Peter Mileff PhD

# Programming of Graphics

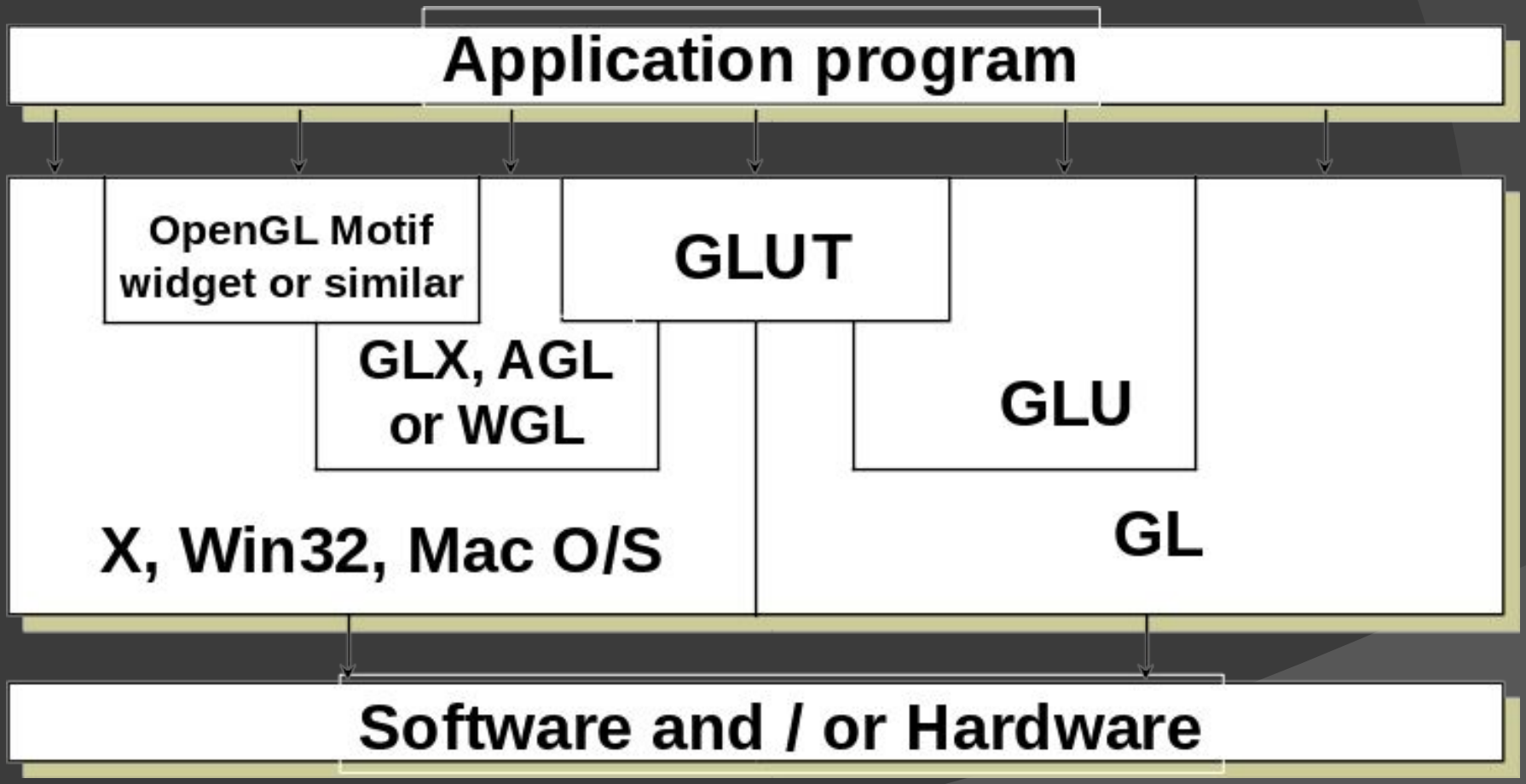
## Introduction to OpenGL

University of Miskolc
Department of Information Technology

# OpenGL libraries

- **GL (Graphics Library):** Library of 2D, 3D drawing primitives and operations
  - API for 3D hardware acceleration
- **GLU (GL Utilities):** Miscellaneous functions
  - dealing with camera setup and higher-level shape descriptions
- **GLUT (GL Utility Toolkit):** Window-system independent toolkit
  - with numerous utility functions, mostly dealing with user interface

# Software Organization



# The OpenGL Context...

# OpenGL Context

- **Before we do anything, an OpenGL Context should be created**

- An OpenGL context represents many things:
  - A context stores all of the state associated with this instance of OpenGL
  - It represents the (potentially visible) default framebuffer that rendering commands will draw to
  - Think of a context as an object that holds all of OpenGL
  - when a context is destroyed, OpenGL is destroyed

