Qt 3D Node Editor and Shader Generator

Paul Lemire – paul.lemire@kdab.com

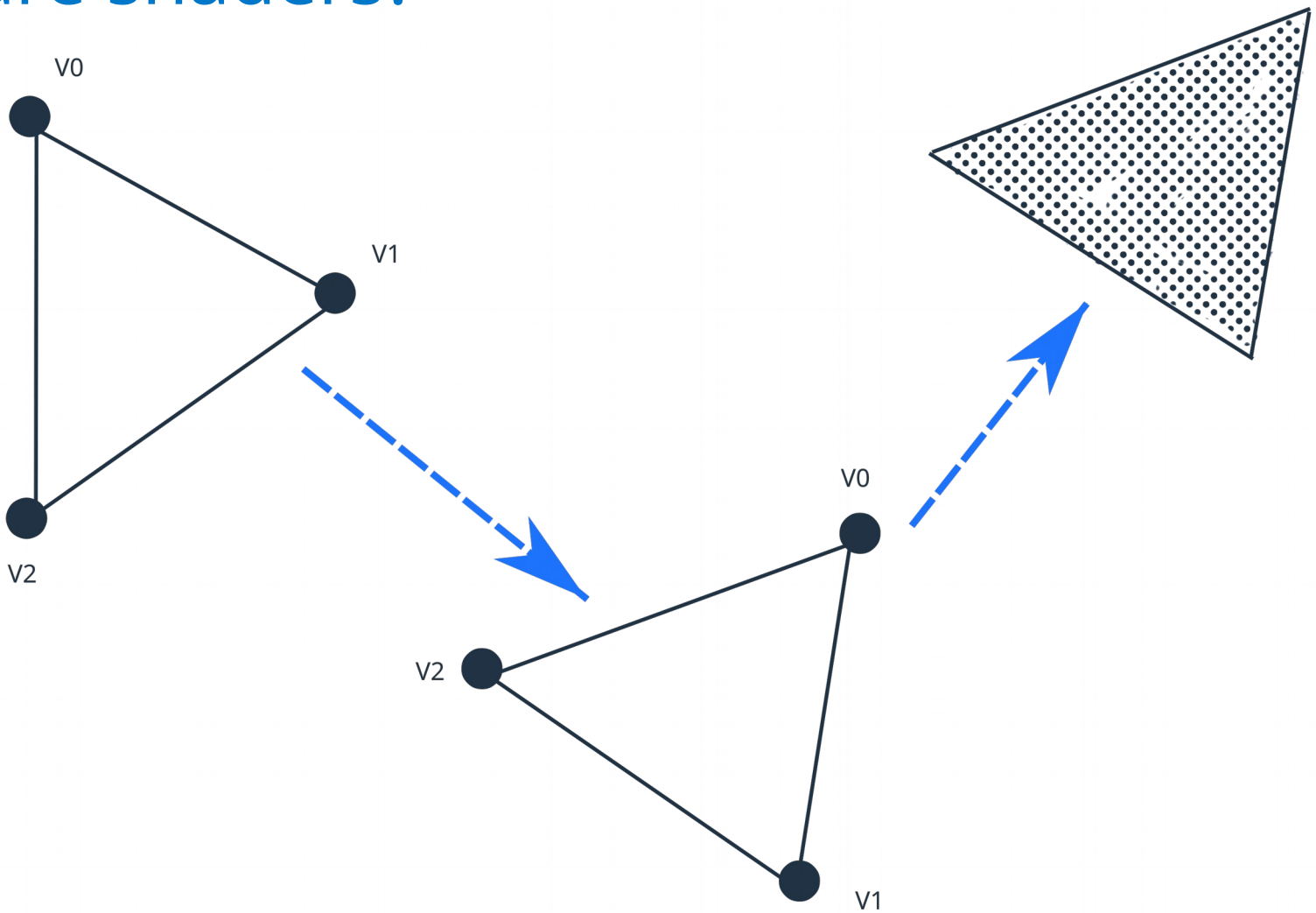
KDAB

A quick recap about shaders

What are shaders?

- A program that runs on the GPU
- Different types of shaders
  - Vertex → transforms points in space
  - Fragment → compute pixels' colors
  - Geometry, Compute, Tessellation …
- Written in different programming languages depending on the Graphics API in use
  - OpenGL uses a language called GLSL
  - DirectX uses a language called HLSL

What are shaders?



