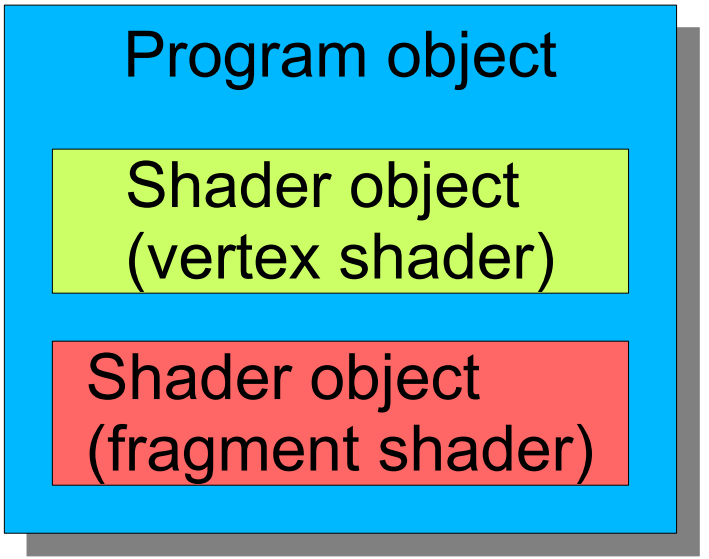
How to get shaders into your programs

There are many steps, but the interface is easy to use

- Shader objects
- Compiling shaders
- Program objects
- Linking programs
- Using programs
- Sending *uniform* variables to your program
- In our code so far, most of this has been handled by the InitShader(...) function



Shader Objects

- Steps to creating a shader object:
  - Create vertex and fragment shader objects
  - Add source code
  - Compile

To create:

- GLuint **glCreateShader**(shaderType)
  - GLenum shaderType
    - GL_VERTEX_SHADER
    - GL_FRAGMENT_SHADER
  - Returns a shader ID

Shader Objects

Shader source code is stored in strings and compiled at runtime

- void **glShaderSource**(shader, count, **strings, *lengths)
  - GLint shader : ID of a previsously created shader object
  - Glsizei count : number of strings
  - const GLchar** strings : array of strings hold source code
    - How many strings? count.
  - const GLint* lengths : array of string lengths
    - Can use NULL if strings are NULL terminated.

Runtime shader compilation

Some interesting possibilities:

- Edit and recompile shader text files without exiting program
- Download shaders from network during execution
- Generate shaders on-the-fly using string processing

Shader Objects

void **glCompileShader**(GLuint shader)

- Status (failure or success) is stored as part of object state

- To check shader object state:
- void **glGetShaderiv**(shader, pname, *params)
  - pname
    - GL_COMPILE_STATUS
    - GL_SHADER_TYPE
    - GL_INFO_LOG_LENGTH
      - Info log holds error messages, warnings and other messages
  - There is a similar function for programs : glGetProgramiv().

Shader Objects : Error Checking

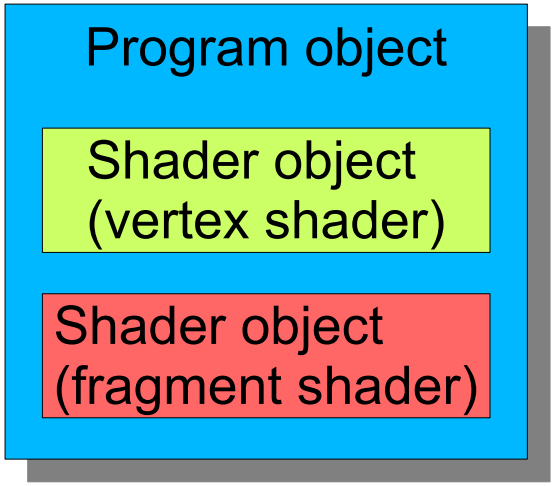
```
void printShaderInfoLog(GLuint obj)
{
        int infologLength = 0;
        int charsWritten  = 0;
        char *infoLog;

        glGetShaderiv(obj, GL_INFO_LOG_LENGTH,&infologLength);

        if (infologLength > 0)
        {
            infoLog = (char *)malloc(infologLength);
            glGetShaderInfoLog(obj, infologLength, &charsWritten, infoLog);
                   printf("%s\n",infoLog);
            free(infoLog);
        }
}
```

