GLSL vs HLSL

Marko Täht
Topics:
• What is a shader?
• Popular shading languages
• GLSL
• HLSL
• GLSL vs HLSL
What is a shader?

- Computer program
- Tells your computer how to draw something in a specific and unique way
- Usually for GPU
- Modern use was introduced by Pixar in May 1988
- First graphics card with programmable pixel shader was Nvidia GeForce 3 (2000)
- Hardware evolved toward unified shader model
Unified Shader Model

All stages of in the rendering pipeline have the same capabilities. They can All read textures and buffers and Instructions sets are identical.

Please note, that the "3D engines" in unified-shader GPUs do still contain fixed function units, e.g. for tessellation! To truly understand the graphics pipeline as defined by OpenGL 4.5, look up the corresponding specification!

The elemental design of the pipeline itself is of course dictated by the algorithm to do the rendering. Rendering is done either by rasterizing OR by ray-tracing. A graphics API does additionally define some pipeline, to facilitate the inter-working of GPUs, the graphics device drivers and the application making use of them for calculations. The most recent APIs (Direct3D 12 (by Microsoft), Mantle (by AMD) and Vulkan (by Khronos)) don’t define any graphics pipeline to be used!
Popular shading languages

• Two main rendering methods: offline rendering, real-time rendering

• Offline rendering has no time constraint and more complex shading techniques can be used to get more realistic end result. Images are pre-processed (pre-calculated) and then they can be assembled to like a video clip or other uses.

• Real-time rendering needs to give result when you ask for it. The time window is ~20 ms. This eliminates many solutions that can be used. Many tricks are used to make it seem realistic.
• Offline rendering shading languages:
  • RenderMan Shading Language – offers 6 different shaders: light source shader, surface shader, displacement shader, deformation shader, volume shader, image shader. Most commonly used for production quality rendering.
  • Houdini VEX shading language – modelled after RSL. Integrated into 3D package giving shader language access to shaders.
  • Gelato Shading language – modelled after RSL. Differences mainly syntactical.
  • Open shading language – developed by Sony Pictures Imageworks for use in their engine. Also used in Blender’s Cycles renderer engine. Allows importance sampling. Good for physical-based rendering
• Real-time rendering shading language:
  • ARB Assembly language – established in 2002 as a standard low-level instruction set for programmable GPU-s by OpenGL architecture review board. High-level OpenGL shading languages often compile into ARB.
  • OpenGL shading language - GLSL
  • DirectX Shader Assembly language – used in Direct3D 8 and 9. Direct representation of intermediate shader bytecode which is passed to graphics driver.
  • DirectX High-Level Shading Language – HLSL
  • Cg programming language – API independent. Compiles into GLSL and HLSL. Depreciated since 2012.
Shader languages mostly resemble C language. (Except the assembly ones)
GLSL

• Preceded by ARM assembly language.
• Unifies vertex and fragment processing in a single instruction set, allowing conditions and branches.
• Originally done in ARM assembly language, but was too complex and unintuitive
• Introduced in OpenGL 1.4, included in OpenGL 2.0
• Cross platform support: Linux, Mac, Windows
• Ability to write code that is supported by any graphic card that support GLSL
• Each hardware vendor includes GLSL compiler
• WebGL – browser support of OpenGL
• GLSL Hello world shaders

void main() { gl_Position = ftransform(); }

void main() { gl_FragColor = vec4(0.4,0.4,0.8,1.0); }

https://www.opengl.org/sdk/docs/tutorials/TyphoonLabs/Chapter_1.pdf
HLSL

- Has five shaders: pixel(fragment), vertex, geometry, compute, tessellation
- Geometry shader takes vertices of primitive and uses this data to generate/degenerate additional primitives to send to rasterizer
- Compute shader is used to compute arbitrary information and is not used directly in drawing triangles and pixels. It has no user defined inputs and outputs. Compute shader has to fetch the data itself.
• HLSL hello world

```cpp
sampler2D tex0;

float4 pixelShader( float2 texCoord : TEXCOORD0,
float4 color : COLOR0 ) : COLOR0
{
    return float4(0,1,0,0);
}
```

http://rbwhitaker.wikidot.com/first-shader

http://www.gamasutra.com/view/feature/1812/creating_a_postprocessing.php?print=1
GLSL vs HLSL

- Unreal engine and Unity used HLSL, Webapps and game maker studio uses GLSL
- Syntax is very similar
- There are some compilers to convert between these 2

- GL gives lower access to synchronization
- HL code is faster because the fxc compilator is very aggressive
- HL is sometimes too much optimized (precision errors)
• HL compilation time is in seconds, GL in milliseconds
• Texture management on GL is full of bugs and issues while HL has well defined rules and validation layers
• On intel cards HL is better for compatibility and older hardware is more likely to run without glitches
• HL might have weird glitches that are not present in GL
• On linux GL has better compatibility
• HL allows to use 12 different samplers and 128 bound textures per shader. GL is limited to 16 -32 depending on driver and GPU
Press F1 to toggle help
Press F2 to toggle animation. [On]
Press F3 to show/hide animated objects. [On]
Press F4 to show/hide palette of spheres. [On]
• https://anteru.net/blog/2016/mapping-between-hlsl-and-glsl/
vertex shader

```cpp
varying vec4 foo
varying vec4 bar;

void main()
{
  foo = ...
  bar = ...
}
```

fragment shader

```cpp
varying vec4 foo
varying vec4 bar;

void main()
{
  gl_FragColor = foo * bar;
}
```

vertex shader

```cpp
struct VS_OUTPUT
{
  float4 foo : TEXCOORD3;
  float4 bar : COLOR2;
}

VS_OUTPUT whatever()
{
  VS_OUTPUT out;
  out.foo = ...
  out.bar = ...
  return out;
}
```

pixel shader

```cpp
void main(float4 foo : TEXCOORD3,
  float4 bar : COLOR2) : COLOR
{
  return foo * bar;
}
```

```cpp
struct vsInput
{
  float4 Pos0 : POSITION;
  float3 Norm : NORMAL;
  float4 TexCd : TEXCOORD0;
};

struct vsOut
{
  float4 PosWVP : SV_POSITION;
  float4 TexCd : TEXCOORD0;
  float3 NormView : NORMAL;
};

vsOut VS(vsInput input)
{
  //Do you processing here
}
```

And a pixel shader like this:

```cpp
struct psInput
{
  float4 PosWVP : SV_POSITION;
  float4 TexCd : TEXCOORD0;
};
```
• http://wiki.unity3d.com/index.php/Getting_Started_with_Shaders
Videos

• https://www.youtube.com/watch?v=HC3JGG6xHN8
• https://www.youtube.com/watch?v=cNDG1lhzcQ4
• https://www.youtube.com/watch?v=hL9iml4k8I8
- https://github.com/walbourn/directx-sdk-samples/tree/master/FluidCS11
- http://www.rastertek.com/dx11tut38.html
- http://www.humus.name/Articles/Persson_LowlevelShaderOptimization.pdf