What does a shader look like?

```
#version 150
in vec3 vertexPosition;
uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4(vertexPosition, 1.0);
}
```

Vertex Shader

```
#version 150 core
out vec4 fragColor;


void main()
{
    fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Fragment Shader

In Practice

Using shaders in Qt 3D

```
Material {
  effect: Effect {
    techniques: [
      Technique {
        // Specify the Graphics API and Version we target
        graphicsApiFilter {
          api: GraphicsApiFilter.OpenGL
          profile: GraphicsApiFilter.CoreProfile
          majorVersion: 4; minorVersion: 4
        }
        renderPasses: RenderPass {
          shaderProgram: ShaderProgram {
            vertexShaderCode: loadSource("qrc:/shaders/phong.vert")
            fragmentShaderCode: loadSource("qrc:/shaders/phong.frag")
          }
        }
      }
    ]
  }
}
```

Shaders with OpenGL

- Multiple desktop versions (GL 2.*, GL 3.*, GL 4.*)
- and embedded versions (ES 2, ES 3.*)
  - Versions don’t all support the same features or use the same exact syntax
  - If you want to support multiple GL versions, you need to provide shader code for each version
- OpenGL expects shaders to be provided as GLSL code*
- The OpenGL Driver takes care of compiling the GLSL code to a program that can be executed by the GPU

Shaders with Vulkan

- Vulkan expects SPIR-V shaders
- SPIR-V is a bytecode
- The glslang tool convert shaders written in various languages (C++, GLSL, OpenCL) to SPIR-V
  - Shader compilation is expected to be a step that takes place at application build time rather than runtime

Handling multiple APIs/versions

- Two options:
  - Provide a shader for each version we target
    - More assets to handle
    - Selection is made at runtime based on which rendering backend was selected
    - Makes it hard to test all possible versions
  - Abstract the shader code into a set of inputs, outputs and operations
    - Provide translation rules for input, output, operations
    - Convert shader code description into actual shader code

KDAB

Abstracting shader code with nodes

The Node Editor

- Builds a graph of nodes
  - Nodes can either be
    - An input
    - An output
    - An operation/function
- Exports .graph files which contains the graph structure + node prototypes
- Part of Kuesa / available as QtCreator plugin

Prototypes and translations

- The prototype is the definition of a specific node
- Translations define how a node has to be converted
- The prototype specifies:
  - Whether the node is an input, output or operation
  - If node is an operation, the number of inputs/outputs
  - Translations for each Graphics API that needs to be supported
  - Header declaration (for uniforms, includes ...)

Simple Prototypes

```
"add": {
  "inputs": ["first", "second"],
  "outputs": ["sum"],
  "parameters": {
    "type": { "type": "QShaderLanguage::VariableType", "value": "QShaderLanguage::Vec3"}
   },
  "rules": [
    {
      "format": { "api": "OpenGLES", "major": 2,"minor": 0},
      "substitution": "highp $type $sum = $first + $second;"
    },
    {
      "format": { "api": "OpenGLCoreProfile", "major": 3, "minor": 0},
      "substitution": "$type $sum = $first + $second;"
    }
  ]
}
```

More complex Prototypes

```
"customFunction": {
  "inputs": ["first", "second"],
  "outputs": ["result"],
  "parameters": {
    "type": { "type": "QShaderLanguage::VariableType", "value": "QShaderLanguage::Vec3"}
   },
  "rules": [
    {
      "format": { "api": "OpenGLCoreProfile", "major": 3,"minor": 0 },
      "headerSnippets": [
        "#pragma include :shaders/es2/myCustomFunction.inc.frag"
       ],
       "substitution": "vec3 $result = myCustomFunction($first, $second);"
    }
  ]
}
```

Layers

- Not to be confused with Qt 3D Layers
- Allows to create different views of a given graph
  - To handle different type of inputs

KDAB

Loading graphs with Qt 3D

The Qt, OpenGL and C++ experts

QShaderProgramBuilder

- Recreates shader code by traversing the graph
- Selects translations that match the rendering backend
- Relies on QShaderGenerator (private API of QtGui)
- Does some optimizations:
  - Cache results of operations which are referenced more than once
  - Inlines operations otherwise

Using shaders in Qt 3D

```
Material {
  effect: Effect {
    techniques: [
      Technique {
        GraphicsApiFilter { ... }
        renderPasses: RenderPass {
          shaderProgram: ShaderProgram {
            id: prog
            ShaderProgramBuilder {
              shaderProgram: prog
              fragmentShaderGraph:"qrc:/shaders/graphs/graph.frag.json"
              enabledLayers: []
            }
            ShaderProgramBuilder {
              shaderProgram: prog
              fragmentShaderGraph:"qrc:/shaders/graphs/graph.vert.json"
            }
          }
        }
      }
    ]
  }
}
```

KDAB

What’s next?

Extending the use of graphs to more than shaders

- FrameGraph
- LogicalDevice
- Particle Systems
- …

Generate Shader Bytecode

- Would allow to create SpirV byte code
- Required for Vulkan / RHI

KDAB

Questions?