GRAPHICS PIPELINE

OUTLINE

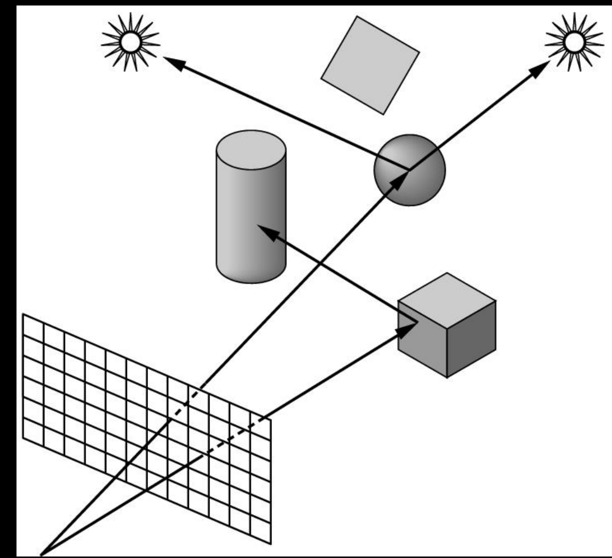
- Learn the basic design of a graphics system
- Introduce pipeline architecture
- Examine software components for a graphics system

IMAGE FORMATION REVISITED

- Can we mimic the synthetic camera model to design graphics hardware software?
- Application Programmer Interface (API)
  - Need only specify
    - Objects
    - Materials
    - Viewer
    - Lights
- But how is the API implemented?

