PV227 GPU Rendering

Marek Vinkler

Department of Computer Graphics and Design

GLSL Main Loop

```
1  #include <GL/glew.h>
2  #include <GL/glut.h>
3
4  void main(int argc, char **argv)
5  {
6     glutInit(&argc, argv);
7     ...
8     glewInit();
9
10    if(glewIsSupported("GL_VERSION_3_3"))
11    {
12       printf("Ready for OpenGL 3.3\n");
13    }
14    else
15    {
16       printf("OpenGL 3.3 not supported\n");
17       exit(1);
18    }
19    setShaders();
20    initGL();
21
22    glutMainLoop();
23 }
```

GLSL Shader Setup – Overview

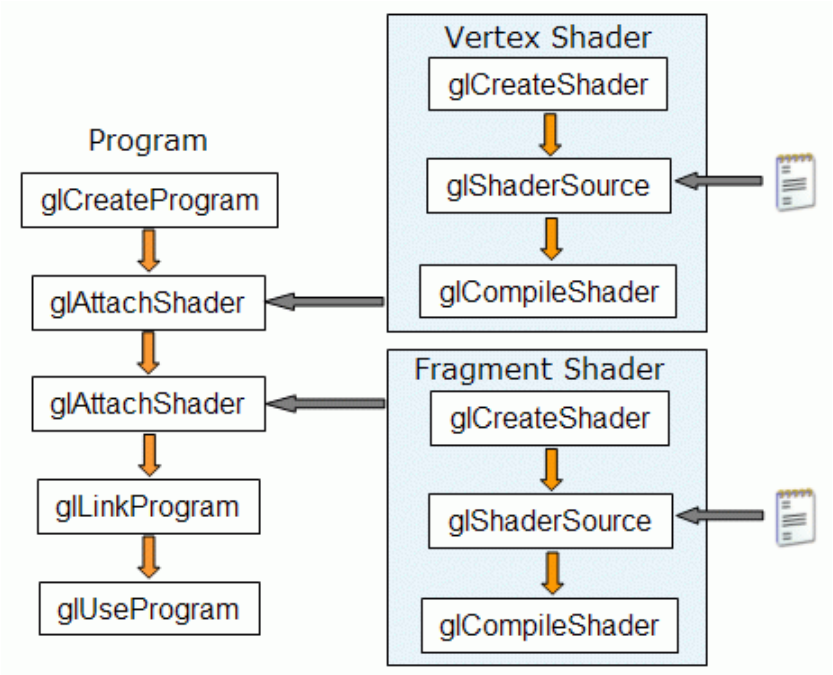

Figure: Taken from lighthouse3d.com

Creating Shader

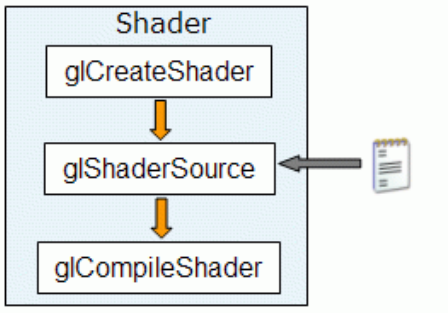

Figure: Taken from lighthouse3d.com

GLuint glCreateShader(GLenum shaderType);

shaderType – GL_{VERTEX|FRAGMENT|GEOMETRY|TESS_CONTROL|TESS_EVALUATION|COMPUTE}_SHADER.

- Creates shader object of a specified type that acts as a container.
- Returns the handle for that container.

Shader Code

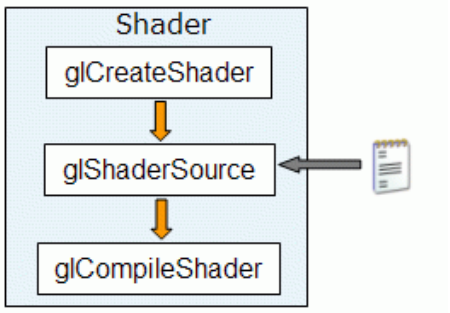

Figure: Taken from lighthouse3d.com

void glShaderSource(GLuint shader, GLsizei count, const GLchar **string, const GLint *length);

shader – the handler to the shader.

count – the number of strings in the arrays.

string – the array of strings.

length – an array with the length of each string; NULL, meaning that the strings are NULL terminated.

- Replaces a source code for the shader.
- Single string can be used instead of an array.
- Multiple strings can define common pieces of code, third-party library functions, ...