Usage:

```
int status = GL_FALSE;
glCompileShader(VShader);
glGetShaderiv(VShader, GL_COMPILE_STATUS, &status);
if(status == GL_FALSE)
{
   printShaderInfoLog(VShader);
}
```

Shader Program Object

- Contains **at least** a vertex shader object and a fragment shader object



- May also contain other optional shaders...
  - There are more programmable stages
    - like geometry shader
  - See CGT 521 or the red book for details

- Shader programs get linked
  - vertex shader ***out***s must match fragment shader ***in***s

Program Objects

- Create program object
  - GLuint **glCreateProgram**(void);
- Add shader objects
  - void **glAttachShader**(GLuint program, GLuint shader);
- Link program
  - void **glLinkProgram**(GLuint program);
  - Assign locations for uniform variables and vertex attributes
  - Resolves references between shader objects
    - *Varying* variables : output from VP, input to FP

Program Objects : Error Checking

```
void printProgramInfoLog(GLuint obj)
{
        int infologLength = 0;
        int charsWritten  = 0;
        char *infoLog;

        glGetProgramiv(obj, GL_INFO_LOG_LENGTH,&infologLength);

        if (infologLength > 0)
        {
            infoLog = (char *)malloc(infologLength);
            glGetProgramInfoLog(obj, infologLength, &charsWritten, infoLog);
                printf("%s\n",infoLog);
            free(infoLog);
        }
}
```

Usage:

```
int status;
glLinkProgram(Program);
glGetProgramiv(Program, GL_LINK_STATUS, &status);
if(status == GL_FALSE)
{
   printf("Shader Link Error\n");
   printProgramInfoLog(Program);
}
```

