


# GLSL
## OpenGL Shading Language

**Language with syntax similar to C**

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

**Some advantages come from the limited environment!**

**“Algol” descendant, 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”**

# Example

## 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);
}
```


# So what do we care about for GPGPU?

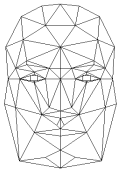
**Vertex shader almost irrelevant.**

**Light sources, vertex colors, geometry shader... forget about most of them here.**

**Fragment shader is the computing kernel.**

**Texture access vital.**

**Uniforms = constant memory.**





# Texture access

## Example:

```
uniform sampler2D texture;

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

**texture2D() performs texture access**

# 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.**

# 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

# Writing to textures

## Use Framebuffer Objects

## Lets you bind a texture as rendering target.



# Ping-ponging in working code

```
void runfilter(GLuint shader, GLuint texin, GLuint
texin2, GLuint fboout)
{
    glBindFramebufferEXT(GL_FRAMEBUFFER, fboout);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, texin2);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texin);

    glViewport(0, 0, width, height);
    if (fboout == 0)
        glViewport(0, 0, lastw, lasth);

    // Clean matrices!
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();

    glUseProgram(shader);

    // Polygon over framebuffer
    glBegin(GL_POLYGON);
    glTexCoord2f(0, 0);
    glVertex2f(-1, -1);
    glTexCoord2f(1, 0);
    glVertex2f(1, -1);
    glTexCoord2f(1, 1);
    glVertex2f(1, 1);
    glTexCoord2f(0, 1);
    glVertex2f(-1, 1);
    glEnd();

    // Restore
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();
}
```



# ...gets pretty nice in the end

```
// Run treshold shader from tex 1 to tex/fbo 2
   runfilter(tresholdShader, tex1, 0, fb2);

// Filter several times
   for (i = 0; i < loops; i++)
   {
      runfilter(lpVShader, tex2, 0, fb3);
      runfilter(lpHShader, tex3, 0, fb2);
   }

// Output result
   runfilter(0, tex2, 0, 0);
```

# Conclusions:

- **Shader-based GPGPU is not dead, it is just not hyped**

Superior compatibility and ease of installation makes it highly interesting for the forseeable future. Especially suitable for all image-related problems.

- **GLSL**

The OpenGL shading language was introduced.

- **How to do GPGPU with shaders**

FBOs, Ping-ponging, algorithms, special considerations.

# That’s all folks!

Next time: OpenCL