The OpenGL API

Patrick Cozzi
University of Pennsylvania
CIS 565 - Fall 2012

Agenda

- Today: OpenGL shaders and uniforms
- Later: efficient buffer usage

OpenGL

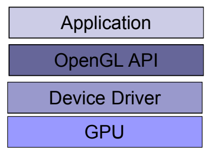
- Is a C-based API
- Is cross platform
- Is run by the *ARB*: Architecture Review Board
- Hides the device driver details
- OpenGL vs. Direct3D

OpenGL

- We are using GL 2
  - No fixed function vertex and fragment shading
  - No legacy API calls:
    - `glBegin()`
    - `glRotatef()`
    - `glTexEnvf()` ← Recall the fixed function light map
    - `AlphaFunc()` ← Why was the alpha test remove?
    - …

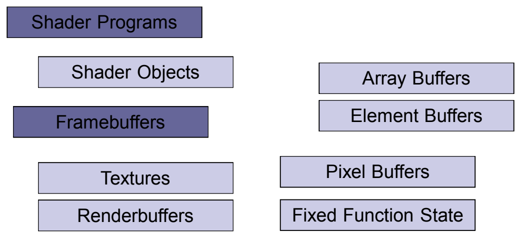
OpenGL

- Software stack:



OpenGL

- Major objects:



- We are not covering everything. Just surveying the most relevant parts for writing GLSL shaders

Shaders

- *Shader object*: an individual vertex, fragment, etc. shader
  - Are provided shader source code as a string
  - Are compiled
- Shader program: Multiple shader objects linked together

Shader Objects

- Compile a shader object:

```
const char *source = // ...
GLint sourceLength = // ...

GLuint v = glCreateShader(GL_VERTEX_SHADER);

glShaderSource(v, 1, &source, &sourceLength);

glCompileShader(v);

GLint compiled;
glGetShaderiv(v, GL_COMPILE_STATUS, &compiled);
// success:  compiled == GL_TRUE

// ...
glDeleteShader(v);
```

Shader Objects

- Compile a shader object:

```
const char *source = // ...
GLint sourceLength = // ...

GLuint v = glCreateShader(GL_VERTEX_SHADER);

glShaderSource(v, 1, &source, &sourceLength);

glCompileShader(v);

GLint compiled;
glGetShaderiv(v, GL_COMPILE_STATUS, &compiled);
// success:  compiled == GL_TRUE

// ...
glDeleteShader(v);
```

OpenGL functions start with `gl`. Why? How would you design this in C++?

`v` is an opaque object
- What is it under the hood?
- How would you design this in C++?

Shader Objects

- Compile a shader object:

```
const char *source = // ...
GLint sourceLength = // ...

GLuint v = glCreateShader(GL_VERTEX_SHADER);

glShaderSource(v, 1, &source, &sourceLength);

glCompileShader(v);

GLint compiled;
glGetShaderiv(v, GL_COMPILE_STATUS, &compiled);
// success:  compiled == GL_TRUE

// ...
glDeleteShader(v);
```

Provide the shader's source code

Where should the source come from?

Why can we pass more than one string?

Shader Objects

- Compile a shader object:

```
const char *source = // ...
GLint sourceLength = // ...

GLuint v = glCreateShader(GL_VERTEX_SHADER);

glShaderSource(v, 1, &source, &sourceLength);

glCompileShader(v);

GLint compiled;
glGetShaderiv(v, GL_COMPILE_STATUS, &compiled);
// success:  compiled == GL_TRUE

// ...
glDeleteShader(v);
```

Compile, but what does the driver really do?

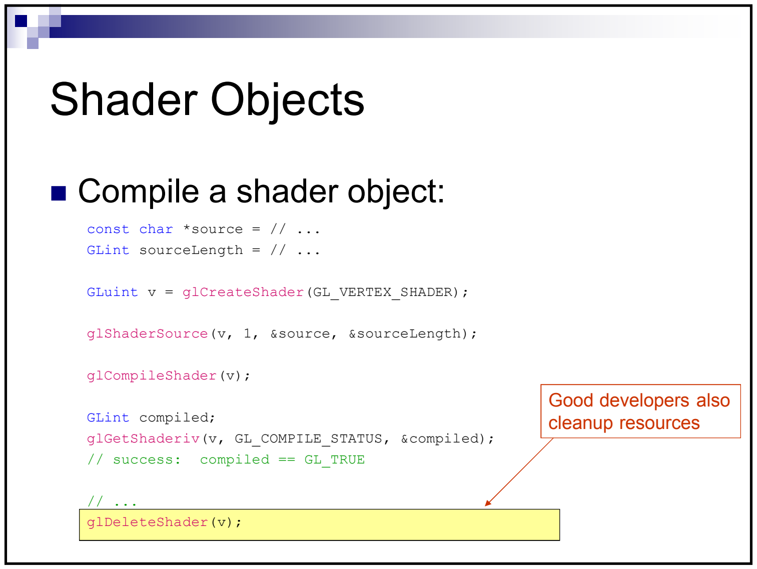
Shader Objects

- Compile a shader object:

```
const char *source = // ...
GLint sourceLength = // ...

GLuint v = glCreateShader(GL_VERTEX_SHADER);

glShaderSource(v, 1, &source, &sourceLength);

glCompileShader(v);

GLint compiled;
glGetShaderiv(v, GL_COMPILE_STATUS, &compiled);
// success:  compiled == GL_TRUE

// ...
glDeleteShader(v);
```

Good developers check for error. Again, how would you design this in C++?