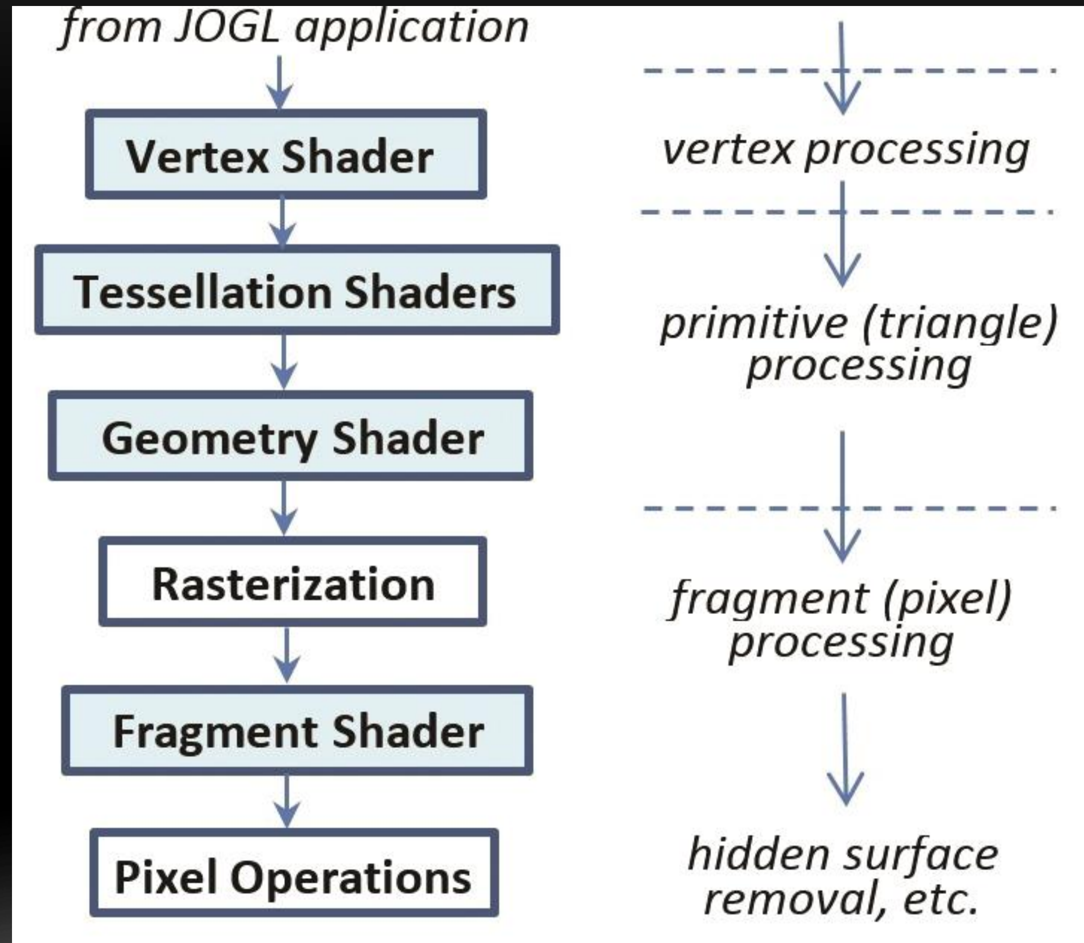
PHYSICAL APPROACHES

- **Ray tracing**: follow rays of light from center of projection until they either are absorbed by objects or go off to infinity
  - Can handle global effects
    - Multiple reflections
    - Translucent objects
  - Slow
  - Must have whole data base available at all times



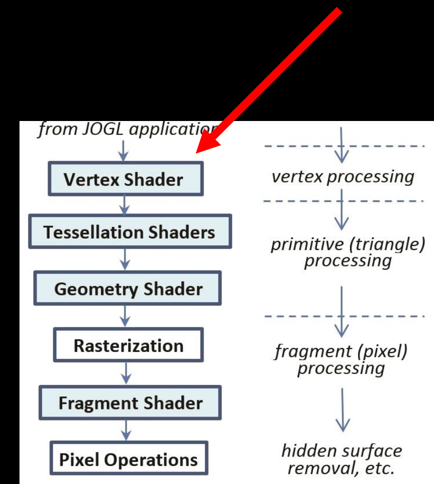
PRACTICAL APPROACH

- Process objects one at a time in the order they are generated by the application
    - Can consider only local lighting
- Pipeline architecture
- All steps can be implemented in hardware on the graphics card



VERTEX PROCESSING

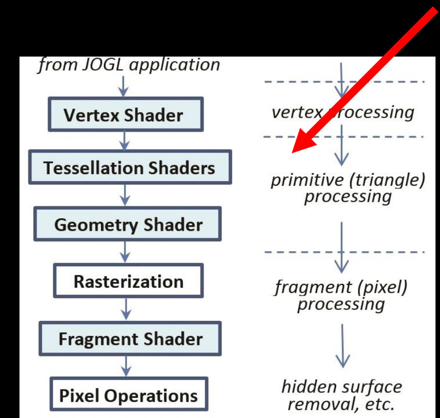
- Much of the work in the pipeline is in converting object representations from one coordinate system to another
  - Object coordinates
  - Camera (eye) coordinates
  - Screen coordinates
- Every change of coordinates is equivalent to a matrix transformation
- Vertex processor also computes vertex colors



PRIMITIVE ASSEMBLY

Vertices must be collected into geometric objects before clipping and rasterization can take place

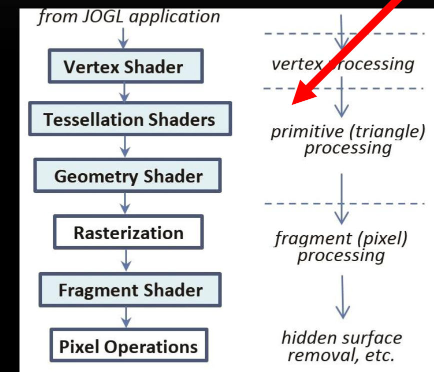
- Line segments
- Polygons
- Curves and surfaces

