GLSL
OpenGL Shading Language

Language with syntax similar to C

- **Syntax somewhere between C och C++**
- **No classes. Stranight ans 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:

```
void main()
{
    gl_Position = gl_ProjectionMatrix *
                  gl_ModelViewMatrix * gl_Vertex;
}
```

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

GLSL
Example

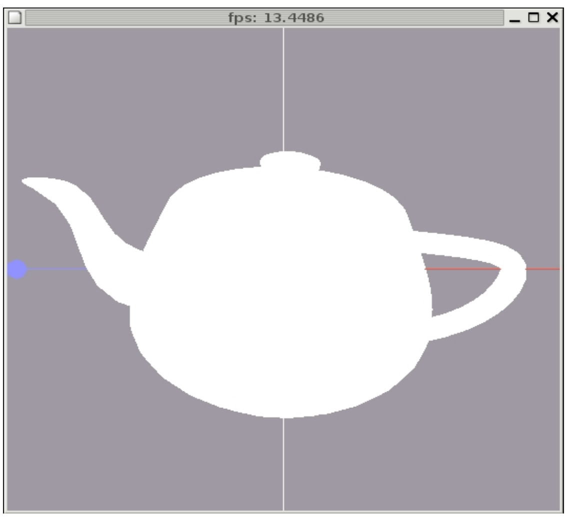
Fragment shader:

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

“Set-to-white shader”

Exempel

**Pass-through vertex shader
+ set-to-white fragment shader**




```
// Vertex shader
void main()
{
    gl_Position = gl_ProjectionMatrix *
            gl_ModelViewMatrix * gl_Vertex;
}

// Fragment shader
void main()
{
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
```

Note:

uilt-in variables:

l_Position	**transformed vertex, out data**
l_ProjectionMatrix	**projection matrix**
l_ModelViewMatrix	**modelview matrix**
l_Vertex	**vertex in model coordinates**
l_FragColor	**resulting fragment color**

lso a new built-in type:

ec4 **4 component vektor**

ome possibililties start to show up, right?

Also note:

Matrix multiplication using the * operator

Shaders always start in main()

Comment: This multiplication is extremely common:

```
gl_Position = gl_ProjectionMatrix * gl_ModelViewMatrix * gl_Vertex;
```

alias:

```
gl_ModelViewProjectionMatrix
```

r

```
ftransform();
```

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

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

 # 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

However, long and double do not exist.

Mer typer

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

mportant!

Modifiers

Variable usage is declared with modifiers:

const

attribute

uniform

varying

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 is argument from OpenGL, per-vertex-data

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

Many predefined variables are "attribute" or "uniform".

varying

data that should be interpolated between vertices

Written in vertex shader

Read (only) by fragment shaders

In both shaders they must be declared “varying”. In the fragment shader, they are read only.

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

Example: Gouraud shader

No, we didn’t learn shaders to do Gouraud shading, but it is a simple example

Transform normal vectors
Calculate shading value per vertex, (here sing diffuse only), by dot product with light irection
Interpolate between vertices

Gouraud shader

Vertex shader

```
varying float shade;

void main()

    vec3 norm;
    const vec3 light = {0.58, 0.58, 0.58};

    gl_Position = gl_ProjectionMatrix *
                        gl_ModelViewMatrix * gl_Vertex;
    norm = normalize(gl_NormalMatrix * gl_Normal);
    shade = dot(norm, light);
```


Gouraud shader

Fragment shader

```
varying float shade;

void main()
{
    gl_FragColor = vec4(clamp(shade, 0, 1));
}
```

Gouraud shader

Note:

The variable “shade” is varying , interpolated between vertices!

dot() och normalize() do what you expect.

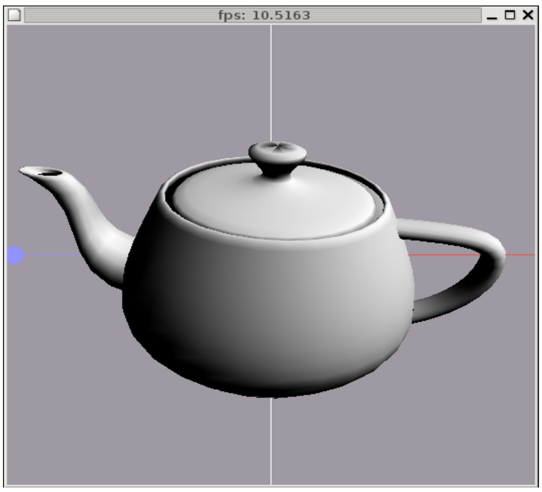
clamp() clamps a variable within a desired interval.

