

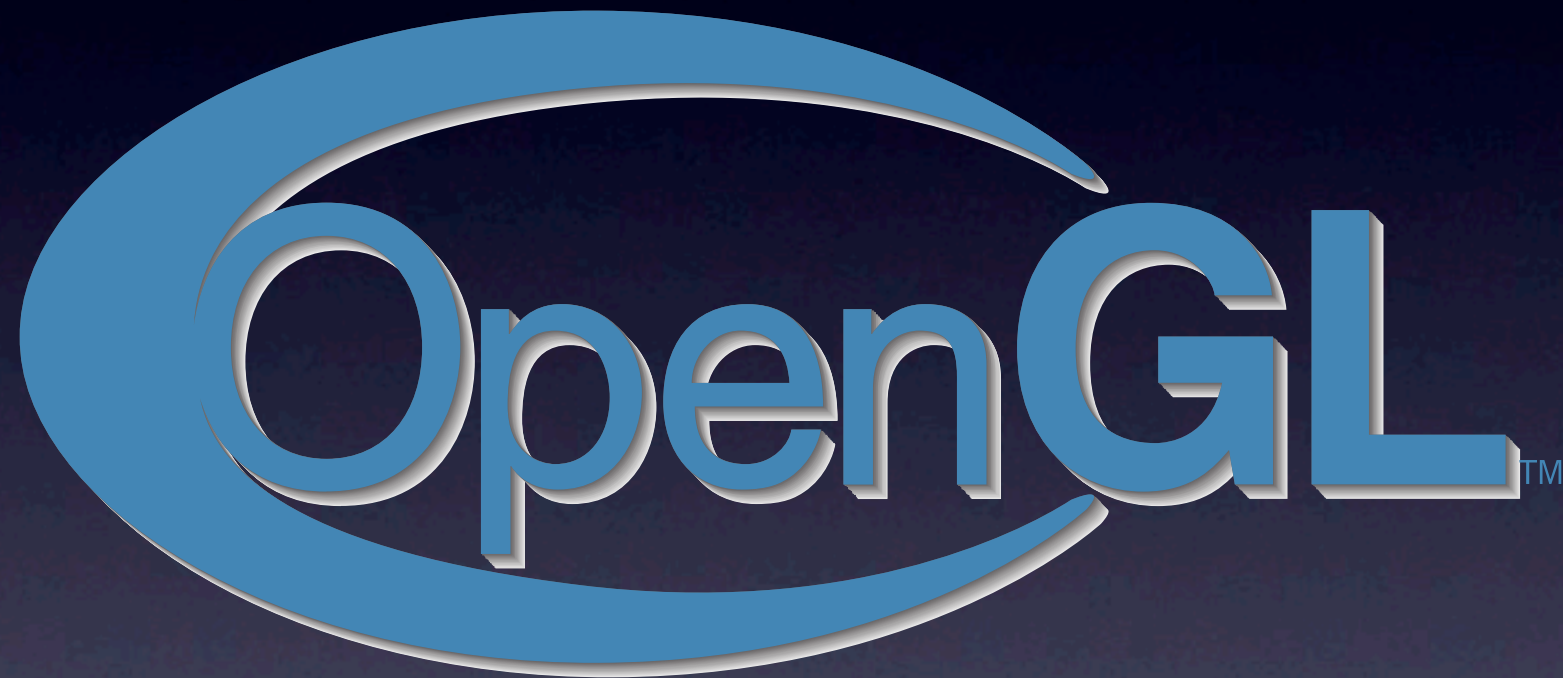
The Care and Feeding of OpenGL 3.2 Core Profile

