VU Entwurf und Programmierung einer Rendering-Engine

Shader Programming
Pipeline

- At least Vertex and Fragment needed
- Rasterizer cannot be programmed
Pipeline IO

- Vertex Attributes
- Uniform/Storage Buffers
- Samplers/Images
- Framebuffer Attachments

- Vertex
- Tessellation
- Geometry
- Rasterizer
- Fragment
Example

Render everything white

```cpp
uniform UBO{
    mat4 MVPMat;
};

in vec4 Position;
void Vertex()
{
    vec4 pos = MVPMat * Position;
    gl_Position = pos;
}

out vec4 Color;
void Fragment()
{
    Color = vec4(1,1,1,1);
}
```
Example: Texturing

Render everything textured

```
uniform UBO{
  mat4 MVPMat;
};

in vec4 Position;
in vec2 TexCoord;
out vec2 fs_TexCoord
void Vertex()
{
  vec4 pos = MVPMat* Position;
  fs_TexCoord = TexCoord;
  gl_Position = pos;
}
```

```
uniform sampler2D tex;

in vec2 fs_TexCoord;
out vec4 Color;
void Fragment()
{
  Color = texture(tex, fs_TexCoord);
}
```
Problem

Adding Effects changes all shaders due to Parameter Passing

Solutions

- Maintain copied code
- Shader ControlFlow (if(textured) {...})
- UBERShaders (#ifdef cascades)
- Shader subroutines
- Higher Level languages (Spark, LibSh, etc.)
Abstract Shade Trees

- McGuire et al. 2006

Idea: High level of abstraction, Automatic shader linkage, basis conversion etc
Embedded Metaprogramming

LibSH, Shader algebra, McCool et al. 2004

- LibSh uses C++ metaprogramming using templates
- Shader Algebra provides higher order operators for combining shader programs
Spark

- Foley et al. 2011
- Pipeline shaders
- Frequency annotations
- Object oriented composition using mixins
- Foley’s team continued research:
  - A System for Rapid Exploration of Shader Optimization Choices [He et al.]
  - Shader modules can be used for performance optimization in rendering engine: Shader Components paper, [He et al 2017]
FShade

- Aardvark’s system
- Functional first-class shaders
- High level operators for composition
- Embedded DSL for F#
- Inspired by CoSMo [Haaser et al. 2014]
Shaders as Modules

First class values with full introspection

\[ P_{out} = MVPMatrix \times P_{in} \]
\[ N_{out} = NMMatrix \times N_{in} \]
\[ T_{out} = T_{in} \]
Shaders as Modules

\[ P_{\text{out}} = \text{MVPMatrix} \times P_{\text{in}} \]

\[ N_{\text{out}} = \text{NMatrix} \times N_{\text{in}} \]

\[ T_{\text{out}} = T_{\text{in}} \]

\[ P_{\text{out}} = P_{\text{in}} + 0.5 \times N_{\text{in}} \]
Shader Composition

\[ P_{out} = MVPMatrix \times P_{in} \]
\[ N_{out} = NMatrix \times N_{in} \]
\[ T_{out} = T_{in} \]

\[ P_{out} = P_{in} + 0.5 \times N_{in} \]
Shader Composition

\[ P_{out} = MVPMatrix \times P_{in} \]
\[ N_{out} = NMatrix \times N_{in} \]
\[ T_{out} = T_{in} \]

\[ P_{out} = P_{in} + 0.5 \times N_{in} \]
Shader Composition

\[
\begin{align*}
P_{out} &= \text{MVPMatrix} \times P_{in} \\
N_{out} &= \text{NMatrix} \times N_{in} \\
T_{out} &= T_{in} \\
P_{out} &= P_{in} + 0.5 \times N_{in}
\end{align*}
\]
Shader Composition

\[ P_{out} = \text{MVPMatrix} \times P_{in} \]
\[ N_{out} = \text{NMatrix} \times N_{in} \]
\[ T_{out} = T_{in} \]

\[ P_{out} = P_{in} + 0.5 \times N_{in} \]
# Shader Composition: Example

## Transform

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vec4 pp = MVPMatrix * InPosition;</code></td>
<td></td>
</tr>
<tr>
<td><code>OutPosition = pp;</code></td>
<td></td>
</tr>
<tr>
<td><code>OutNormal = NMatrix * InNormal;</code></td>
<td></td>
</tr>
<tr>
<td><code>OutTexCoord = InTexCoord;</code></td>
<td></td>
</tr>
</tbody>
</table>

## Offset

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vec4 p = InPosition;</code></td>
<td></td>
</tr>
<tr>
<td><code>vec4 n = InNormal;</code></td>
<td></td>
</tr>
<tr>
<td><code>OutPosition = p + 0.5 * n;</code></td>
<td></td>
</tr>
<tr>
<td><code>OutNormal = InNormal;</code></td>
<td></td>
</tr>
</tbody>
</table>
Shader Composition: Example

Compose [Transform; Offset]

```cpp
vec4 pp = MVPMatrix * InPosition;

OutPosition = pp;
OutNormal = NMatrix * InNormal;
OutTexCoord = InTexCoord;

vec4 p = InPosition;
vec4 n = InNormal;

OutPosition = p + 0.5 * n;
OutNormal = InNormal;
```
Shader Composition: Example

