# 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).**



# 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 variables:**

**gl_Position transformed vertex, out data**
**gl_ProjectionMatrix projection matrix**
**gl_ModelViewMatrix modelview matrix**
**gl_Vertex vertex in model coordinates**
**gl_FragColor resulting fragment color**

**Also a new built-in type:**
**vec4 4 component vektor**

**Some 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
```

**or**

```
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**

**. + - / * % < > [ ] { } ^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**

**However, long and double do not exist.**

# 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**

**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 (”in”, ”out”)

**data that should be interpolated between vertices**

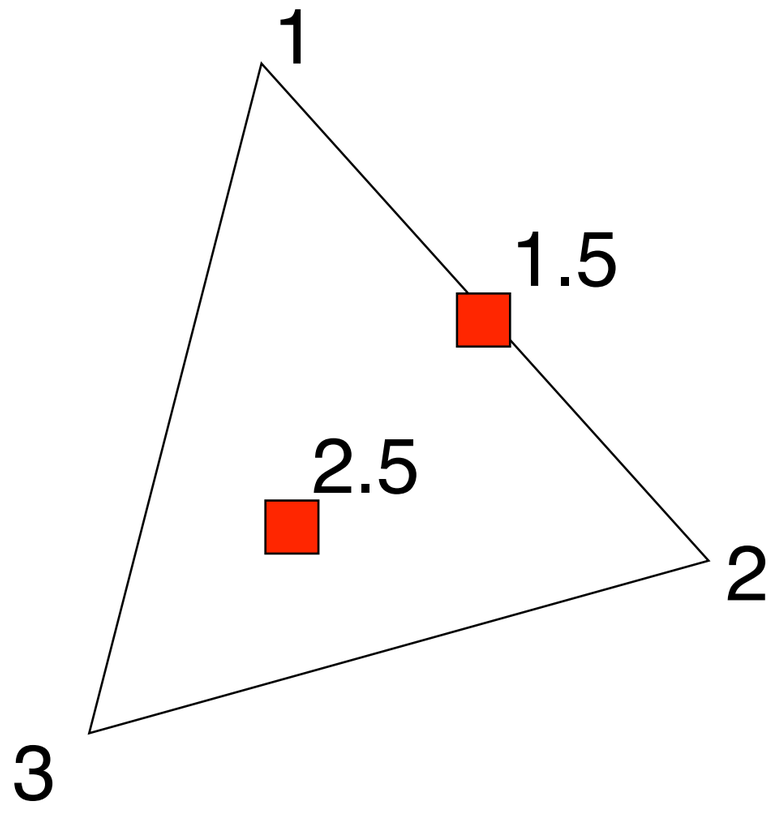
**Written in vertex shader**

**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**

# varying (”in”, ”out”)



# ”varying” or ”in/out”?

**”varying” is being replaced by ”in/out” (somewhat more intuitive)**

**We use ”varying” for compatibility.**

**For our purposes, both are ”right”**

# 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);
```