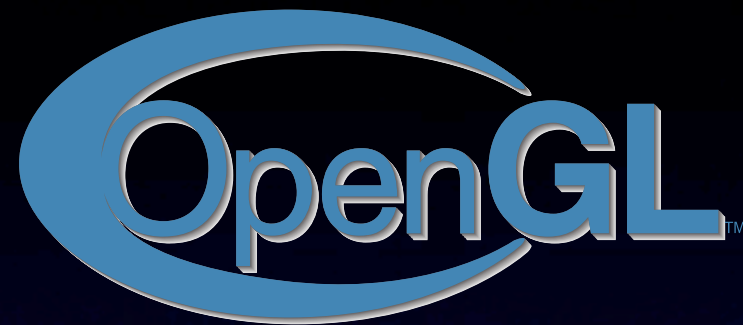
Concepts, Best Practices, and Pitfalls



Cocoa Heads Aachen, 25.08.2011
Thorsten Karrer and Moritz Wittenhagen

```
GLdouble size = 1.5;  
glutSolidTeapot(size);  
glFlush();
```





Server

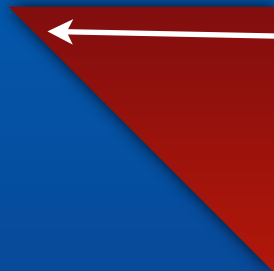
(actually your graphics board
and its driver)

Client

(your application)



Server



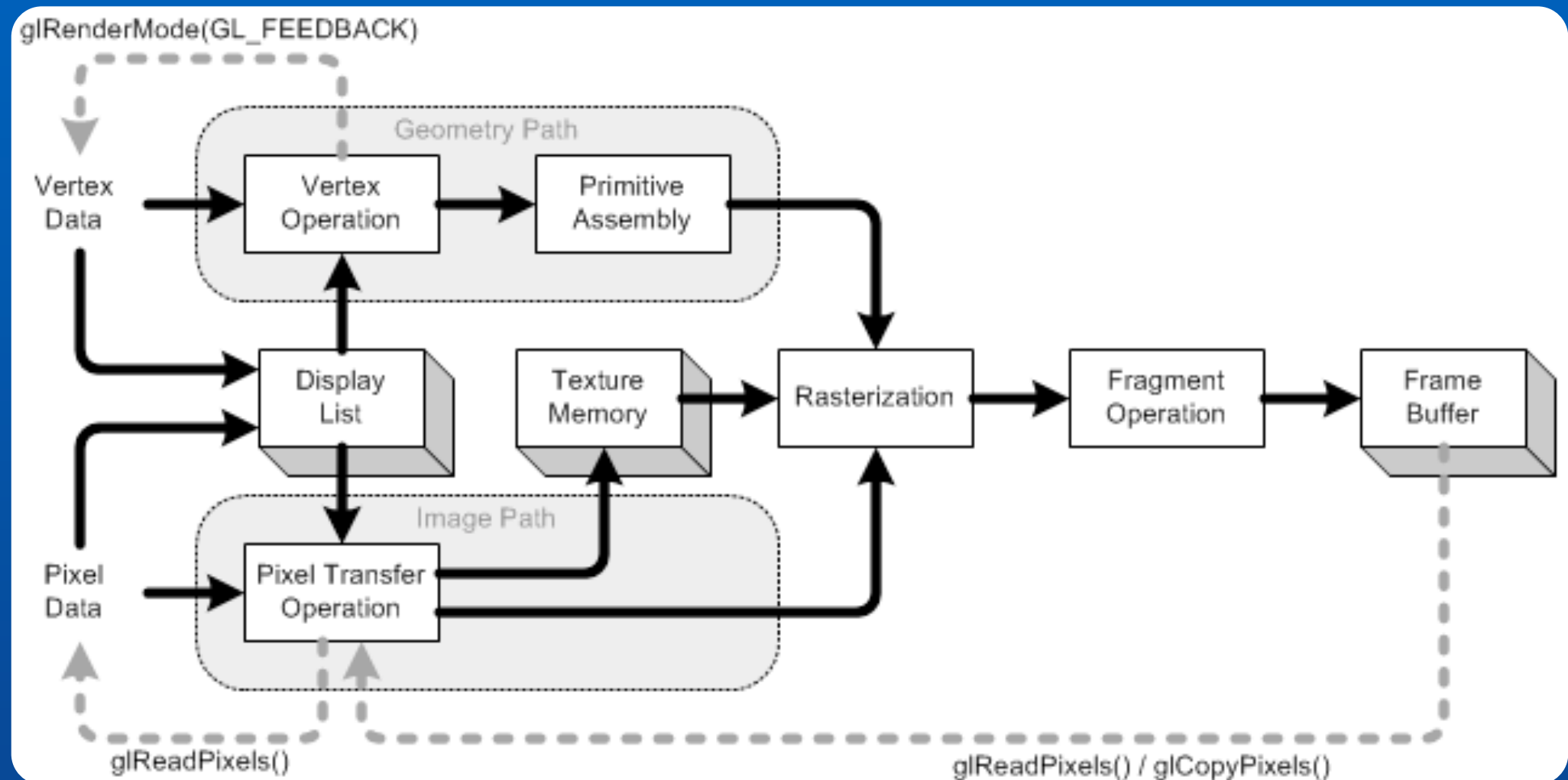
Client

```
glBegin(GL_TRIANGLE);  
glVertex3f(.5, .5, .0);  
glVertex3f(.0, .5, .0);  
glVertex3f(.5, .0, .0);  
glEnd();
```