Compiling Shader

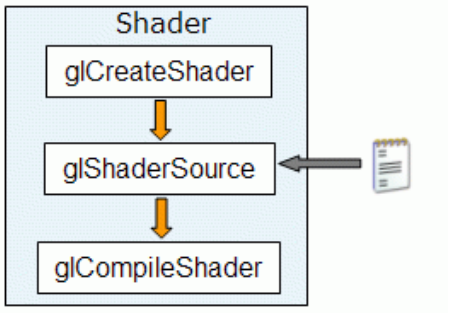

Figure: Taken from lighthouse3d.com

void glCompileShader(GLuint shader);

shader – the handler to the shader.

- Compiles the shader.
- Checks its validity.

Creating Program

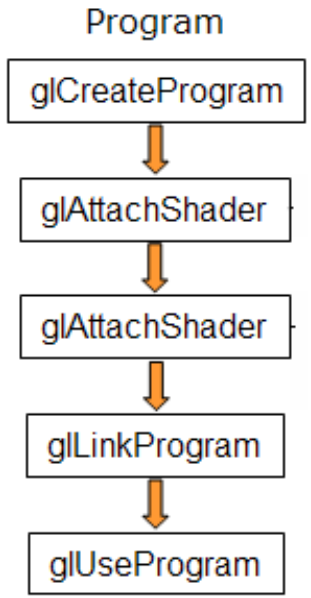

Figure: Taken from lighthouse3d.com

GLuint glCreateProgram(void);

- Creates program object that acts as a container.
- Returns the handle for that container.
- Any number of programs can be created and used in a single frame.
- Programes can be switched at runtime.
- No program used $\rightarrow$ fixed pipeline.

Attaching Shaders

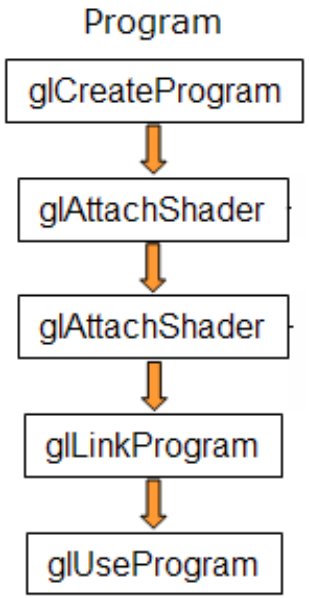

Figure: Taken from lighthouse3d.com

void glAttachShader(GLuint program, GLuint shader);

program – the handler to the program.

shader – the handler to the shader you want to attach.

- Attaches a shader into the program.
- The shaders need neither be compiled nor have source code.
- Any number of shaders can be attached, but only one main for each shader type.
- Single shader can be attached to many programes.

Linking Program

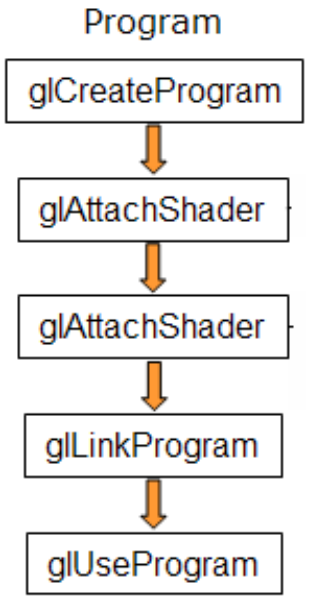

Figure: Taken from lighthouse3d.com

void glLinkProgram(GLuint program);

program – the handler to the program.

- Links the program, resolves cross-shader references.
- Shaders must be compiled at this point.
- Afterwards the shaders can be modified & recompiled.
- Uniform variables are assigned locations and set to 0.

Using Program

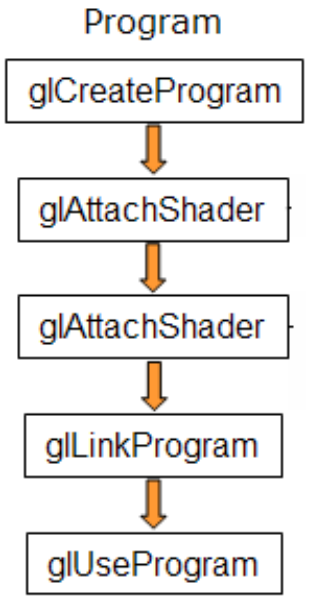

Figure: Taken from lighthouse3d.com

void glUseProgram(GLuint prog);

program – the handler to the program; zero to use fixed functionality .

- Sets the program for use in rendering.
- Relinking a used program also sets it for use.

Cleanup

void glDetachShader(GLuint program, GLuint shader);

program – the program to detach from.

shader – the shader to detach.

