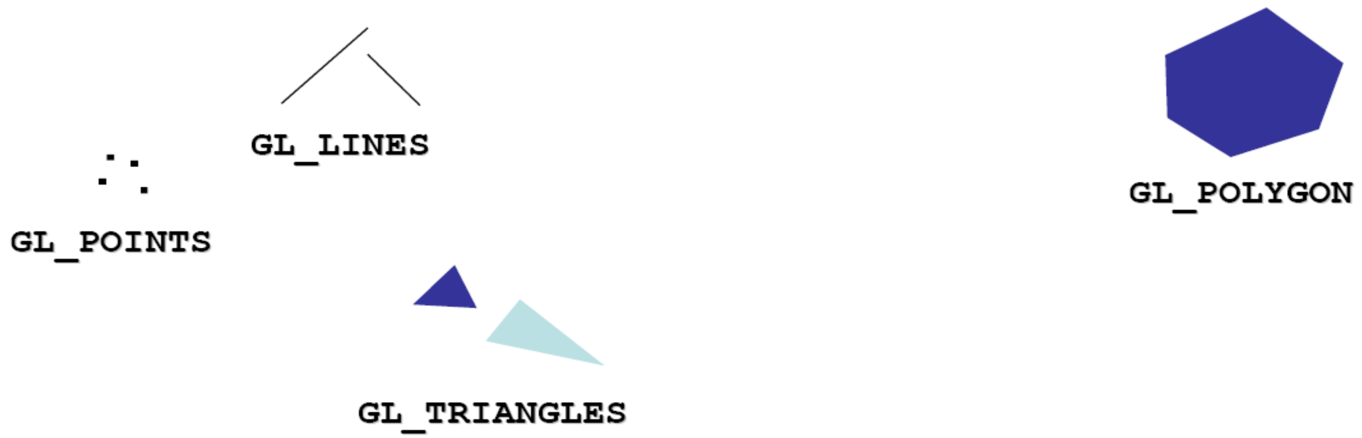
}

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`

OpenGL Geometric Primitives

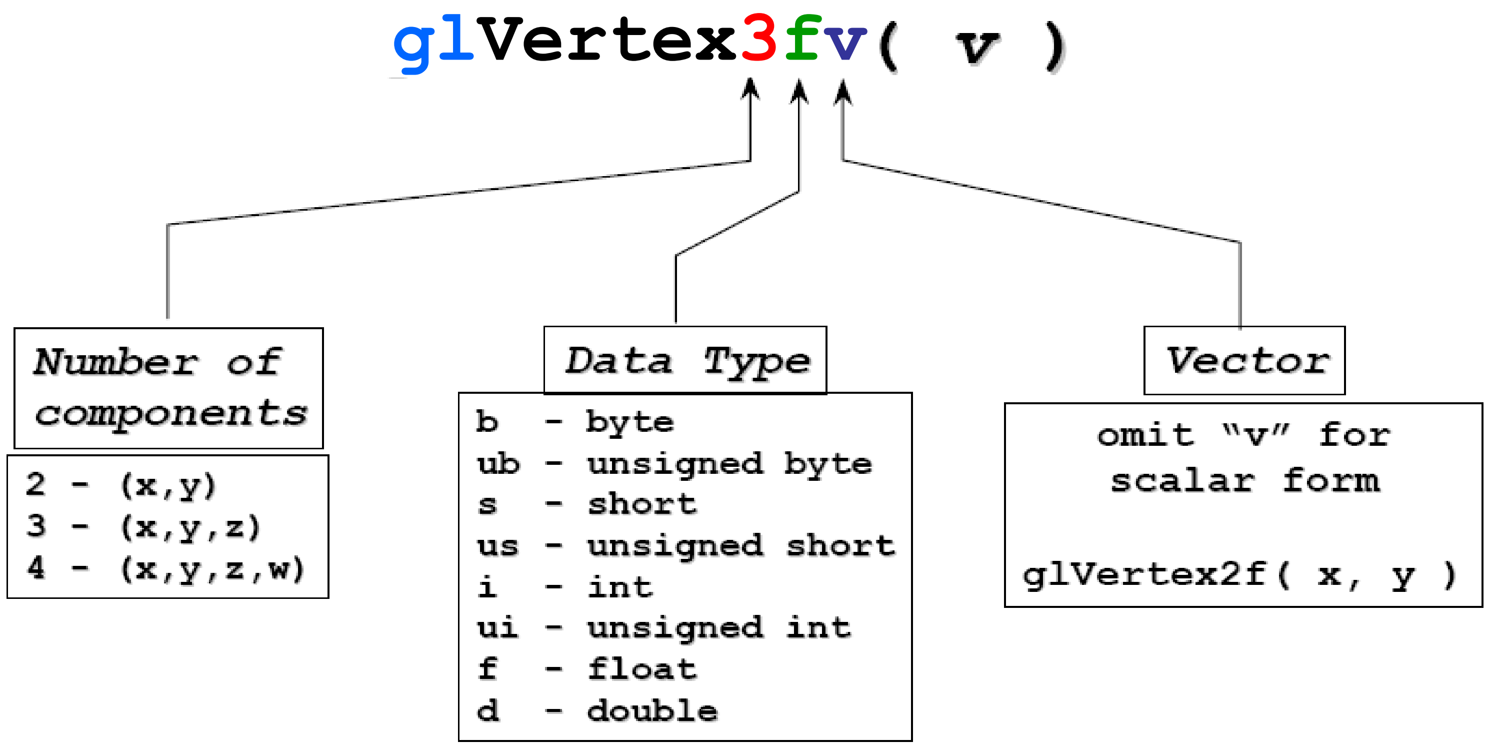
- The geometry is specified by vertices.
- There are ten primitive types:



