FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Diplomarbeitspräsentation

Shading Framework for Modern Rendering Engines

Masterstudium:
Visual Computing

Onur Dogangönül

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Arbeitsbereich Computergraphik
Betreuung:
Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

Simple GLSL Example

Three different features are implemented
- Common **vertex transformation** (Red highlights)
- Simple **diffuse lighting** model (Blue highlights)
- Simple **texturing** (Green highlights)

Decomposition into **vertex** and **fragment** shader

Pass-through code for **texture coordinates** in vertex shader

Lack of reusability
Lack of composability
Lack of modularity

```
#version 420 core

uniform Uniforms
{
 mat4 modelView;
 mat4 proj;
 mat4 view;
 mat3 normalMatrix;
 vec4 w_lightPos;

} uniforms;

in vec3 vertex;
in vec3 normal;
in vec2 texCoord;

out VS2PS
{
 vec3 v_normal;
 vec3 v_lightDir;
 vec2 texCoord;

} vs2ps;

void main(void)
{
 vec4 pos = vec4(vertex, 1.0f);
 vec4 v_pos = uniforms.modelView * pos;

 vec4 v_lightPos = uniforms.view * uniforms.w_lightPos;
 vec3 v_lightPos3 = v_lightPos.xyz / v_lightPos.w;

 vec3 v_normal = uniforms.normalMatrix * normal;

 vec3 v_pos3 = v_pos.xyz / v_pos.w;
 vec3 v_lightDir = normalize(v_lightPos3 - v_pos3);

 vs2ps.v_normal = v_normal;
 vs2ps.v_lightDir = v_lightDir;
 vs2ps.texCoord = texCoord;

 gl_Position = uniforms.proj * v_pos;
}
```

```
#version 420 core

uniform sampler2D tex;

in VS2PS
{
 vec3 v_normal;
 vec3 v_lightDir;
 vec2 texCoord;

} vs2ps;

void main(void)
{
 vec4 diffuse = texture(tex,
    vs2ps.texCoord);
 float lighting = max(0.0f,
  dot(normalize(vs2ps.v_lightDir),
  normalize(vs2ps.v_normal)));
 vec4 color = diffuse * lighting;

 gl_FragData[0] = color;
}
```

Motivation

Challenging development environment for graphics developers
- C/C++ application code
- **Procedural per-stage** HLSL/GLSL shaders
- Hardware state setup

Numerous shader files
Undesireable **duplicate**- and **pass-through** shader code
- Simple changes have to be refactored over whole code base

Code Reusability? "Copy-paste" programming, duplicate code, bad code maintainability
Composability? Results often in complex über-shader
Modularity? Algorithms are propagated to according pipeline stages

```
abstract shader class Base extends OpenGL42DrawPass
{
 // @Uniform
 input @Uniform mat4 world;
 input @Uniform mat4 view;
 input @Uniform mat4 proj;

 @Uniform mat4 modelView = view * world;
 @Uniform mat4 mvp = proj * modelView;
 @Uniform mat3 normalMatrix =
     mat3(transpose(inverse(modelView)));

 // @AssembledVertex
 input @AssembledVertex vec3 pos;

 // @RasterVertex
     override RS_Position = mvp * vec4(pos, 1.0f);

 // @Pixel
 abstract output @Pixel vec4 target;
}

shader class Diffuse extends Base
{
 // @Uniform
 input @Uniform vec4 w_lightPos;

 // @AssembledVertex
 input @AssembledVertex vec3 normal;

 // @CoarseVertex
 @CoarseVertex vec4 v_pos = modelView * vec4(pos, 1.0f);
 @CoarseVertex vec3 v_pos3 = v_pos.xyz / v_pos.w;

 @CoarseVertex vec4 v_lightPos = view * w_lightPos;
 @CoarseVertex vec3 v_lightPos3 = v_lightPos.xyz / v_lightPos.w;

 @CoarseVertex vec3 v_lightDir = normalize(v_lightPos3 - v_pos3);
 @CoarseVertex vec3 v_normal = normalMatrix * normal;

 // @Fragment
 virtual @Fragment vec4 diffuse = vec4(1.0f, 1.0f, 1.0f, 1.0f);
 @Fragment float lighting =
  max(0.0f, dot(normalize(v_lightDir), normalize(v_normal)));
 @Fragment vec4 color = diffuse * lighting;

     // @Pixel
     override target = color;
}
```