# A JOGL Application

- In order to create a Java window, we should extend the **JFrame** class from Swing
- We can access the OpenGL functionality from JOGL by implementing the **GLEventListener** interface

```
public class Game extends JFrame implements GLEventListener {
    private static final long serialVersionUID = 1L;

    public void display(GLAutoDrawable drawable) {
    }

    public void dispose(GLAutoDrawable drawable) {
    }

    public void init(GLAutoDrawable drawable) {
    }

    public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height){
    }
}
```

# OpenGL Context in JOGL

```
// Get GL profile
final GLProfile profile = GLProfile.get(GLProfile.GL2);
GLCapabilities capabilities = new GLCapabilities(profile);

// Create a canvas for 3D Graphics
final GLCanvas glcanvas = new GLCanvas(capabilities);

// Create Frame for canvas
final JFrame frame = new JFrame("3d Triangle (solid)");
frame.getContentPane().add(glcanvas);
frame.setSize(frame.getContentPane().getPreferredSize());
frame.setVisible(true);

final FPSAnimator animator = new FPSAnimator(glcanvas, 60, true);
animator.start();
```

# A JOGL Application

- Try and investigate the MyJogIDemoSimple application!

# OpenGL Context in JOGL

- **GLProfile:** determines the used OpenGL version
  - The field of computer graphics changes rapidly.
  - Graphics APIs must often break backwards compatibility with each revision.
  - It is important that programmers have control over which version of OpenGL is used by the application

- To select OpenGL 2.1, for example, a GLProfile would be created as follows:

```
GLProfile glp = GLProfile.get(GLProfile.GL2);
```

# OpenGL Context in JOGL

- **GLCapabilities:**

- This object describes some specific capabilities a rendering context should support
- The default settings are usually fine, but many options are available:
  - E.g: effects such as full-scene anti-aliasing, stereo rendering.

- This object takes an instance of GLProfile as a parameter.

```
GLCapabilities caps = new GLCapabilities(glp);
```

# OpenGL Context in JOGL

- **GLCanvas:**

- A heavyweight AWT component which provides OpenGL rendering support.
- Everything will be drawn to this component.

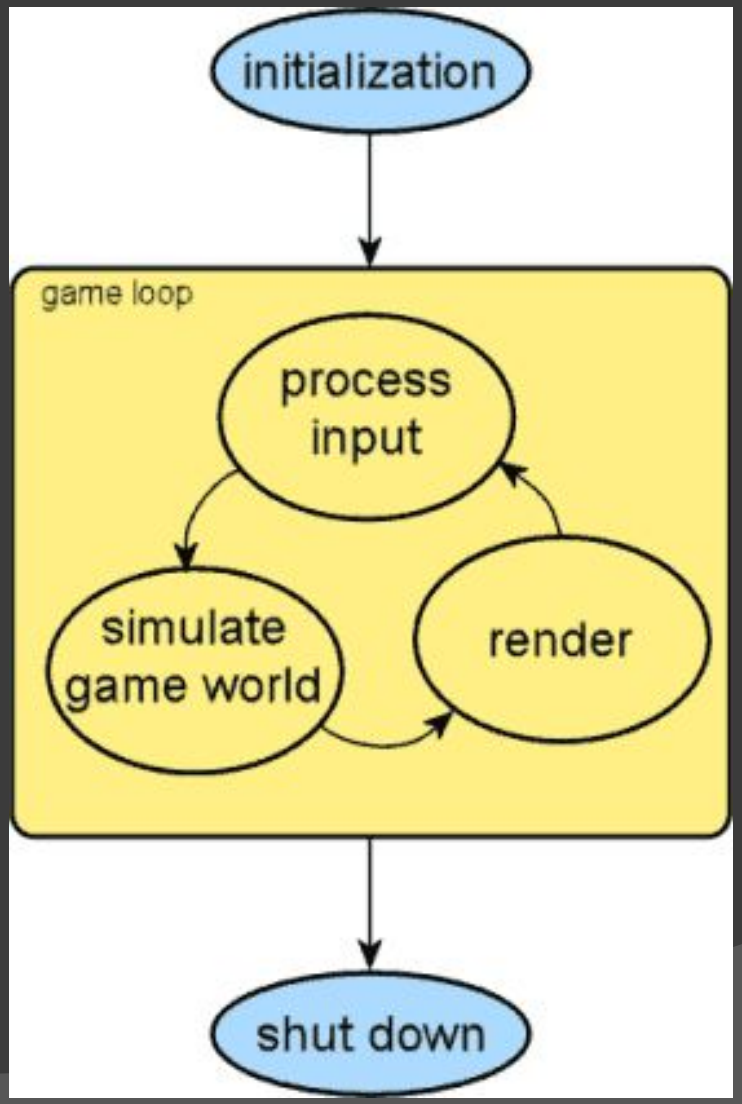
```
GLCanvas glcanvas = new GLCanvas(capabilities);
```