- Detaches shader from a program.

void glDeleteShader(GLuint id);

void glDeleteProgram(GLuint id);

id – the handler of the shader / program to erase.

- When attached shader/program is deleted, it is only “marked for deletion” and is fully deleted when no longer used.
- Shaders may be deleted as soon as they are attached, everything will be cleaned up when program is deleted.

GLSL Setup Example

```
1  void setShaders()
2  {
3     char *vs, *fs;
4
5     // Setup
6     v = glCreateShader(GL_VERTEX_SHADER);
7     f = glCreateShader(GL_FRAGMENT_SHADER);
8
9     vs = textFileRead("simple.vert");
10    fs = textFileRead("simple.frag");
11
12    const char * vv = vs;
13    const char * ff = fs;
14
15    glShaderSource(v, 1, &vv, NULL);
16    glShaderSource(f, 1, &ff, NULL);
17
18    free(vs);
19    free(fs);
20
21    glCompileShader(v);
22    glCompileShader(f);
```

GLSL Setup Example (cont.)

```
23
24    p = glCreateProgram();
25
26    glAttachShader(p, v);
27    glAttachShader(p, f);
28
29    glLinkProgram(p);
30    glUseProgram(p);
31
32    ...
33
34    // Clean up
35    glDetachShader(p, v);
36    glDetachShader(p, f);
37
38    glDeleteShader(v);
39    glDeleteShader(f);
40
41    glUseProgram(0);
42    glDeleteProgram(p);
43 }
```

Shader State Query

void glGetShaderiv(GLuint shader, GLenum pname, GLint *params);

shader – the shader to query.

pname – parameter to query.

params – queried state.

pname:

- **GL_SHADER_TYPE** – type of the shader,
- **GL_DELETE_STATUS** – marked for deletion?,
- **GL_COMPILE_STATUS** – last compile successful?,
- **GL_INFO_LOG_LENGTH** – length of the information log,
- **GL_SHADER_SOURCE_LENGTH** – length of the concatenated shader.

Program State Query

void glGetProgramiv(GLuint program, GLenum pname, GLint *params);

program – the shader to query.

pname – parameter to query.

params – queried state.

pname (not all shown):

- **GL_LINK_STATUS** – last link successful?,
- **GL_DELETE_STATUS** – marked for deletion?,
- **GL_VALIDATE_STATUS** – last validation successful?,
- **GL_INFO_LOG_LENGTH** – length of the information log,
- information on number of shaders attached, number of attribute values and uniform variables.

Shader Info Log

void glGetShaderInfoLog(GLuint shader, GLsizei maxLength, GLsizei *length, GLchar *infoLog);

shader – the shader to query.

maxLength – maximal length of output buffer.

length – actual length of the log.

infoLog – the shader log.

- updated during shader compile,
- may contain diagnostic messages, errors, warnings etc. (implementation specific).

Program Info Log

```
void glGetProgramInfoLog(GLuint program, GLsizei maxLength, GLsizei *length,
GLchar *infoLog);
```

program – the program to query.

maxLength – maximal length of output buffer.

length – actual length of the log.

infoLog – the shader log.

- updated during program validation or link,
- may contain diagnostic messages, errors, warnings etc. (implementation specific).

Validation

void glValidateProgram(GLuint program);

program – the program to validate.

- checks whether program can execute given current OpenGL state,
- updates the program log,
- only for developement (slow).

Shader Query Example

```
1  void printShaderInfoLog(GLuint obj)
2  {
3      int infologLength = 0;
4      int charsWritten  = 0;
5      char *infoLog;
6
7      glGetShaderiv(obj, GL_INFO_LOG_LENGTH, &infologLength);
8
9      if(infologLength > 0)
10     {
11         infoLog = (char *)malloc(infologLength);
12         glGetShaderInfoLog(obj, infologLength, &charsWritten,
               infoLog);
13         printf("%s\n",infoLog);
14         free(infoLog);
15     }
16 }
```

Program Query Example

```
1  void printProgramInfoLog(GLuint obj)
2  {
3      int infologLength = 0;
4      int charsWritten  = 0;
5      char *infoLog;
6
7      glGetProgramiv(obj, GL_INFO_LOG_LENGTH, &infologLength);
8
9      if(infologLength > 0)
10     {
11         infoLog = (char *)malloc(infologLength);
12         glGetProgramInfoLog(obj, infologLength, &charsWritten,
               infoLog);
13         printf("%s\n",infoLog);
14         free(infoLog);
15     }
16 }
```