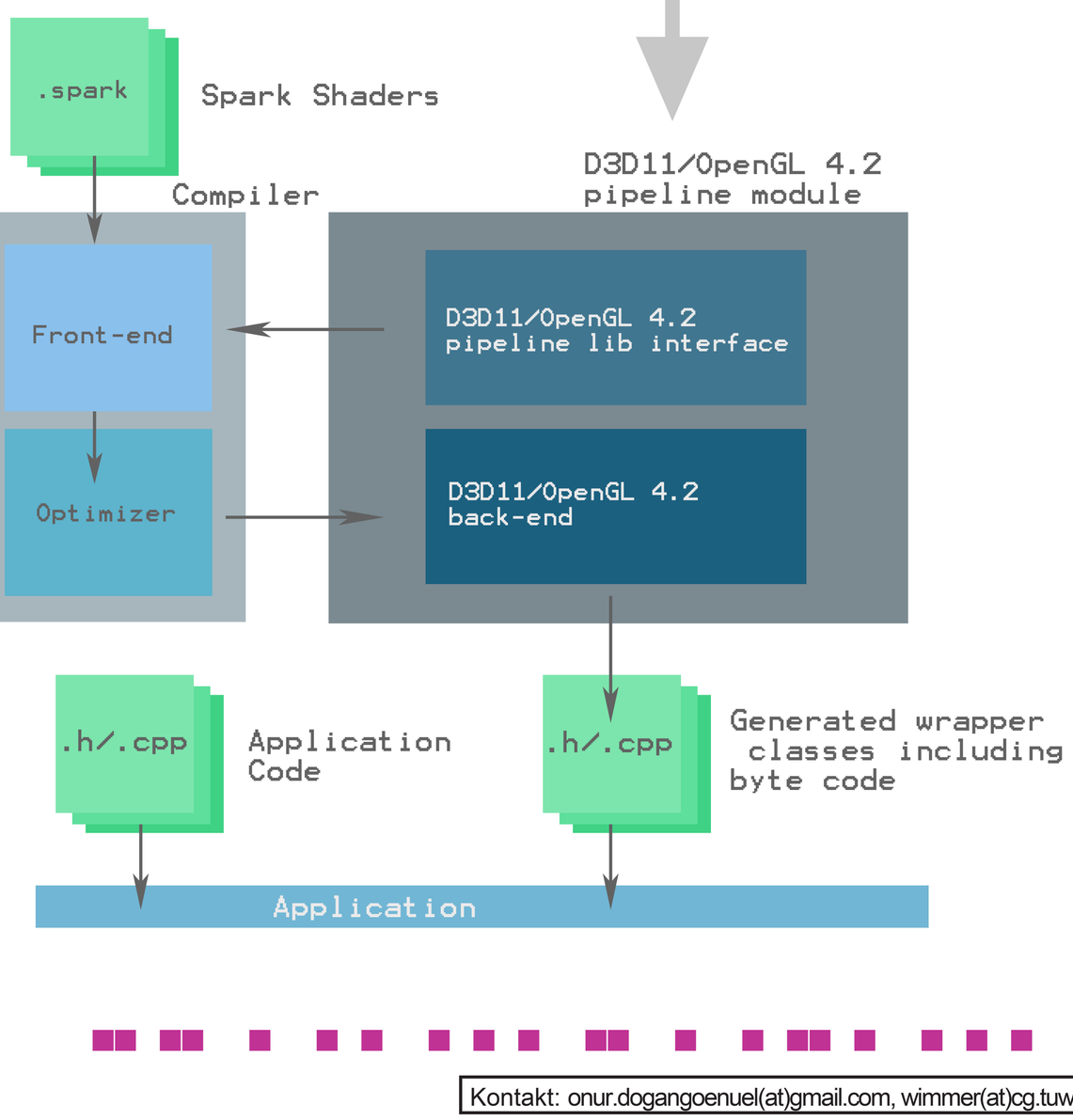
```
shader class Texturing extends Diffuse
{
 // @Uniform
 input @Uniform sampler2D tex;

 // @AssembledVertex
 input @AssembledVertex vec2 texCoord;

 // @Fragment
 override diffuse = texture(tex, texCoord);
}
```

Spark Shading Framework [Foley et al., 2011]

Novel **aspect-oriented per-pipeline** shading language
Supports **D3D11 pipeline, Separation of concerns**
Shader **classes** and **inheritance, Computation rates**
Automatic data plumbing

Within this Thesis

New **OpenGL 4.2 back-end**
Documention of **low-level compiler details**
New Shading **examples**
Evaluation of the new back-end



Results

Method	*GLSL (fps)*	*Spark (fps)*	*% Spark vs. GLSL*
Simple Shading (19.2k)*	112.158	109.665	**+2.25%**
Cube mapping (1.4k)*	159.514	149.544	**+6.67%**
Quad Tessellation (1.6k) **	279.178	269.178	**+3.66%**

* number of triangles rendered ** number of quads with 16 control points

Kontakt: onur.dogangoenuel(at)gmail.com, wimmer(at)cg.tuwien.ac.at