Compose [Transform; Offset]

```glsl
vec4 pp = MVPMatrix * InPosition;

vec4 PC = pp;
vec3 NC = NMatrix * InNormal
vec2 TCC = InTexCoord

vec4 p = PC;
vec3 n = NC;

OutPosition = p + 0.5 * n;
OutNormal = NC;
OutTexCoord = TCC;
```

```glsl
vec4 pp = MVPMatrix * InPosition;

OutPosition = pp;
OutNormal = NMatrix * InNormal;
OutTexCoord = InTexCoord;

vec4 p = InPosition;
vec3 n = InNormal;

OutPosition = p + 0.5 * n;
OutNormal = InNormal;
```

High Level Stages

Primitives:

Vertex → Tessellation → Geometry

Fragments:

Fragment
Possible Compositions

- **Vertex → Vertex**: as shown in example
- **Tessellation → Vertex**: use VS code in TessEval after interpolation
- **Geometry → Vertex**: use VS code to post-process all output vertices
- **Geometry → Geometry**: apply second to all primitives generated by the first
- **Fragment → Fragment**: as shown in example
Linking

Shaders have many outputs

Only a few needed (Colors, Normals, etc.)

Remove unused outputs

Via dead code elimination inputs are no longer used
=> remove outputs from prior stage (repeat until VS reached)
## Automatic Passing

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vertex</strong></td>
<td>simple</td>
</tr>
</tbody>
</table>
| **Tessellation** | Interpolate using `gl_TessCoord` when possible  
What to do with int, etc.? |
| **Geometry** | Point Input: easy  
In General: impossible  
FShade provides special API for guiding GS passing |
| **Fragment** | simple           |
Optimizations

Dead Code Elimination

Constant Folding

Function Inlining

Stage Hoisting

Common Subexpression Elimination

Many more...
Implementation Overview

Uses F#'s Quotation Expressions to represent shaders

Quotations allow us to get the function’s code as typed AST

Composition achieved via transforming/combining the AST

Translate AST to GLSL/HLSL/SpirV/etc.
White: FShade

type Vertex =
{
    [<>Position>] pos : V4d
    [<>Color>]    color : V4d
}

let transform (v : Vertex) =
vertex {
    return {
        pos = uniform.MVPMat * v.pos
        color = v.color
    }
}

let white (v : Vertex) =
fragment {
    return { v with color = V4d(1,1,1,1) }
}
White: GLSL output

```glsl
uniform UBO{
    mat4 MVPMat;
};

in vec4 Position;

void Vertex()
{
    vec4 pos = MVPMat* Position;
    gl_Position = pos;
}

out vec4 Color;

void Fragment()
{
    Color = vec4(1,1,1,1);
}
```

compose [ transform; white ]
Texturing: FShade

type Vertex =
  {
    [<Position>]  pos  : V4d
    [<TexCoord>]  tc   : V2d
  }

let tex =
sampler2d {
  addressU  WrapMode.Wrap
  addressV  WrapMode.Wrap
}

let texturing (v : Vertex) =
fragment {
  return { v with color = tex.Sample(v.tc) }
}
Texturing: GLSL output

compose [ transform; texturing ]

uniform UBO{
  mat4 MVPMat;
};

in vec4 Position;
in vec2 TexCoord;
out vec2 fs_TexCoord
void Vertex() {
  vec4 pos = MVPMat * Position;
  fs_TexCoord = TexCoord;
  gl_Position = pos;
}

uniform sampler2D tex;

in vec2 fs_TexCoord;
out vec4 Color;
void Fragment() {
  Color = texture(tex, fs_TexCoord);
}
Demo

Eigi the Raptor

Thanks to Manuel Wieser for the Eigi Model,
http://www.manuelwieser.com/
Advanced Topics

Layered rendering (stereo, cubemap)

Geometry composition

Tessellation

Specialization (e.g. known to have no texture)

Unification (runtime switch)
Geometry Composition

Divide

Shrink

Extrude
Layered Rendering (stereo)
Takeaways

Shader modules are very flexible

Generated code remains readable

Powerful operations (layered rendering, instancing, etc.)

First class representation for shaders

Can be implemented using virtually any AST as input
Further reading

- Abstract Shade Trees, McGuire et al. 2006,
  https://cs.brown.edu/~sk/Publications/Papers/Published/mspk-abstract-shade-trees/paper.pdf
- Spark: Modular, Composable Shaders for Graphics Hardware,
- A System for Rapid Exploration of Shader Optimization Choices, He et al, 2016,
  http://graphics.cs.cmu.edu/projects/spire/
- Shader Components: Modular and High Performance Shader Development, He et al.
  http://graphics.cs.cmu.edu/projects/shadercomp/he17_shadercomp.pdf
- CoSMo, Intent-based Composition of Shader Modules, Haaser et al.
- LibSH, Shader algebra, McCool et al. 2004,
Outlook and Status

Rendering Engine internals (this LV)

- Materials, Lighting (e.g. Lightmap generation)
- Scene representation and data-structures
- Notation of renderable things
- Graphics API and Low Level Optimizations
- GPU

Upcoming lectures

Recent topics