Client

Here lives your model.

Server—Rendering Pipeline



http://www.songho.ca/opengl/gl_pipeline.html

This is a fully programmable pipeline.

What does that mean for me?

- **No** more immediate mode rendering.
- **No** more client-side storage.
- **No** pre-defined vertex attributes.
- **No** matrix stack.
- **No** light sources.
- **None** of the fixed function pipeline features!

How will we draw?

- **VBOs** for storing generic vertex attributes.
- **VAOs** for telling OpenGL what's in our VBOs.
- **TBOs** for 1D textures that use buffer storage.
- **FBOs** for off-screen rendering.
- **Shaders** to run our homebrewn pipeline!
- A lot of **helper structs and functions** to write...

Vertex Buffer Objects (VBOs)

- With target `GL_ARRAY_BUFFER`:
 - Stores everything that could be a vertex attribute
 - Position, Color, Normal, TexCoords...
 - `GL_BYTE`, `GL_UNSIGNED_BYTE`, `GL_SHORT`,
`GL_UNSIGNED_SHORT`, `GL_INT`, `GL_UNSIGNED_INT`,
`GL_FLOAT`, `GL_DOUBLE`
- With target `GL_ELEMENT_ARRAY_BUFFER`:
 - Stores vertex indices (some call it IBO)

Vertex Buffer Objects (VBOs)

```

// Contains our vertex positions, normals, colors
GLfloat vertex[300] = {...};

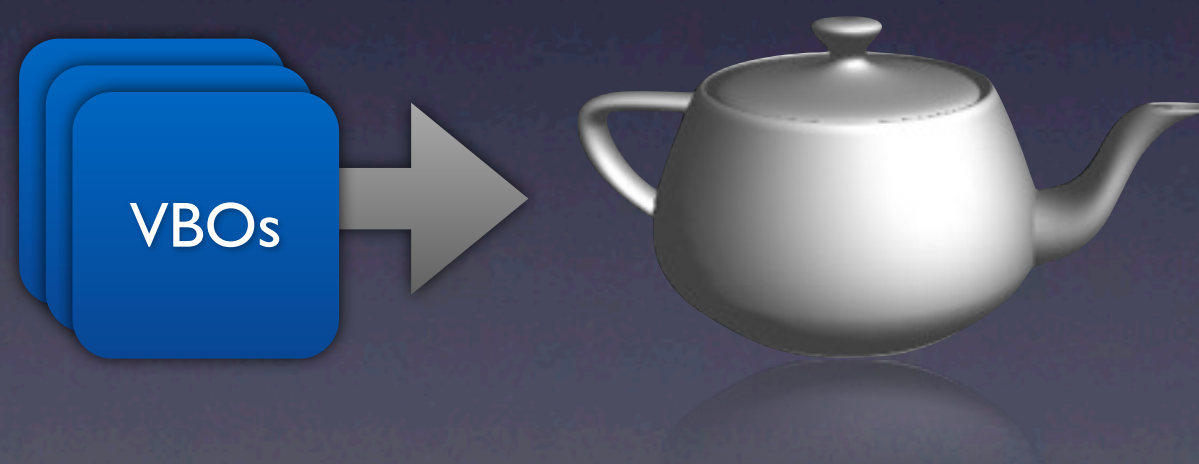
GLuint myVboId;
glGenBuffers(1, &myVboId);

glBindBuffer(GL_ARRAY_BUFFER, myVboId);

glBufferData(GL_ARRAY_BUFFER, 300 * sizeof(GLfloat),
             vertex, GL_STATIC_DRAW);
  