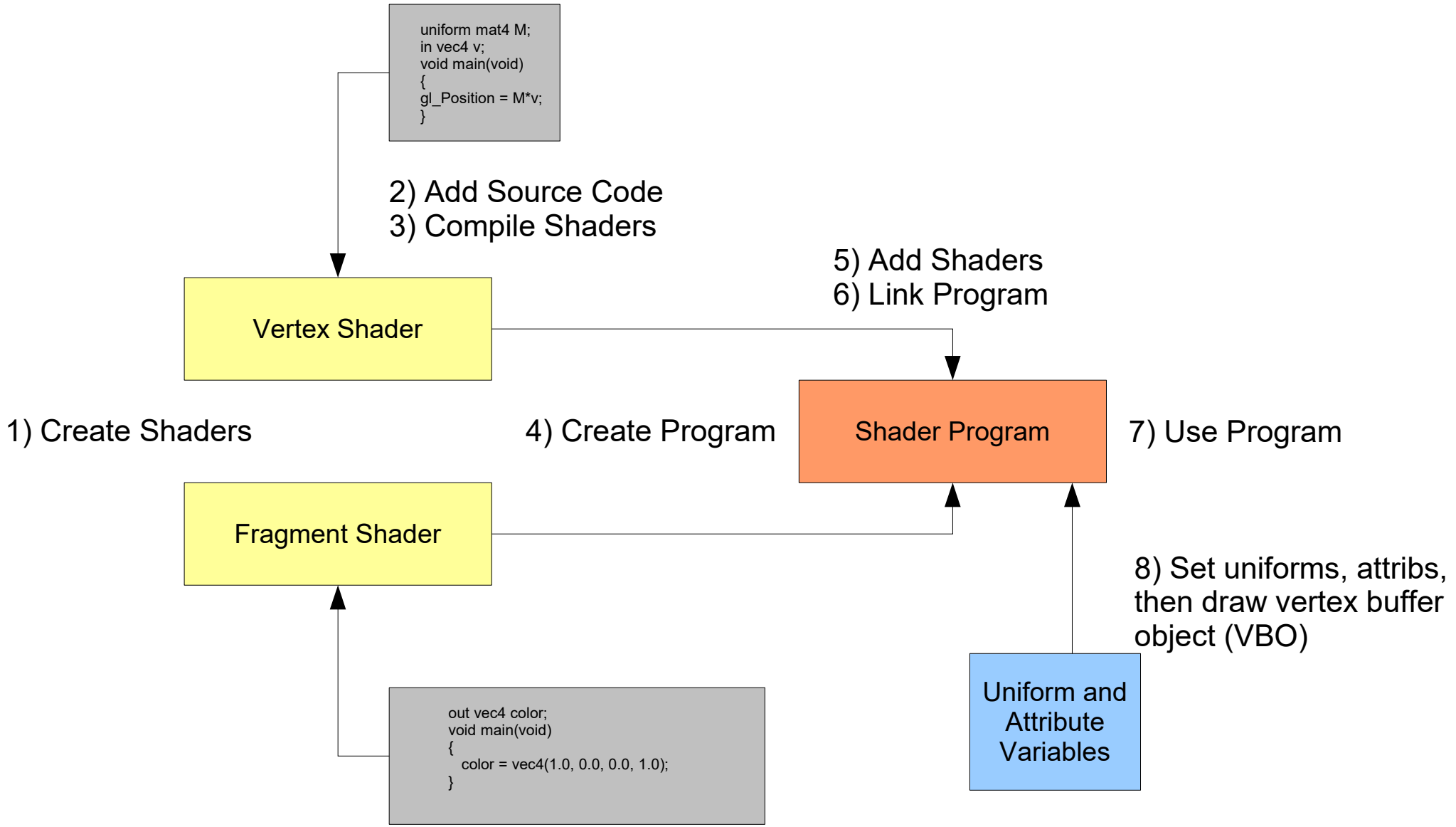
Using a program

void **glUseProgram**(GLuint program);

Subsequent drawing calls will use the specified program.

We can pass the value of uniform variables to the program currently in use.

Overview


uniform mat4 M;
in vec4 v;
void main(void)
{
gl_Position = M*v;
}
2) Add Source Code
3) Compile Shaders
5) Add Shaders
6) Link Program
Vertex Shader
1) Create Shaders
4) Create Program
Shader Program
7) Use Program
Fragment Shader
8) Set uniforms, attribs, then draw vertex buffer object (VBO)
out vec4 color;
void main(void)
{
color = vec4(1.0, 0.0, 0.0, 1.0);
}
Uniform and Attribute Variables

Passing variables to programs

We can also pass the value of uniform variables to the program currently in use.

- GLint glGetUniformLocation(program, *name)
  - const GLchar* name
    - Variable name in shader source code
  - Returns -1 if name is invalid / does not exist
- void glUniform1f(GLint location, GLfloat v)
  - Passes a single floating point variable (1f) to a shader
  - Be sure to use the version of glUniform* that matches the uniform variable type

Example: time based animation

- If the GLSL vertex or fragment shader declares:
  - `uniform float time;`

Then in client code:

```
static float t = 0.0f;
t += 0.1f; //fake time
glUseProgram(prog);
int timeloc = glGetUniformLocation(prog, "time");
glUniform1f(timeloc, t);
```

or, eliminate temp variable:

```
glUniform1f(glGetUniformLocation(prog, "time"), t);
```

Optimization idea

```
int timeloc = glGetUniformLocation(prog, “time”);

glUniform1f(timeloc, t);
```

- Uniform locations don't change after the program is linked.
- So you can get uniform locations one time (after program is linked) and reuse those locations over and over

Warning

- Uniform variables with the same name in different shaders may have different locations

```
glUseProgram(prog1);

int timeloc1 = glGetUniformLocation(prog1, “time”);

…

glUseProgram(prog2);

int timeloc2 = glGetUniformLocation(prog2, “time”);
```

timeloc1 and timeloc2 may have different values

glUniform*

- There are glUniform* functions for
  - float, int
  - Scalars
  - vectors (2, 3, 4 components)
  - matrices (2x2,3x3,4x4)
  - Use int or float for bool
    - 0 = false, otherwise true
  - Use int for texture sampler types...

glUniform* family

- glUniform1f, glUniform2f, glUniform3f , glUniform4f
- glUniform1i, glUniform2i, glUnform3i, glUniform4i
- glUniform1fv, glUniform2f, glUniform3f, glUniform4f
- glUniform1iv, glUniform2iv, glUnform3iv, glUniform4iv
- glUniformMatrix2f, glUniformMatrix3f, glUniformMatrix4f
- glUniformMatrix2fv, glUniformMatrix3fv, glUniformMatrix4fv
- Recent GLSL versions also allow rectangular matrices

- Pass pointers when calling glUniform*v functions.

Passing by pointer

- void glUniform4fv(GLint location,GLsizei **count**, const GLfloat *value);
  - Can pass an array of vec4s at once
  - When passing a single vec4 let **count** = 1

```
glm::vec4 colors[3]; //shader declares uniform vec4 colors[3]
…
glUniform4fv(loc, 3, glm::value_ptr(colors));
```