Shading Framework for Modern Rendering Engines

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

Motivation

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

Numerous shader files
Undesirable 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 shader pipeline

Modularity? Algorithms are preprogrammed to according pipeline stages

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
- Documentation of low-level compiler details
- New Shading examples
- Evaluation of the new back-end

Results

<table>
<thead>
<tr>
<th>Method</th>
<th>GLSL (fps)</th>
<th>Spark (fps)</th>
<th>% Spark vs. GLSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Shading (1920)</td>
<td>112,158</td>
<td>109,665</td>
<td>+2.25</td>
</tr>
<tr>
<td>Cube mapping (1.4k)</td>
<td>159,514</td>
<td>149,544</td>
<td>+6.67</td>
</tr>
<tr>
<td>Quad Tessellation (1.6k)</td>
<td>279,178</td>
<td>269,178</td>
<td>+3.66</td>
</tr>
</tbody>
</table>

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