gl_Normal is the normal vector in model coordinates
gl_NormalMatrix transform for normal vectors

The contstant vector light() is here hard coded

Gouraud shader

Result



Very good - for this model

Example:
Phong shader

A more meaningful example

- **Transform normal vectors**
- **Interpolate normal vectors between vertices**
- **Calculate shading value per fragment**

Practically the same operations, but the light calculation are done in the fragment shader

Phong shader
Vertex shader

```
varying vec3 norm;

void main()
{
    gl_Position = gl_ProjectionMatrix *
                  gl_ModelViewMatrix * gl_Vertex;
    norm = normalize(gl_NormalMatrix * gl_Normal);
}
```

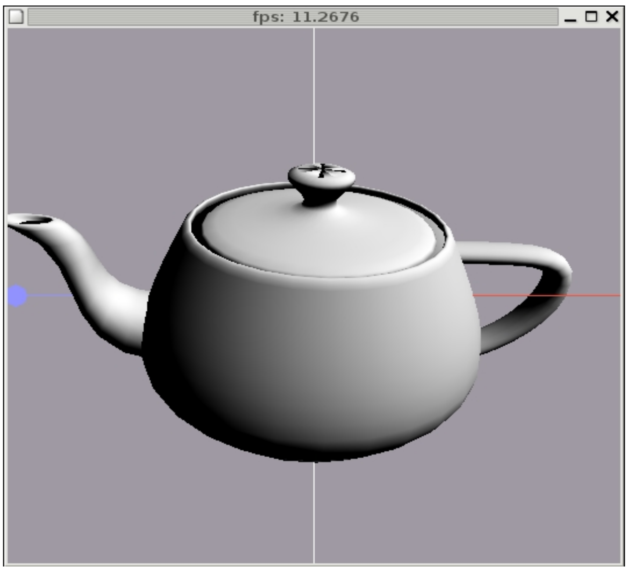
Phong shader
Fragment shader

```
varying vec3 norm;

void main()
{
    float shade;
    const vec3 light = {0.58, 0.58, 0.58};

    shade = dot(normalize(norm), light);
    shade = clamp(shade, 0, 1);
    gl_FragColor = vec4(shade);
}
```


Phong shader
Result



Nice and smooth!

Texture coordinates

Built-in variables:

gl_MultiTexCoord0 is texture coordinate for vertex for texture unit 0.

gl_TexCoord[0] is a built-in varying for interpolating texture coordinates.

gl_TexCoord[0].s and gl_TexCoord[0].t give the S and T components separately.

Example:
Procedural texture

Texture generated by fragment shader!

- **Vertex shader passes on texture coordinates**
- **Texture coordinates are used in a texture generating function in the fragment shader**

Simpler than you might think!

Procedural texture Vertex shader

```
void main()
{
    gl_Position = gl_ProjectionMatrix *
            gl_ModelViewMatrix * gl_Vertex;
    gl_TexCoord[0] = gl_MultiTexCoord0;
}
```

Simple “pass-through” shader, but here including passing on texture coordinates

Procedural texture Fragment shader

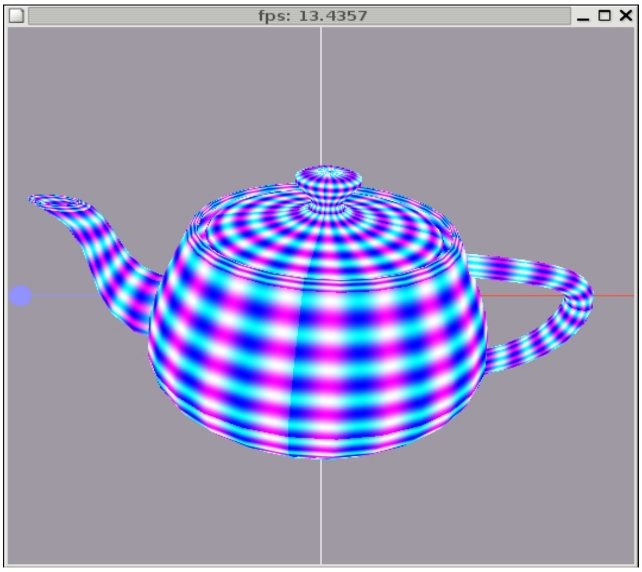
```
void main()
{
    gl_FragColor = vec4(1.0, 1.0, 1.0, 0.0);

    float a = sin(gl_TexCoord[0].s*30)/2+0.5;
    float b = sin(gl_TexCoord[0].t*30)/2+0.5;
    gl_FragColor = vec4(a, b, 1.0, 0.0);
}
```

Simple! The fragment color is a function of S and T, in this case a simple sin for each.