# The rendering / game loop...

# Rendering Loop / Game loop

- **Loop:** Graphics are repeatedly drawn on screen and interactive (frames)
- This is real-time rendering
- This is the type of rendering used in games
- This style of rendering contrasts offline rendering
  - where single images or frames are calculated over a long period of time
- Rendering loop should reach **50-60** Frames per Secundum (FPS)

# Rendering Loop / Game loop



# Rendering Loop / Game loop

**Initialization:**

- Choosing a GLProfile and configuring GLCapabilities for a rendering context
- Creating a window and GLContext through the GLAutoDrawable
- Making an animator thread
- Loading resources needed by program

**Process Input:**

- Listen for mouse and keyboard events
- Update user's view (often called a camera)

# Rendering Loop / Game loop

**Update (Simulate Game World):**

- Calculate geometry
- Rearrange data
- Perform computations

**Render:**

- Draw scene geometry from a particular view

**Shut Down:**

- Save persistent data
- Clean up resources on graphics card

# Animating the loop in JOGL

- JOGL provides some utility classes for animating our program.
- An Animator object can be created to ensure the display method of a GLAutoDrawable is repeatedly called.
- An FPSAnimator allows us make the framerate relatively consistent and can reduce the resource consumption of the program

```
FPSAnimator animator = new FPSAnimator(canvas, 60);
animator.start();
```

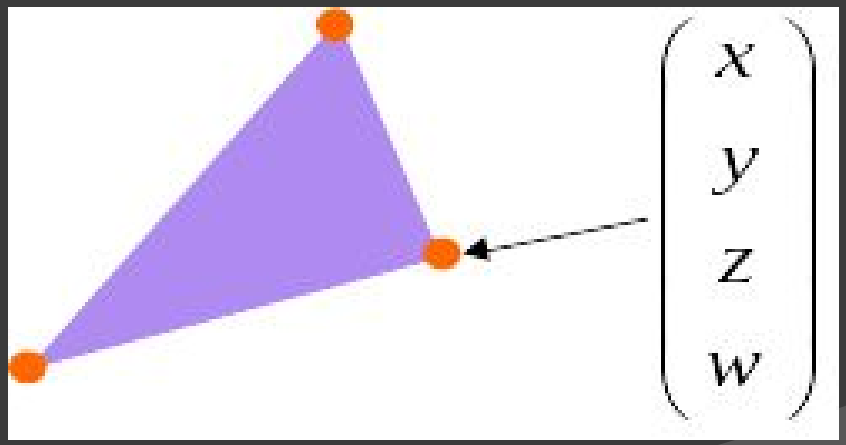
# Animating the loop in JOGL

- The animator is attached to the GLCanvas and asked to render at roughly 60 frames per second
- The *display method* will be called approximately every 17 ms (1000 / 60)