```

Our first Assumption

```
glBindBuffer(GL_ARRAY_BUFFER, myVboId);  
glDrawArrays(GL_TRIANGLES, 0, 300);  
glFlush();
```





Vertex Shader

```
#version 150
in vec3 in_position;

void main() {
    gl_Position = vec4(in_position, 1.0);
}
```

Fragment Shader

```
#version 150
out vec4 fragColor;

void main() {
    fragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
```

Shader Construction

```

NSURL *url = [[NSBundle mainBundle] URLForResource:@"vShader"
withExtension:@"vs"];
const char *vShaderSrc = [[NSString stringWithContentsOfURL:url
encoding:NSUTF8StringEncoding error:NULL] UTF8String];

GLuint vertexShaderID = glCreateShader(GL_VERTEX_SHADER);

glShaderSource(vertexShaderID, 1, &vShaderSrc, NULL);
glCompileShader(vertexShaderID);

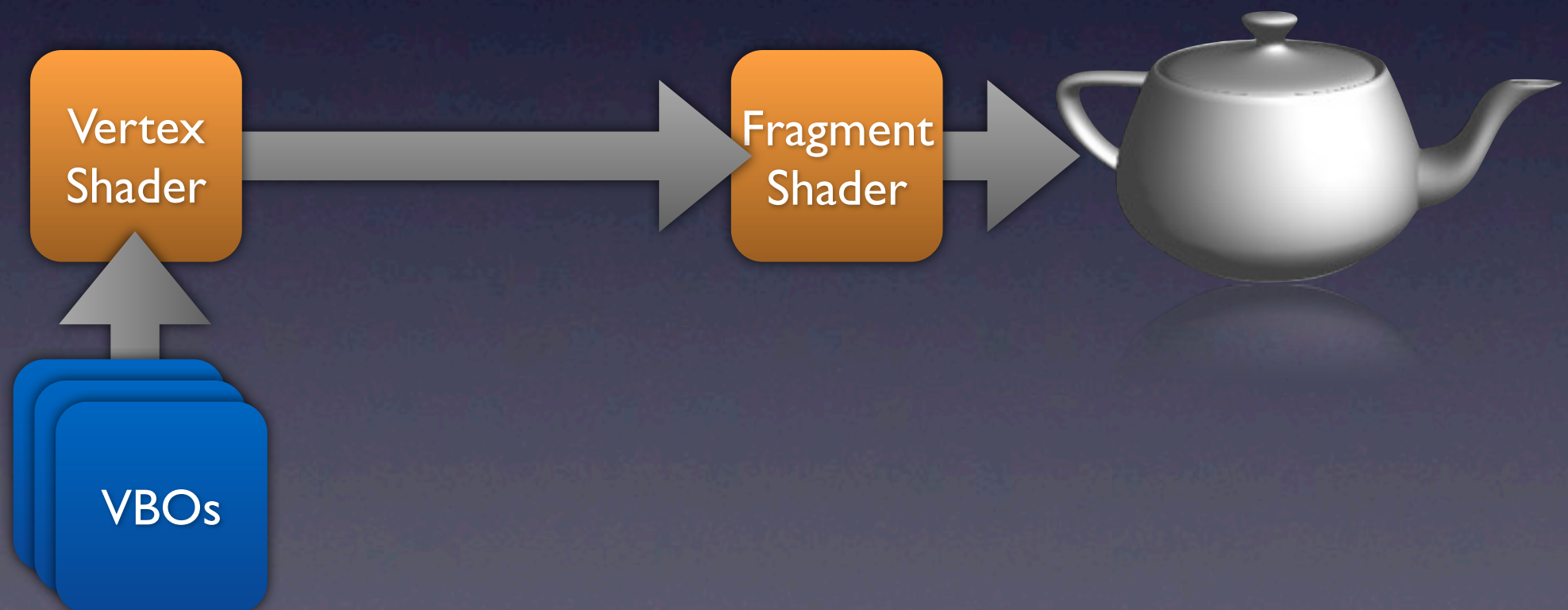
// [...same for fragment shader...]

GLuint shaderProgramId = glCreateProgram();

glAttachShader(shaderProgramId, vertexShaderID);
glAttachShader(shaderProgramId, fragmentShaderID);
  