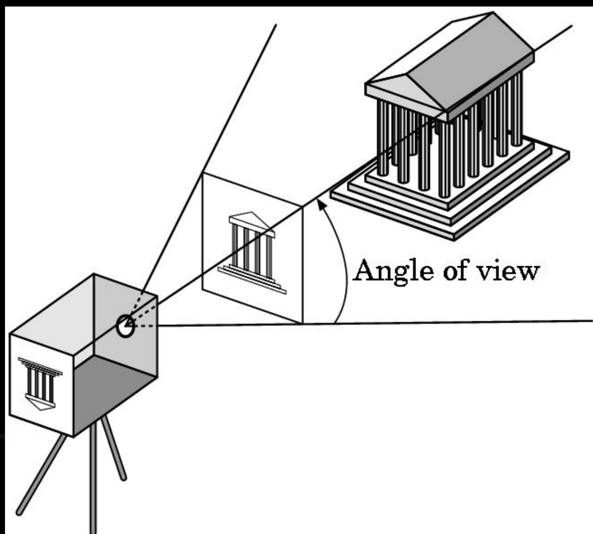


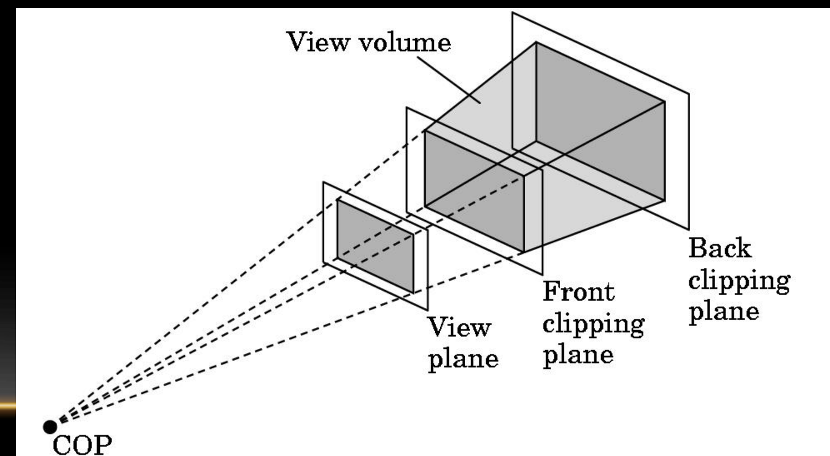
CLIPPING

Just as a real camera cannot “see” the whole world, the virtual camera can only see part of the world or object space

- Objects that are not within this volume are said to be *clipped* out of the scene

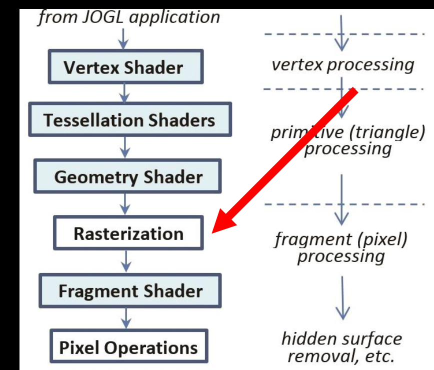






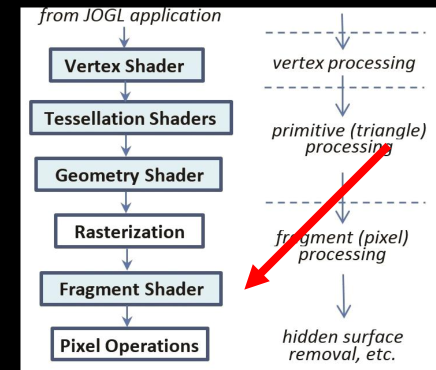
RASTERIZATION

- If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors
- Rasterizer produces a set of fragments for each object
- Fragments are “potential pixels”
  - Have a location in frame buffer
  - Color and depth attributes
- Vertex attributes are interpolated over objects by the rasterizer