# The Basics of OpenGL...

# Geometry Basics

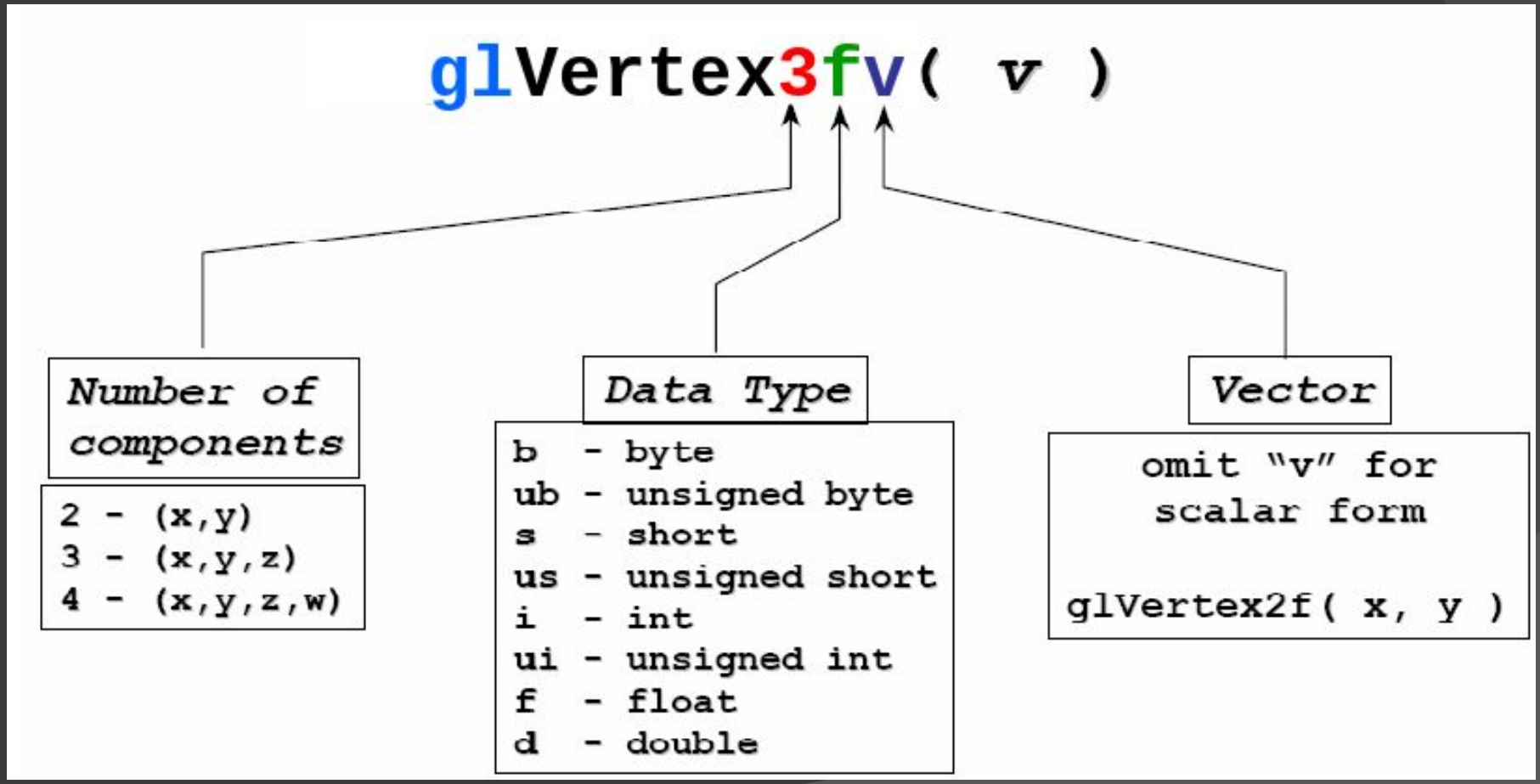
- **Geometric objects are represented using vertices**
- A vertex is a collection of generic attributes
  - positional coordinates
  - colors
  - texture coordinates
  - any other data associated with that point in space



# Basics

- **OpenGL is not object oriented**
  - so that there are multiple functions for a given logical function
  - E.g.:
    - glVertex3f
    - glVertex2i
    - glVertex3dv
- **OpenGL is a state machine:**
  - We put it into various states (or modes) that then remain in effect until you change them
  - E.g: colors, current viewing and projection transformations, line and polygon stipple patterns, etc
  - Every state has a default value

# Geometry Basics



# Rendering Loop

- Every rendering loop has three phases:
  - **Start part:** usually reinitialises the view and states
    - reset events, clears screen, etc. E.g.
      glClear( GL_COLOR_BUFFER_BIT );
  - **Drawing part:** draw geometry

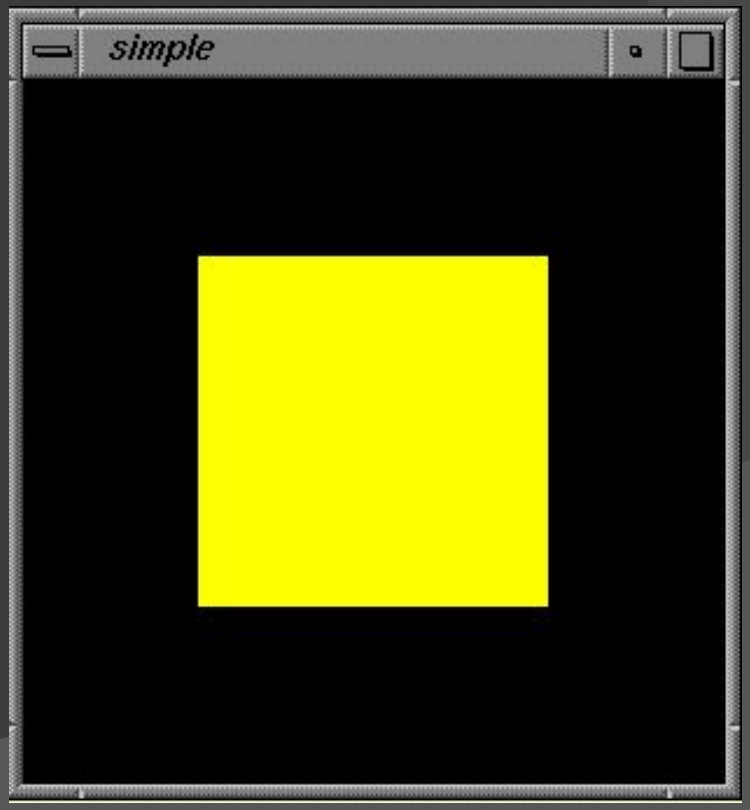
```
glBegin( XXXX );
    .....
glEnd();
```

