Shader languages

Four different:

Assembly language: Obsolete. Avoid.

Cg: “C for graphics”, NVidia
HLSL: “High-level shading language”, Microsoft
GLSL: “OpenGL shading language”

Choice depends on platform and needs (and taste).

All are similar - easy to port between!

GLSL
OpenGL Shading Language

Language with syntax similar to C

- **Syntax somewhere between C och C++**
- **No classes. Straight and simple code. Remarkably understandable and obvious!**
- **Avoids most of the bad things with C/C++.**

Some advantages come from the limited environment!

“Algol” descentant, easy to learn if you know any of its followers.

GLSL
Example

Vertex shader:

```
#version 150

in  vec3 in_Position;

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

“Pass-through shader”, implements the minimal functionality of the fixed pipeline

GLSL
Example

Fragment shader:

```
#version 150

out vec4 out_Color;

void main(void)
{
	out_Color = vec4(1.0);
}
```

“Set-to-white shader”

Note:

Built-in or custom variables:

gl_Position transformed vertex, out data
in_Position vertex in model coordinates
out_Color resulting fragment color

Also a new built-in type:
vec4 4 component vector

Some possibililties start to show up, right?

What if it looks something like this?

Vertex

```
#version 150

in  vec4 in_Position;
mat4 some_Matrix;

void main(void)
{
	gl_Position = some_Matrix * in_Position;
}
```

Fragment

```
#version 150

in ???
out vec4 out_Color;

void main(void)
{
	out_Color = calcLight(something here?);
}
```

We might want some kind of input, right?

GLSL basics

A tour of the language (with some examples)

- Character set
- Preprocessor directives
- Comments
- Identifiers
- Types
- Modifiers
- Constructors
- Operators
- Built-in functions and variables
- Activating shaders from OpenGL
- Communication with OpenGL

Character set

Alphanumerical characters: a-z, A-Z, _, 0-9

. + - / * % < > [ ] { } ^I & ~ = ! : ; ?

 # for preprocessor directives (!)

space, tab, FF, CR, FL

Note! Tolerates both CR, LF och CRLF! ☺

Case sensitive

BUT

Characters and strings do not exist! ‘a’, “Hej” mm

The preprocessor

#define #undef #if etc

VERSION is useful for handling version differences. It will hardly be possible to avoid in the long run.

#include does not exist! ☺

Comments

/* This is a comment
that spans more than one line */

// but personally I prefer the one-line version

Just like we are used to! ☺

So litter your code with comments!

Identifiers

Just like C: alphanumerical characters, first non-digit

BUT

Reserved identifiers, predefined variables, have the prefix gl_!

It is not allowed to declare your own variables with the gl_ prefix!

Types

There are some well-known scalar types:

void: return value for procedures
bool: Boolean variable, that is a flag
int: integer value
float: floating-point value
double: double precision floating-point value

More types

Vector types:

vec2, vec3, vec4: Floating-point vectors with 2, 3 or 4 components

bvec2, bvec3, bvec4: Boolean vectors

ivec2, ivec3, ivec4: Integer vectors

mat2, mat3, mat4: Floating-point matrices of size 2x2, 3x3, 4x4

Most common: vec2, vec3, vec4, mat3, mat4!

Important!

Modifiers

Variable usage is declared with modifiers:

const

attribute (in)

uniform

varying (in/out)

If none of these are used, the variable is “local” in its scope and can be read and written as you please.

const

constant, assigned at compile time, can not be changed

attribute and uniform

attribute (declared "in" in the shader) is argument from OpenGL, per-vertex-data

uniform is argument from OpenGL, per primitive. Can not be changed within a primitive.

varying (”in”, ”out”)

data that should be interpolated between vertices

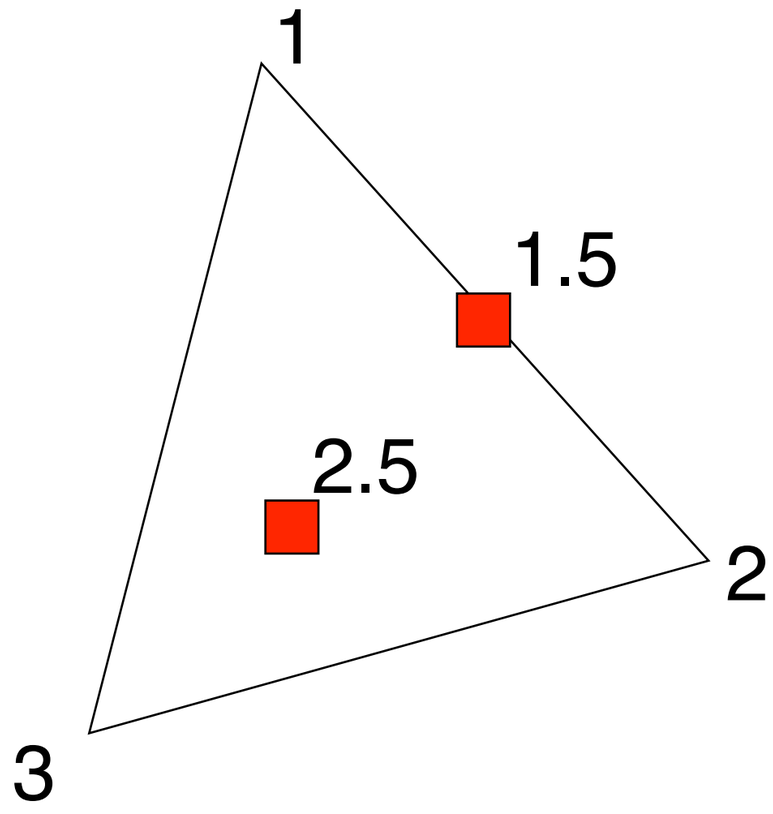
Written in vertex shader

Read (only) by fragment shaders

Declared "out" in vertex, "in" in framgent shader. In the fragment shader, they are read only.

Examples: texture coordinates, normal vectors for Phong shading, vertex color, light value for Gouraud shading

varying (”in”, ”out”)



”varying” or ”in/out”?

”varying” is a keyword in older GLSL, replaced by ”in/out” in newer (somewhat more intuitive)

I will use "varying" as a term denoting this kind of interpolated variables.

Compilation and execution

Done in two steps:

1) Initialization, compilation

- **Create a “program object”**
- **Create a “shader object” and pass source code to it**
- **Compile the shader programs**

2) Activation

- **Activate the program object for rendering**

The entire initialization in code

```
PROG = glCreateProgram();

VERT = glCreateShader(GL_VERTEX_SHADER);
   text = readTextFile(“shader.vert”);
glShaderSource(VERT, 1, text, NULL);
   glCompileShader(VERT);
```

Same for fragment shader

```
glAttachShader(PROG, VERT);
glAttachShader(PROG, FRAG);

   glLinkProgram(PROG);
```

Activate the program for rendering

With an installed and compiled shader program:

```
extern GLuint PROG;
```

activate:

```
glUseProgram(PROG);
```

deactivate:

```
glUseProgram(0);
```