Calling `glGet*` has performance implications. Why?

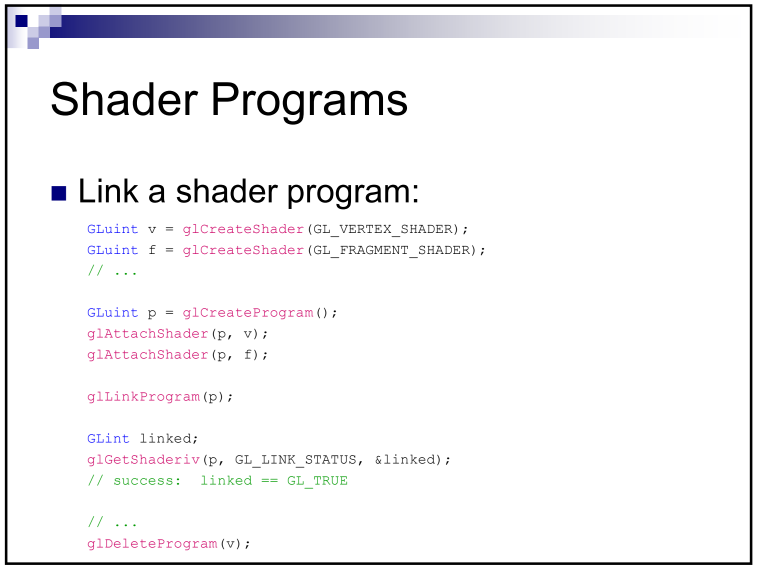


Shader Objects

- Compile a shader object:

```
const char *source = // ...
GLint sourceLength = // ...

GLuint v = glCreateShader(GL_VERTEX_SHADER);

glShaderSource(v, 1, &source, &sourceLength);

glCompileShader(v);

GLint compiled;
glGetShaderiv(v, GL_COMPILE_STATUS, &compiled);
// success:  compiled == GL_TRUE

// ...
glDeleteShader(v);
```

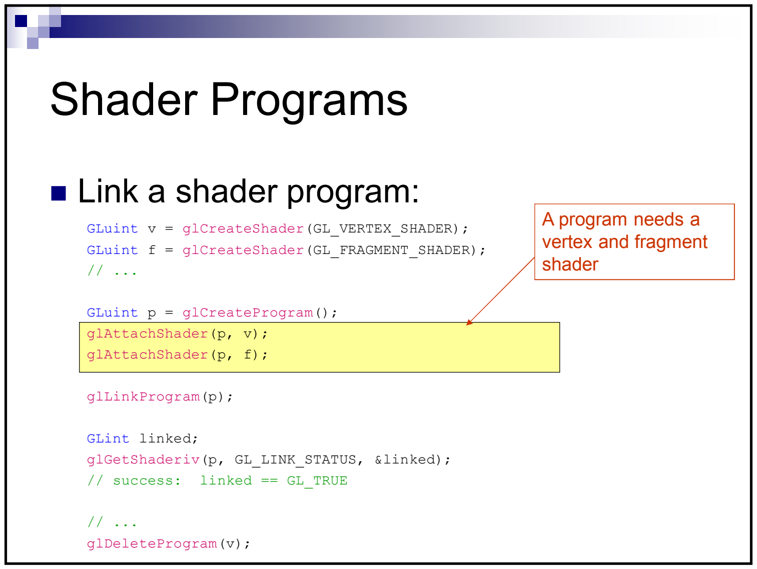
Good developers also cleanup resources



Shader Programs

- Link a shader program:

```
GLuint v = glCreateShader(GL_VERTEX_SHADER);
GLuint f = glCreateShader(GL_FRAGMENT_SHADER);
// ...

GLuint p = glCreateProgram();
glAttachShader(p, v);
glAttachShader(p, f);

glLinkProgram(p);

GLint linked;
glGetShaderiv(p, GL_LINK_STATUS, &linked);
// success:  linked == GL_TRUE

// ...
glDeleteProgram(v);
```