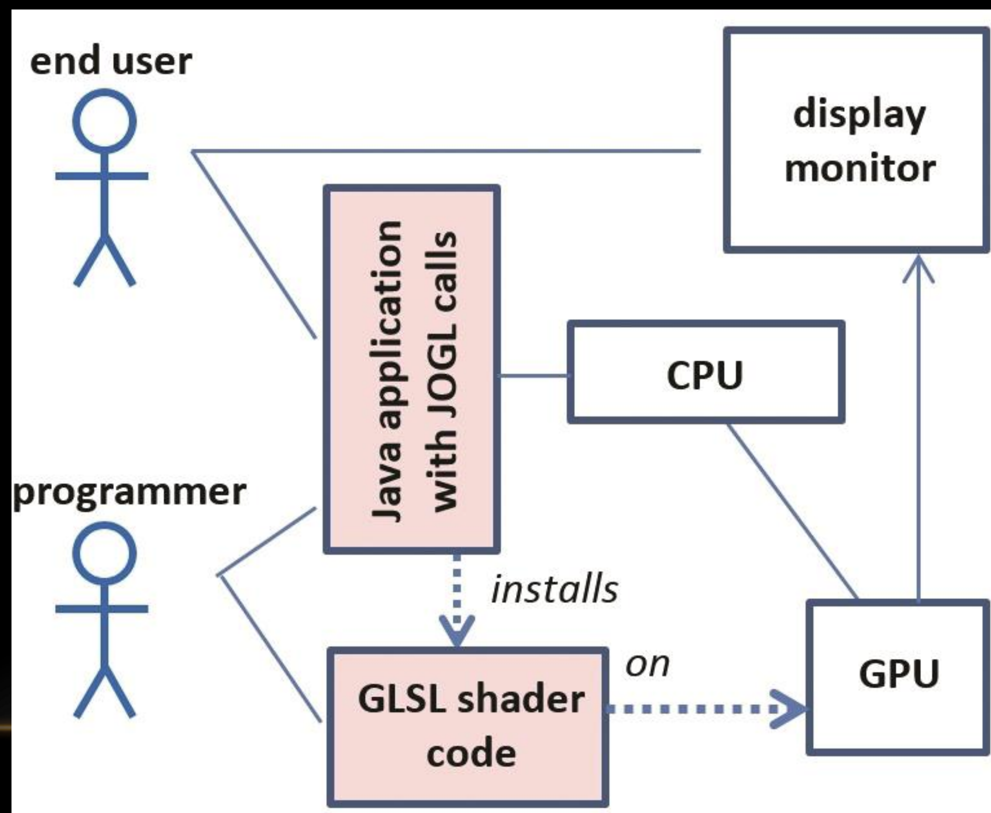
FRAGMENT PROCESSING

- Fragments are processed to determine the color of the corresponding pixel in the frame buffer
- Colors can be determined by texture mapping or interpolation of vertex colors
- Fragments may be blocked by other fragments closer to the camera
  - Hidden-surface removal



THE PROGRAMMER'S INTERFACE

- Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)



API CONTENTS

- Functions that specify what we need to form an image
  - Objects
  - Viewer
  - Light Source(s)
  - Materials
- Other information
  - Input from devices such as mouse and keyboard
  - Capabilities of system

OBJECT SPECIFICATION

- Most APIs support a limited set of primitives including
  - Points (0D object)
  - Line segments (1D objects)
  - Polygons (2D objects)
    - Triangles!!
  - Some curves and surfaces
    - Quadrics
    - Parametric polynomials

- All are defined through locations in space or *vertices*

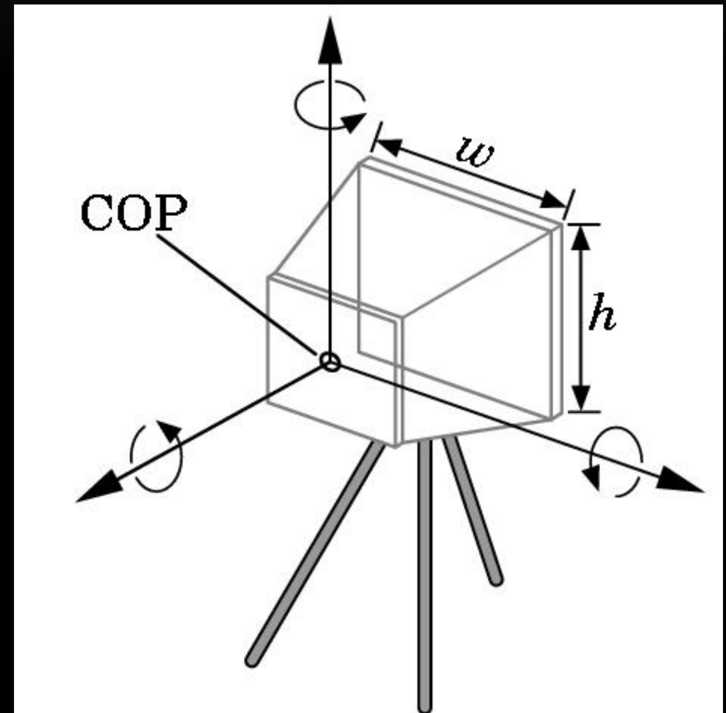
EXAMPLE

- Put geometric data in an array
- Send array to GPU
- Tell GPU to render as triangle

```
vec3 points[3];
points[0] = vec3(0.0, 0.0, 0.0);
points[1] = vec3(0.0, 1.0, 0.0);
points[2] = vec3(0.0, 0.0, 1.0);
```