Note sin(), one out of many commmon mathematical functions, built-in!

Procedural texture
Result



Texture data

In order to use predefined texture data, they should be communicated from OpenGL!

This is done by a “uniform”, a variable that can not be changed within a primitive.

“samplers”: pre-defined type for referencing texture data

Texture access

Exempel:

```
uniform sampler2D texture;

void main()
{
    gl_FragColor = texture2D(texture,
                             gl_TexCoord[0].st);
}
```

texture2D() performs texture access

Communication with host

Important! The host must be able to set unoformand attribute variables for GLSL to read.

GLSL can only output information through fragments.

OpenGL sends address and names to GLSL with special calls.

Example: uniform float:

```
float myFloat;
GLint loc;

loc = glGetUniformLocation(p, "myFloat");
glUniform1f(loc, myFloat);
```

p: Ref to shader program, as installed earlier,

loc: address to variable

Now the variable can be used in GLSL:

```
uniform float myFloat;
```

Note that the string passed to glGetUniformLocation specifies the name in GLSL!

Example: texture, uniform sampler:

```
GLuint tex;

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex);
loc = glGetUniformLocation(PROG, "tex");
glUniform1i(loc, 0);
```

zero to glUniform1i = texture unit number!

Används i shader:

```
uniform sampler2D tex;

vec3 texval = vec3(texture2D(texture, gl_TexCoord[0].st));
```

Example: Multitexturing

Bind one texture per texturing unit
Pass GLSL enhetsnummer and name
Declare as samplers in GLSL

Many possibilities:

- **Combine texture data using arbitrary function.**
- **Make one texture sensitive to lighting and another not.**
- **Use texture as bump map**

My simple exmaple: Select different texture dependning of light level.

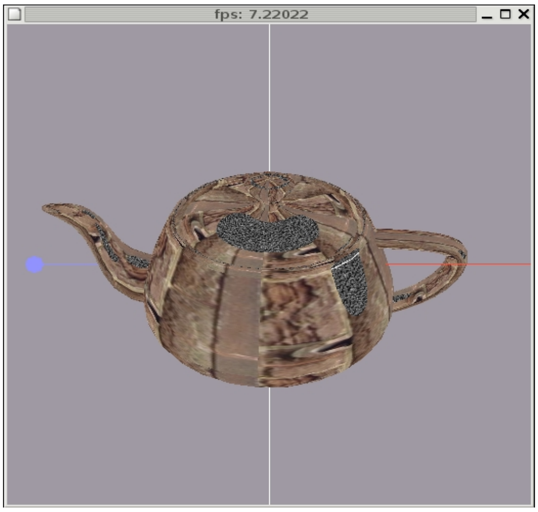
Example: Multitexturing

(Lighting omitted, calculates light from two light sources, spec/spec2)

```
uniform sampler2D tex;
uniform sampler2D bump;
...
    vec3 texval = vec3(texture2D(tex, gl_TexCoord[0].st));
if (spec+spec2 > kLimit)
    texval = vec3(texture2D(bump, gl_TexCoord[0].st));
```

Example: Multitexturing

Switches texture in “highlights”



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

) Activation

- Activate the program object for rendering

Create a “program object”

lCreateProgram
glCreateProgramObjectARB)

he “program object” is the root node to all nformation OpenGL has about our shaders. reate one for each shader pair in your pplication.

Create “shader objects”

lCreateShader (glCreateShaderObjectARB)

ead source code and pass to the shader bject:

lShaderSource (glShaderSourceARB)

ompile!

lCompileShader (glCompileShaderARB)

Attach and link

For both vertex and fragment shader:

glAttachShader (glAttachObjectARB)

Link:

glLinkProgram (glLinkProgramARB)

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

Givet ett installerat och kompilerat programobjekt:

```
extern GLuint PROG; // Was GLhandleARB
```

activate:

```
glUseProgram(PROG);
```

deactivate:

```
glUseProgram(0);
```