```


Our second Assumption

```
glBindBuffer(GL_ARRAY_BUFFER, myVboId);  
glUseProgram(shaderProgramId);  
glDrawArrays(GL_TRIANGLES, 0, 300);  
glFlush();
```





VBO

```
GLfloat  
GLfloat  
GLfloat  
GLfloat  
GLfloat  
GLfloat  
GLfloat  
GLfloat  
GLfloat  
GLfloat  
GLfloat  
[...]
```

Hmmm...?

```
GLfloat vertex[0].position.x  
GLfloat vertex[0].position.y  
GLfloat vertex[0].position.z  
GLfloat vertex[0].color.r  
GLfloat vertex[0].color.g  
GLfloat vertex[0].color.b  
GLfloat vertex[1].position.x  
GLfloat vertex[1].position.y  
GLfloat vertex[1].position.z  
GLfloat vertex[1].color.r  
GLfloat vertex[1].color.g  
GLfloat vertex[1].color.b  
[...]
```


Vertex Array Objects (VAOs)

```

struct VertexArrayObject {
    BufferObject      *pElementArrayBufferObject = NULL; // IBO
    VertexAttribute  attributes[GL_MAX_VERTEX_ATTRIB]; // below
}

struct VertexAttribute {
    bool              bIsEnabled                = GL_FALSE;
    int               iSize                     = 4;      // components
    unsigned int      iStride                   = 0;
    VertexAttribType  eType                     = GL_FLOAT;
    bool              bIsNormalized             = GL_FALSE;
    bool              bIsIntegral               = GL_FALSE;
    void              *pBufferObjectOffset     = 0;      // location
    BufferObject       *pBufferObj              = 0;      // VBO
};
  