CAMERA SPECIFICATION

- Position of center of lens
- Orientation
- Film size
- Orientation of film plane



LIGHTS AND MATERIALS

- Types of lights
  - Point sources vs distributed sources
  - Spot lights
  - Near and far sources
  - Color properties
- Material properties
  - Absorption: color properties
  - Scattering
    - Diffuse
    - Specular

CODE EXPLANATION – SHADERS: DRAWING A POINT

- Look at code in the order of execution
  - Slides show code “out of order”
- Import libraries
- Definition of main() and other methods required by interface
- Constructor
- init() and display()
- Creating GLSL code for the GPU

IMPORT LIBRARIES

```
import javax.swing.*;
import static com.jogamp.opengl.GL4.*;
import com.jogamp.opengl.*;
import com.jogamp.opengl.awt.GLCanvas;
import com.jogamp.opengl.GLContext;
```

MAIN METHOD AND INTERFACE METHODS

```
public static void main(String[] args)
   {
      new Code();
   }

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

public void dispose(GLAutoDrawable drawable)
   {}
```

CLASS DECLARATION, CONSTRUCTOR

```
public class Code extends JFrame implements GLEventListener
  {
    private GLCanvas myCanvas;
    private int rendering_program;
    private int vao[] = new int[1];

    public Code()
    {
      setTitle("Chapter2 - program2");
      setSize(600, 400);
      myCanvas = new GLCanvas();
      myCanvas.addGLEventListener(this);
      getContentPane().add(myCanvas);
      setVisible(true);
    }
```

INIT AND DISPLAY – CALLED WHEN SETVISIBLE MADE TRUE

```
public void display(GLAutoDrawable drawable)
  {
     GL4 gl = (GL4) GLContext.getCurrentGL();
     gl.glUseProgram(rendering_program);
     gl.glPointSize(30.0f); gl.glDrawArrays(GL_POINTS,0,1);
  }

public void init(GLAutoDrawable drawable)
  {
     GL4 gl = (GL4) GLContext.getCurrentGL();
     rendering_program = createShaderProgram();
     gl.glGenVertexArrays(vao.length, vao, 0);
     gl.glBindVertexArray(vao[0]);
  }
```

GLSL CODE

```
private int createShaderProgram()
  {
     GL4 gl = (GL4) GLContext.getCurrentGL();
     String vshaderSource[] =
     {
        "#version 430 \n",
        "void main(void) \n",
        "{ gl_Position = vec4(0.0, 0.0, 0.0, 1.0); } \n"
     };
     String fshaderSource[] =
     {
        "#version 430 \n",
        "out vec4 color; \n",
        "void main(void) \n",
        "{ color = vec4(0.0, 0.0, 1.0, 1.0); } \n"
     };

…
```

COMPILING, ATTACHING, LINKING GLSL CODE

```
...
   int vShader = gl.glCreateShader(GL_VERTEX_SHADER);
   gl.glShaderSource(vShader, 3, vshaderSource, null, 0);
   gl.glCompileShader(vShader);

   int fShader = gl.glCreateShader(GL_FRAGMENT_SHADER);
   gl.glShaderSource(fShader, 4, fshaderSource, null, 0);
   gl.glCompileShader(fShader);

   int vfprogram = gl.glCreateProgram();
   gl.glAttachShader(vfprogram, vShader);
   gl.glAttachShader(vfprogram, fShader);
   gl.glLinkProgram(vfprogram);

   gl.glDeleteShader(vShader);
   gl.glDeleteShader(fShader);
   return vfprogram;
}
```

SUMMARY

- Learn the basic design of a graphics system
- Introduce pipeline architecture
- Examine software components for a graphics system