- **End part:** closes the rendering process
  - OpenGL should be informed about the end of rendering

  Command: **glFlush();**

# Basic Example

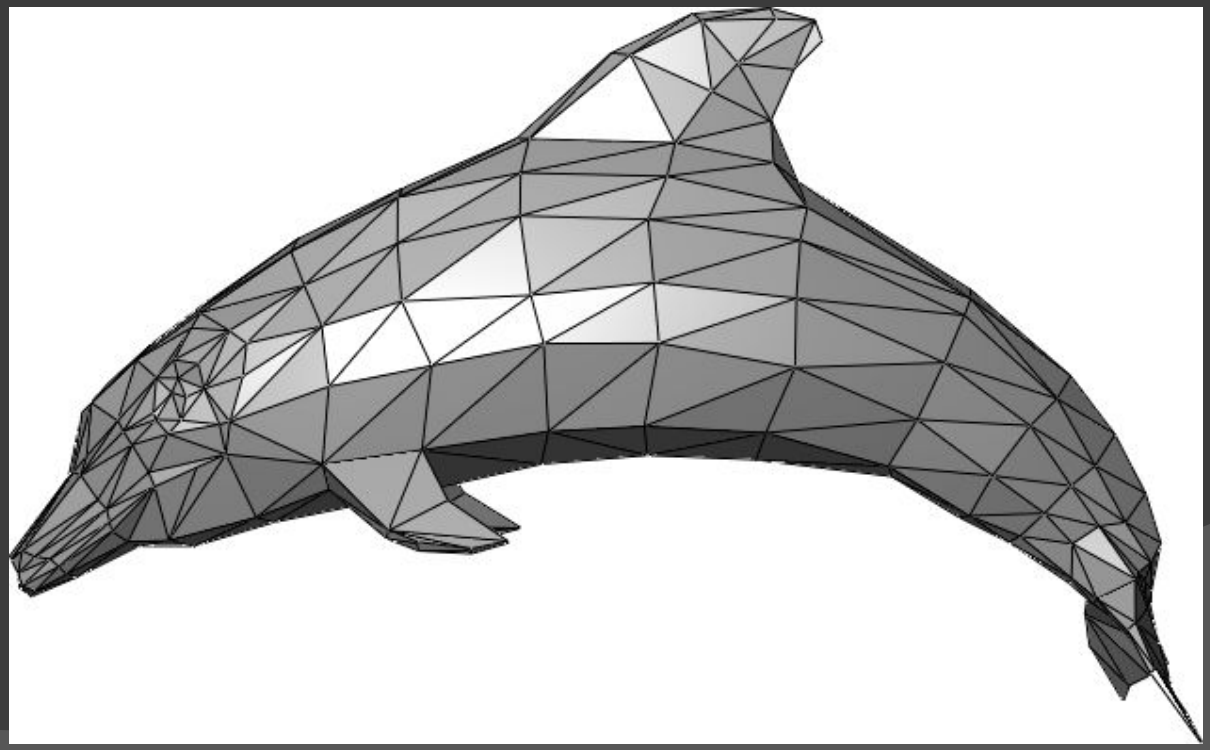
```
void Display()  {

glClear(GL_COLOR_BUFFER_BIT);
  glColor4f(1,1,0,1);
  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();
}
```



# Vertices and Primitives...

# Vertices and Primitives

- Geometry object are built from primitives
- OpenGL support many primitive types
- Game models are usually based on Triangles