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
- Triangles satisfy all conditions
- That's why we need triangulation algorithms!

OpenGL Command Format



`glVertex3fv( v )`

Number of components

2	(x,y)
3	(x,y,z)
4	(x,y,z,w)

Data Type

b	byte
ub	unsigned byte
s	short
us	unsigned short
i	int
ui	unsigned int
f	float
d	double

Vector

omit "v" for scalar form

`glVertex2f( x, y )`

Vertices and Primitives

- Primitives are specified using

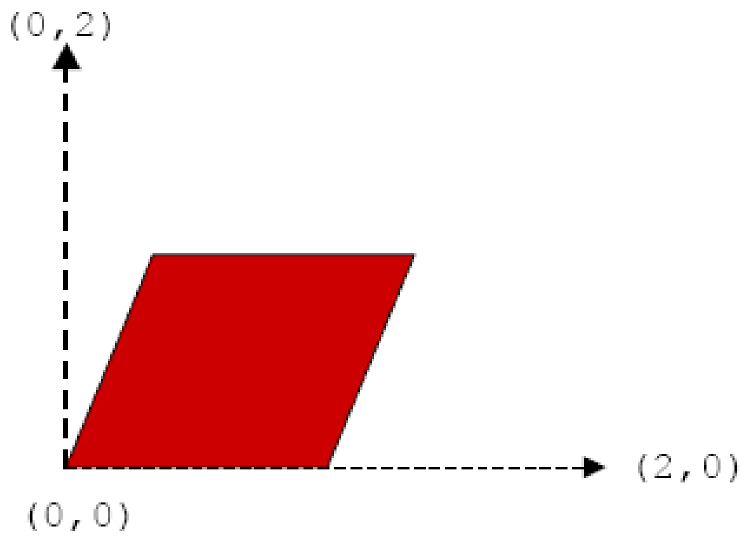
```
glBegin( primType );
…
glEnd();
```

  - *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();
```

An Example

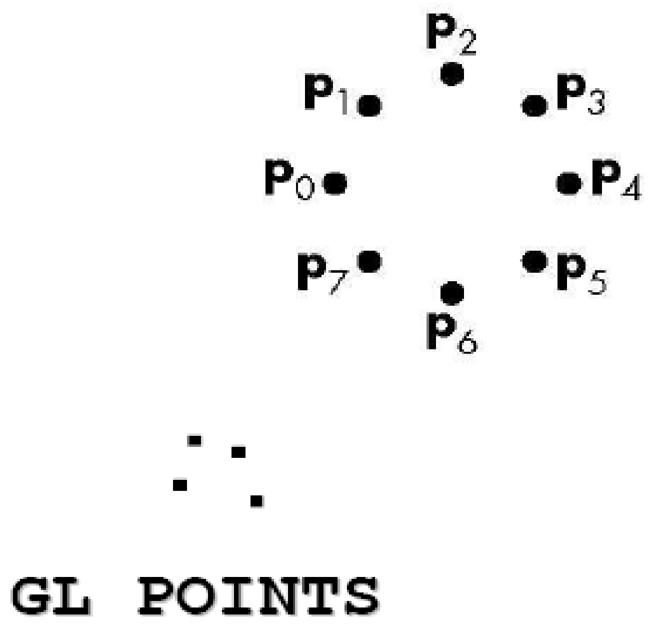
```
void drawParallelogram( GLfloat
   color[] )
{
    glBegin( GL_QUADS );
    glColor3fv( color );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( 1.0, 0.0 );
    glVertex2f( 1.5, 1.118 );
    glVertex2f( 0.5, 1.118 );
    glEnd();
}
```



Vertices and Primitives

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

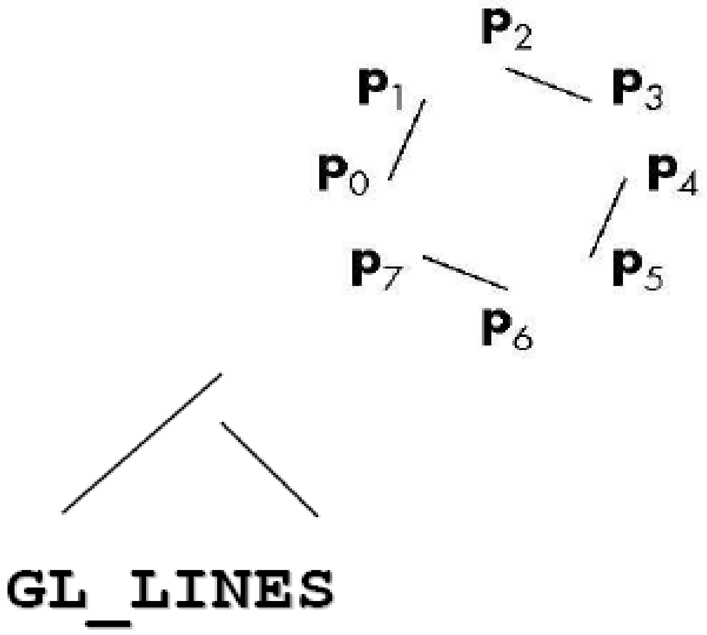
```
glBegin(GL_POINTS);
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```



Vertices and Primitives

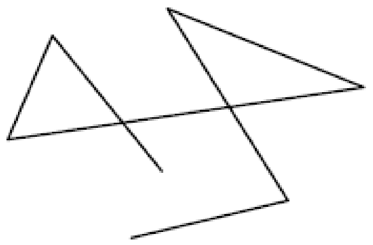
- Lines, **`GL_LINES`**
  - Pairs of vertices interpreted as individual line segments
  - Can specify line width using:
    - ***glLineWidth (float width)***

```
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();
```