```

Vertex Array Objects (VAOs)

```

#define ATTR_POS 0
#define ATTR_NOR 1
#define DENSE 0

glGenVertexArrays(1, &myVaoId);
glBindVertexArray(myVaoId);

glBindBuffer(GL_ARRAY_BUFFER, myPosVboId); // VB0 for positions
glVertexAttribPointer(ATTR_POS, 3, GL_FLOAT, GL_FALSE, DENSE, 0);

glBindBuffer(GL_ARRAY_BUFFER, myNorVboId); // VB0 for normals
glVertexAttribPointer(ATTR_NOR, 4, GL_FLOAT, GL_FALSE, DENSE, 0);

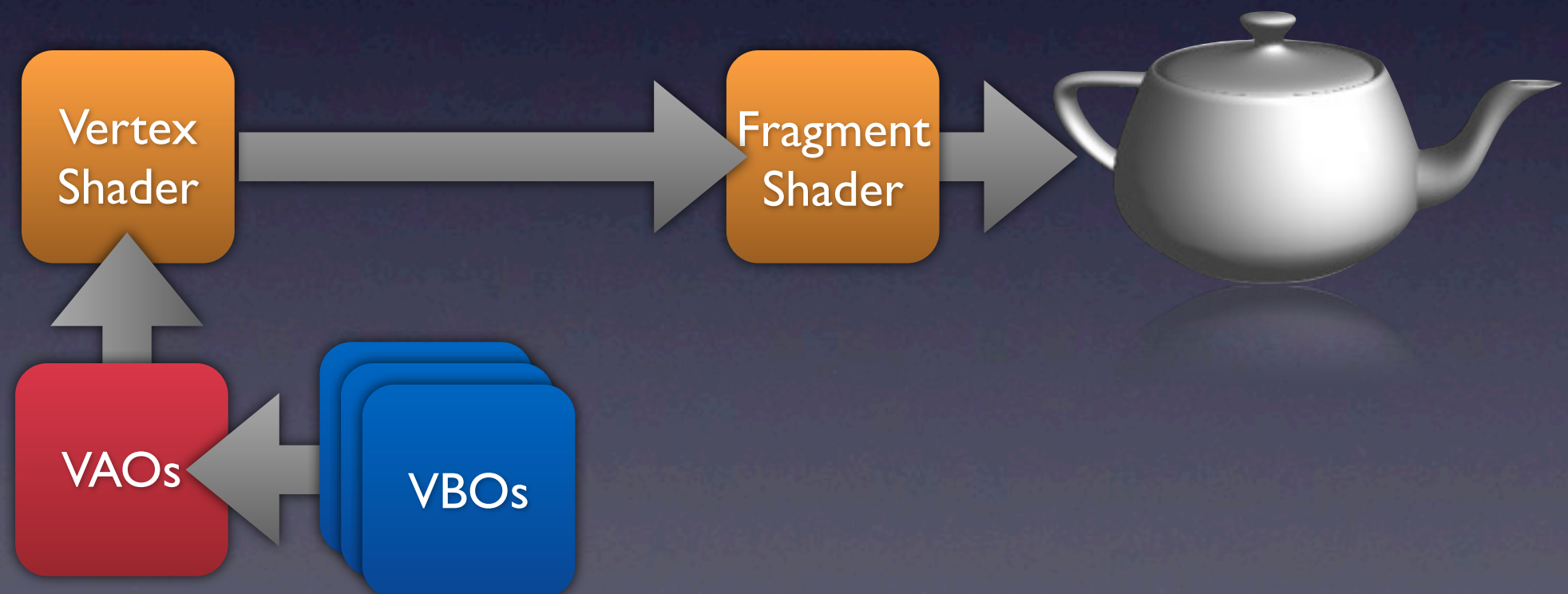
glEnableVertexAttribArray(ATTR_POS);
glEnableVertexAttribArray(ATTR_NOR);
  
```

Shader Construction

```
// [...compile vertex and fragment shaders like above...]  
  
GLuint shaderProgramId = glCreateProgram();  
  
glAttachShader(shaderProgramId, vertexShaderID);  
glAttachShader(shaderProgramId, fragmentShaderID);  
  
glBindAttribLocation(shaderProgramId, ATTR_POS, "in_position");  
glBindAttribLocation(shaderProgramId, ATTR_NOR, "in_normal");  
  
glLinkProgram(shaderProgramId);
```


Our third Assumption

```
glBindVertexArray(myVaoId);  
glDrawArrays(GL_TRIANGLES, 0, 300);  
glFlush();
```