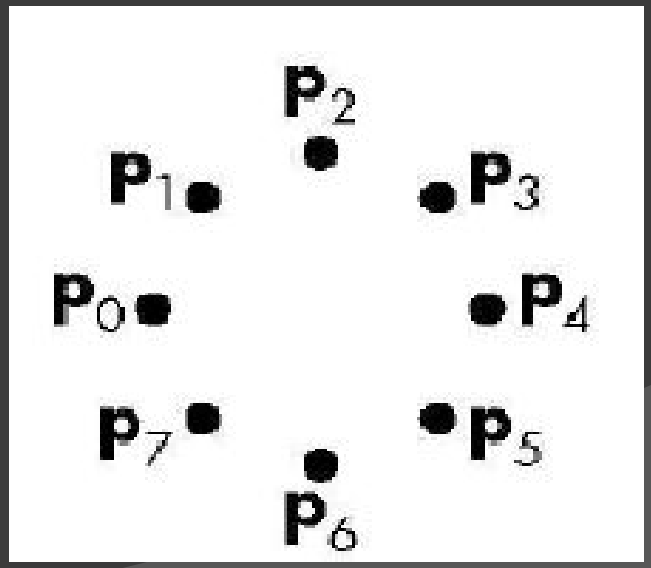


# Vertices and Primitives

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

Example:

```
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();
```

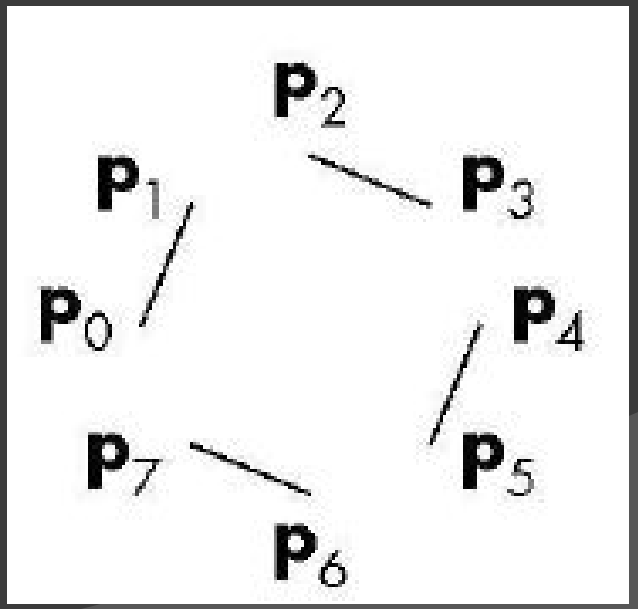


# Vertices and Primitives

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

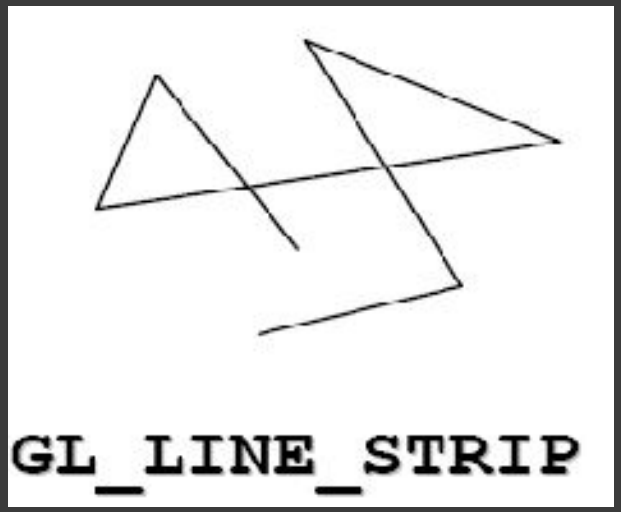
Example:

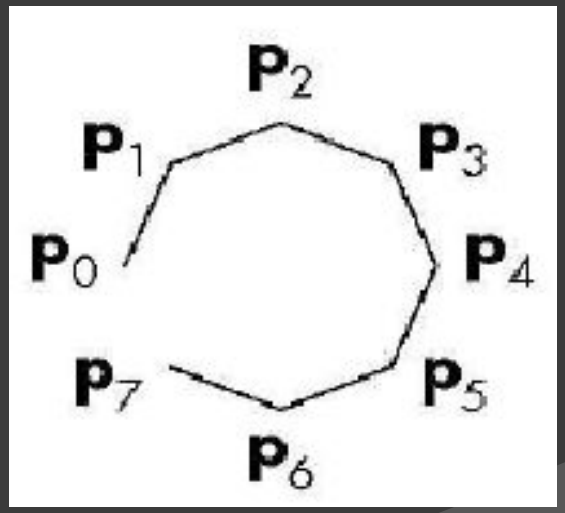
```
glBegin(GL_LINES);
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```



# Vertices and Primitives

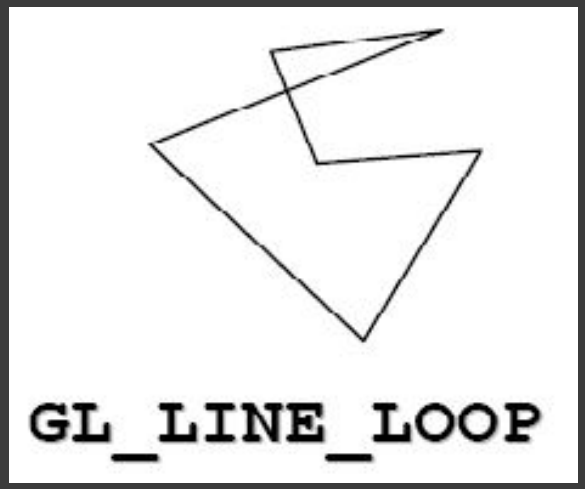
- **Line Strip, GL_LINE_STRIP**
  - series of connected line segments

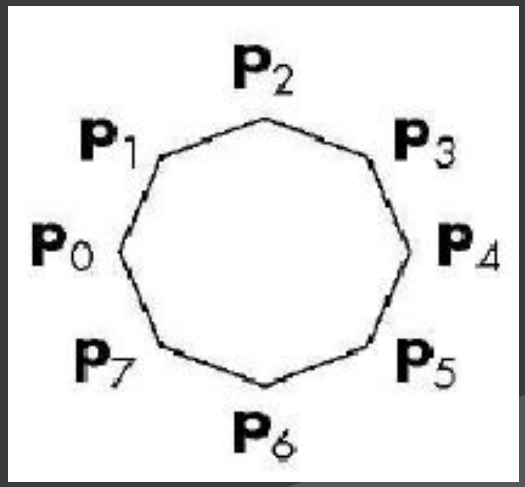


# Vertices and Primitives

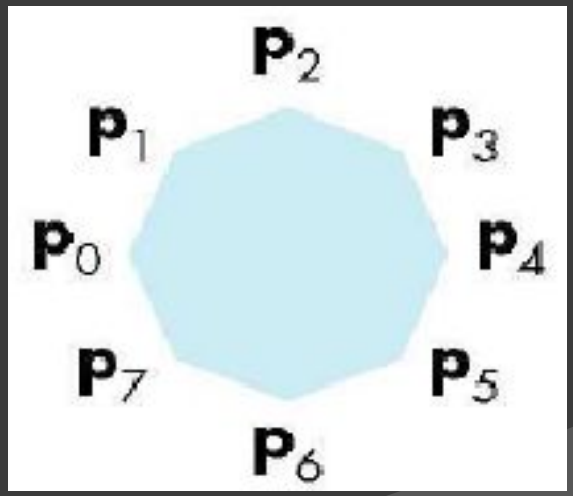
- **Line Loop, GL_LINE_LOOP**
  - Line strip with a segment added between last and first vertices



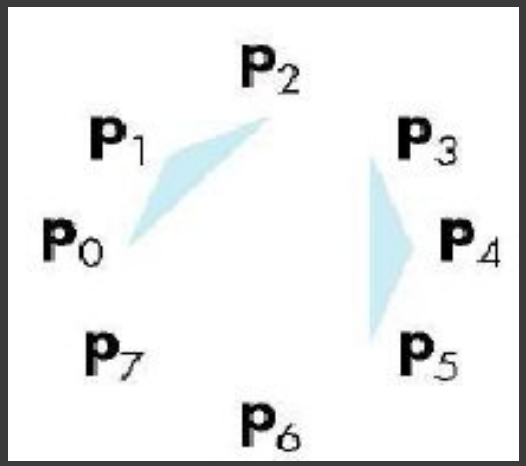
# Vertices and Primitives

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

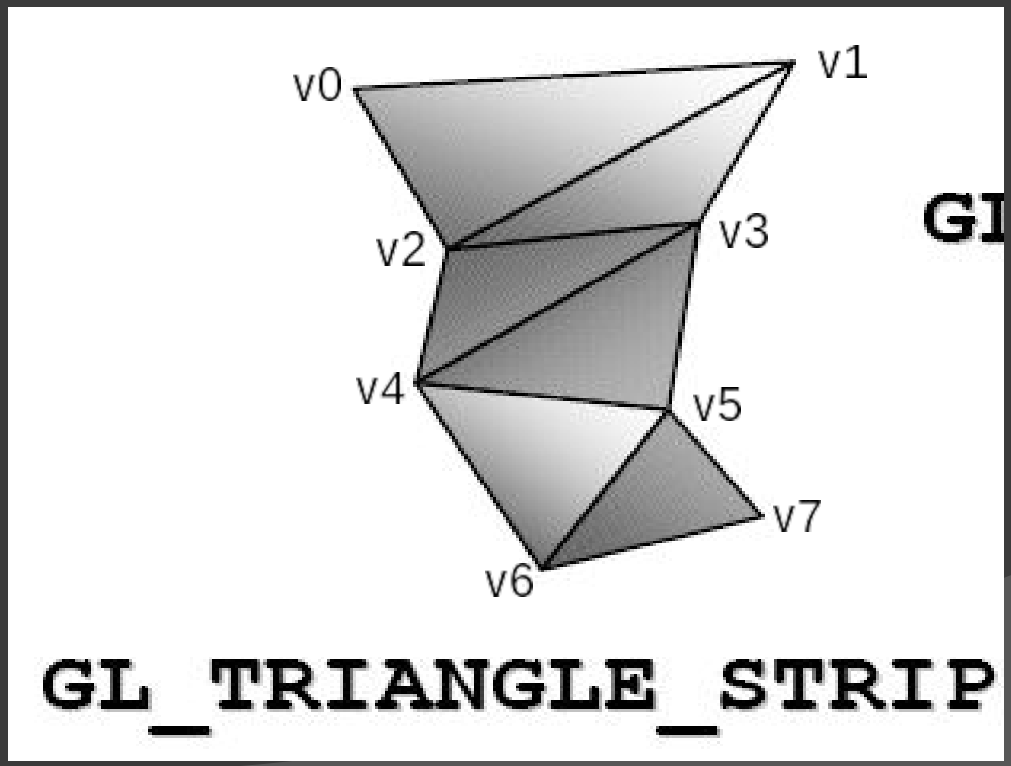