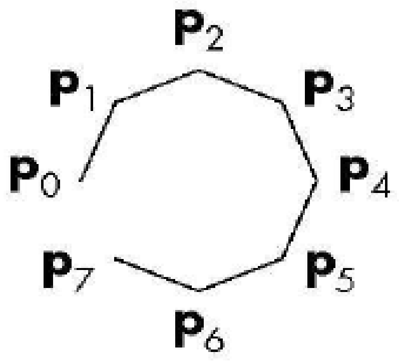


Vertices and Primitives

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

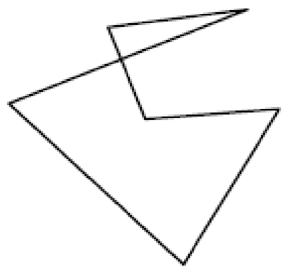


`GL_LINE_STRIP`

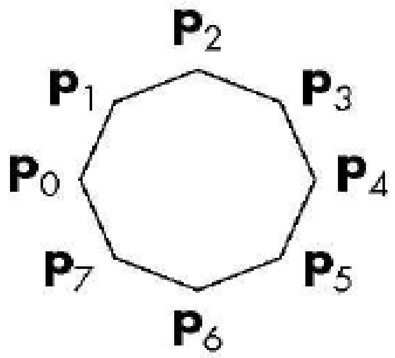


Vertices and Primitives

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

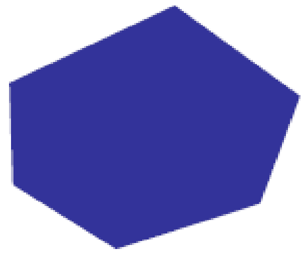


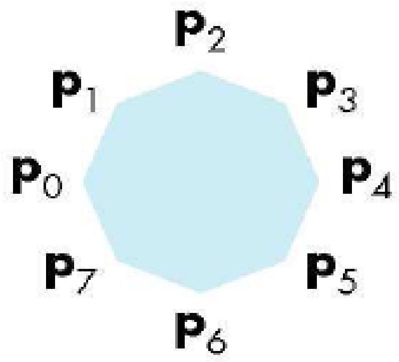
`GL_LINE_LOOP`



Vertices and Primitives

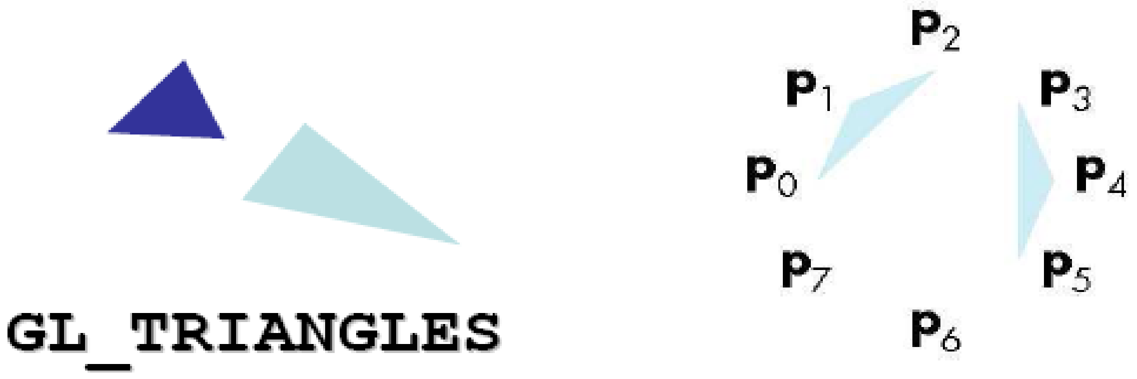
- Polygon , **`GL_POLYGON`**
  - boundary of a simple, convex polygon


GL_POLYGON



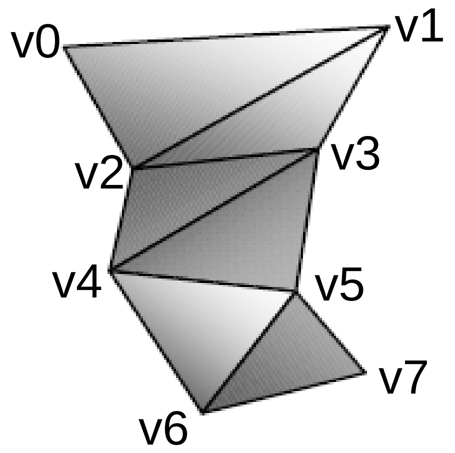
Vertices and Primitives

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



Vertices and Primitives

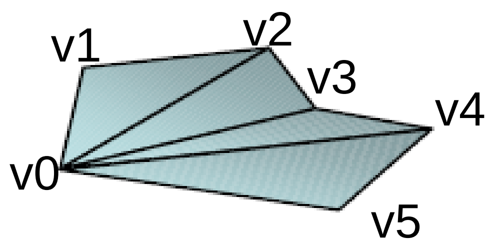
- Triangle Strip , **`GL_TRIANGLE_STRIP`**
  - linked strip of triangles



`GL_TRIANGLE_STRIP`