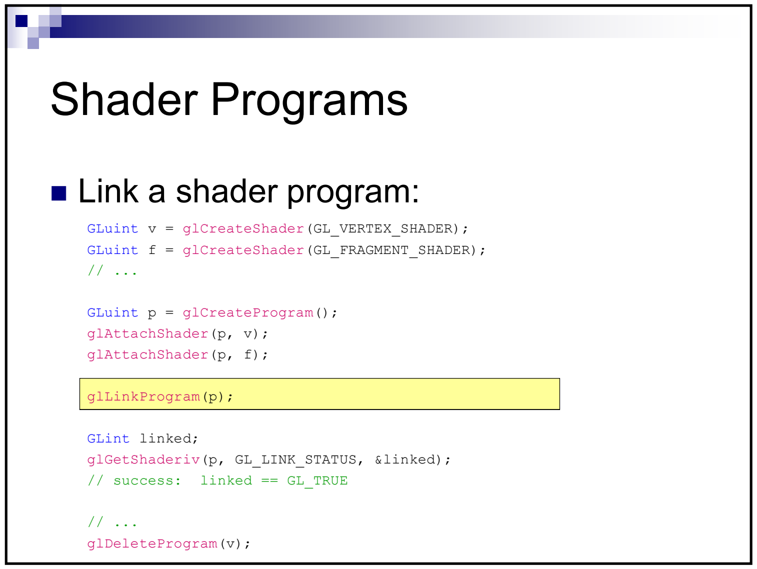


Shader Programs

- Link a shader program:

```
GLuint v = glCreateShader(GL_VERTEX_SHADER);
GLuint f = glCreateShader(GL_FRAGMENT_SHADER);
// ...

GLuint p = glCreateProgram();
glAttachShader(p, v);
glAttachShader(p, f);

glLinkProgram(p);

GLint linked;
glGetShaderiv(p, GL_LINK_STATUS, &linked);
// success:  linked == GL_TRUE

// ...
glDeleteProgram(v);
```

A program needs a vertex and fragment shader



Shader Programs

- Link a shader program:

```
GLuint v = glCreateShader(GL_VERTEX_SHADER);
GLuint f = glCreateShader(GL_FRAGMENT_SHADER);
// ...

GLuint p = glCreateProgram();
glAttachShader(p, v);
glAttachShader(p, f);

glLinkProgram(p);

GLint linked;
glGetShaderiv(p, GL_LINK_STATUS, &linked);
// success:  linked == GL_TRUE

// ...
glDeleteProgram(v);
```

Shader Programs

- Link a shader program:

```
GLuint v = glCreateShader(GL_VERTEX_SHADER);
GLuint f = glCreateShader(GL_FRAGMENT_SHADER);
// ...

GLuint p = glCreateProgram();
glAttachShader(p, v);
glAttachShader(p, f);

glLinkProgram(p);

GLint linked;
glGetShaderiv(p, GL_LINK_STATUS, &linked);
// success:  linked == GL_TRUE

// ...
glDeleteProgram(v);
```

Be a good developer again

Using Shader Programs

```
GLuint p = glCreateProgram();
// ...

glUseProgram(p);
glDraw*(); // * because there are lots of draw functions
```

Part of the current state
- How do you draw different objects with different shaders?
- What is the cost of using multiple shaders?
- How do we reduce the cost?
  - Hint: write more CPU code – really.

Uniforms

```
GLuint p = glCreateProgram();
// ...
glLinkProgram(p);

GLuint m = glGetUniformLocation(p, "u_modelViewMatrix");
GLuint l = glGetUniformLocation(p, "u_lightMap");

glUseProgram(p);
mat4 matrix = // ...
glUniformMatrix4fv(m, 1, GL_FALSE, &matrix[0][0]);
glUniform1i(l, 0);
```

Uniforms

```
GLuint p = glCreateProgram();
// ...
glLinkProgram(p);

GLuint m = glGetUniformLocation(p, "u_modelViewMatrix");
GLuint l = glGetUniformLocation(p, "u_lightMap");

glUseProgram(p);
mat4 matrix = // ...
glUniformMatrix4fv(m, 1, GL_FALSE, &matrix[0][0]);
glUniform1i(l, 0);
```

Each *active* uniform has an integer index location.

Uniforms

```
GLuint p = glCreateProgram();
// ...
glLinkProgram(p);

GLuint m = glGetUniformLocation(p, "u_modelViewMatrix");
GLuint l = glGetUniformLocation(p, "u_lightMap");

glUseProgram(p);
mat4 matrix = // ...
glUniformMatrix4fv(m, 1, GL_FALSE, &matrix[0][0]);
glUniform1i(l, 0);
```

`mat4` is part of the C++ GLM library

GLM: http://www.g-truc.net/project-0016.html#menu

Uniforms

```
GLuint p = glCreateProgram();
// ...
glLinkProgram(p);

GLuint m = glGetUniformLocation(p, "u_modelViewMatrix");
GLuint l = glGetUniformLocation(p, "u_lightMap");

glUseProgram(p);
mat4 matrix = // ...
glUniformMatrix4fv(m, 1, GL_FALSE, &matrix[0][0]);
glUniform1i(l, 0);
```

`glUniform*` for all sorts of datatypes

Uniforms can be changed as often as needed, but are constant during a draw call

Not transposing the matrix

Uniforms

```
GLuint p = glCreateProgram();
// ...
glLinkProgram(p);

GLuint m = glGetUniformLocation(p, "u_modelViewMatrix");
GLuint l = glGetUniformLocation(p, "u_lightMap");

glUseProgram(p);
mat4 matrix = // ...
glUniformMatrix4fv(m, 1, GL_FALSE, &matrix[0][0]);
glUniform1i(l, 0);
```

Why not `glUniform*(p, …)`?