# Vertices and Primitives

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

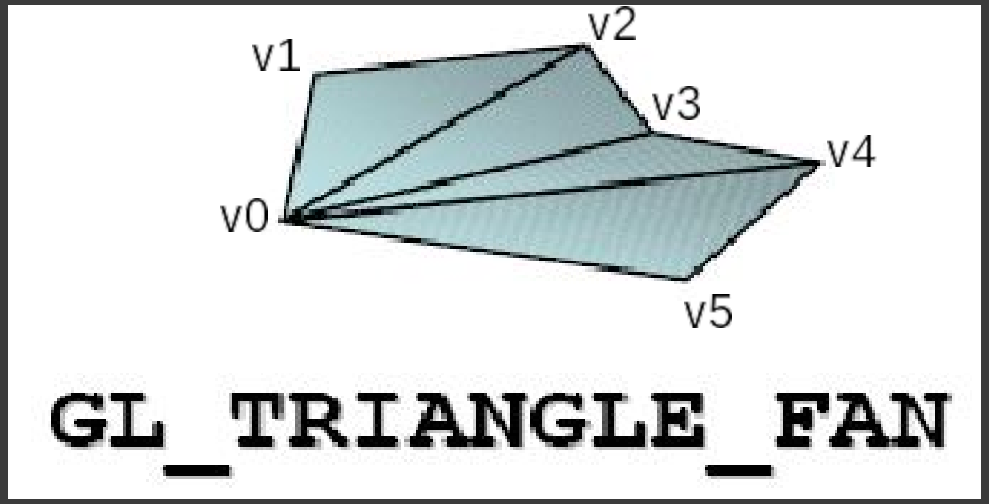
# Vertices and Primitives

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



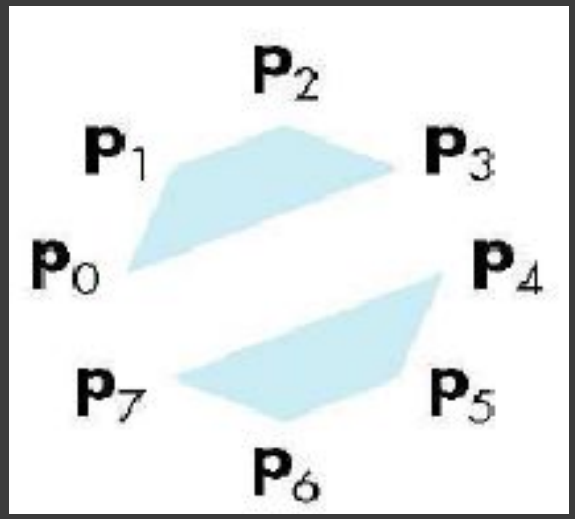
# Vertices and Primitives

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



# 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)

# Transformations…

# Transformations is OpenGL

- **Modeling transformation**
  - Refer to the transformation of models (i.e., the scenes, or objects)
- **Viewing transformation**
  - Refer to the transformation on the camera
- **Projection transformation**
  - Refer to the transformation from scene to image

GAME OVER