Vertex Shader

```

#version 150
in vec3 in_position;
in vec3 in_normal;

vec3 lightPos = vec3(0.0, 1.0, -2.0);

out vec3 normal;
out vec3 lightDir;

void main() {
    vec4 worldPos = vec4(in_position, 1.0);
    gl_Position = worldPos;

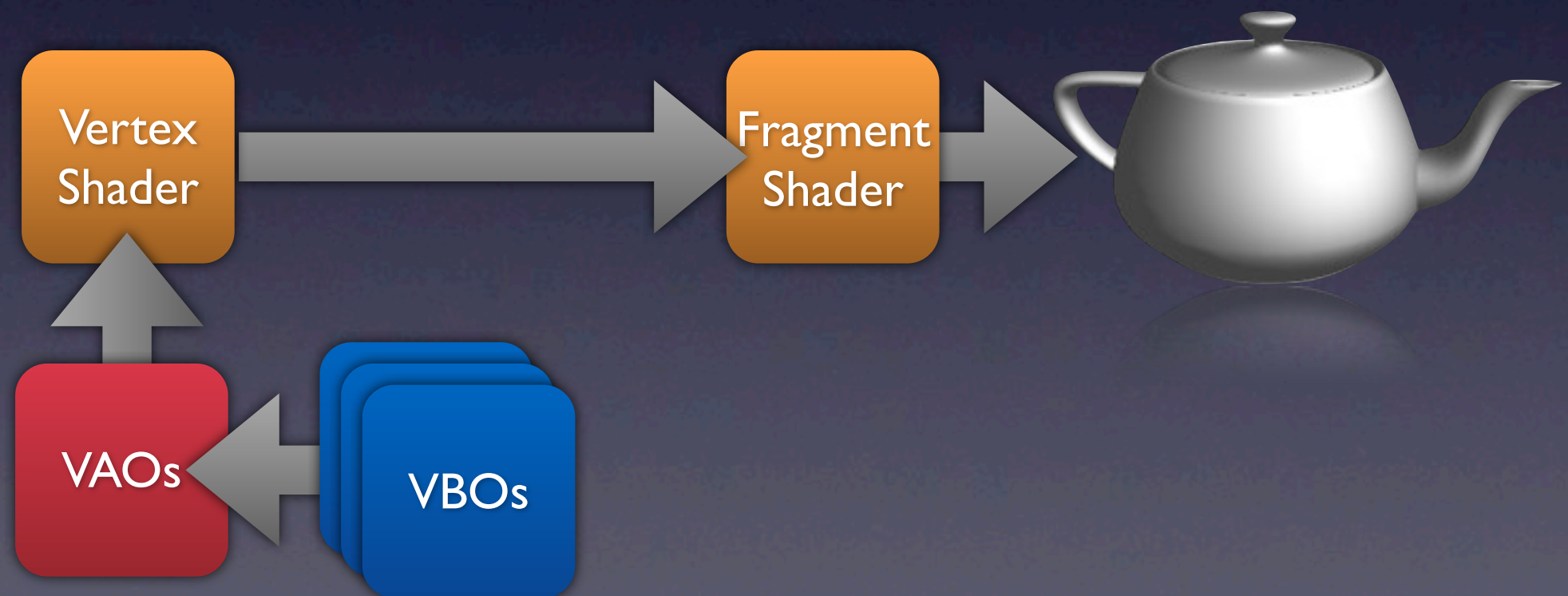
    normal = normalize(in_normal);
    lightDir = normalize(lightPos - worldPos.xyz);
}
  
```


Fragment Shader

```
#version 150
in vec3 normal;
in vec3 lightDir;
out vec4 fragColor;

void main() {
    float lambert = max(dot(normal, lightDir), 0.0);
    float ambient = 0.2;
    vec3 white = vec3(1.0, 1.0, 1.0);
    vec3 c = white * (lambert + ambient);
    fragColor = clamp(vec4(c, 1.0), 0.0, 1.0);
}
```

```
glBindVertexArray(myVaoId);  
glDrawArrays(GL_TRIANGLES, 0, 300);  
glFlush();
```





Vertex Shader (partial)

```
uniform mat4 modelview;  
uniform vec4 lightPos;  
void main() {  
    worldPos = modelview * vec4(in_position, 1.0);  
    lightDir = normalize(modelview * lightPos - worldPos).xyz;  
}
```

Supplying Client-Side Data

```
GLint lightPosId = glGetUniformLocation(shaderProgramId,  
"lightPos");  
  
GLfloat myLightPos[4] = {3.0, 1.0, -1.0, 1.0};  
  
glUniform3fv(lightPosId, 1, (GLfloat *)&myLightPos);
```



```
glBindVertexArray(myVaoId);
glUniform3fv(lightPosId, 1, (GLfloat *)&myLightPos);
glUniformMatrix4fv( modelviewId, 1, GL_TRUE,
                   (GLfloat *)&myMatrix);
glDrawArrays(GL_TRIANGLES, 0, 300);
glFlush();
```