Vertices and Primitives

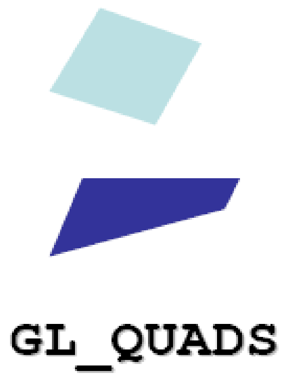
- Triangle Fan , **`GL_TRIANGLE_FAN`**
  - linked fan of triangles

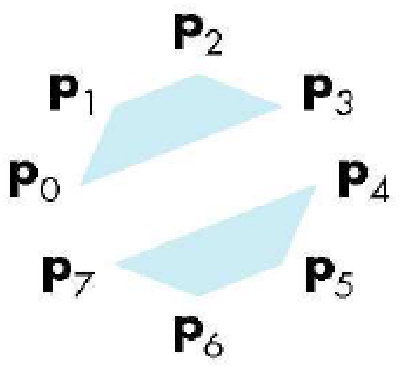


`GL_TRIANGLE_FAN`

Vertices and Primitives

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





Vertices and Primitives

- Between glBegin/ glEnd, those opengl commands are allowed:
  - glVertex*() : set vertex coordinates
  - glColor*() : set current color
  - glIndex*() : set current color index
  - glNormal*() : set normal vector coordinates (Light.)
  - glTexCoord*() : set texture coordinates (Texture)

References

1. *http://www.opengl.org/documentation/spec.html*
2. *http://www.opengl.org/documentation/red_book_1.0/*
3. *http://www.cs.rit.edu/~jdb/cg1/openGLIntro.pdf*
4. *http://www.**ceng**.**metu**.edu.tr/courses/**ceng**477/2005/documents/recitations/**opengl**.ppt*