GLSL vs HLSL

Marko Täht

Topics:

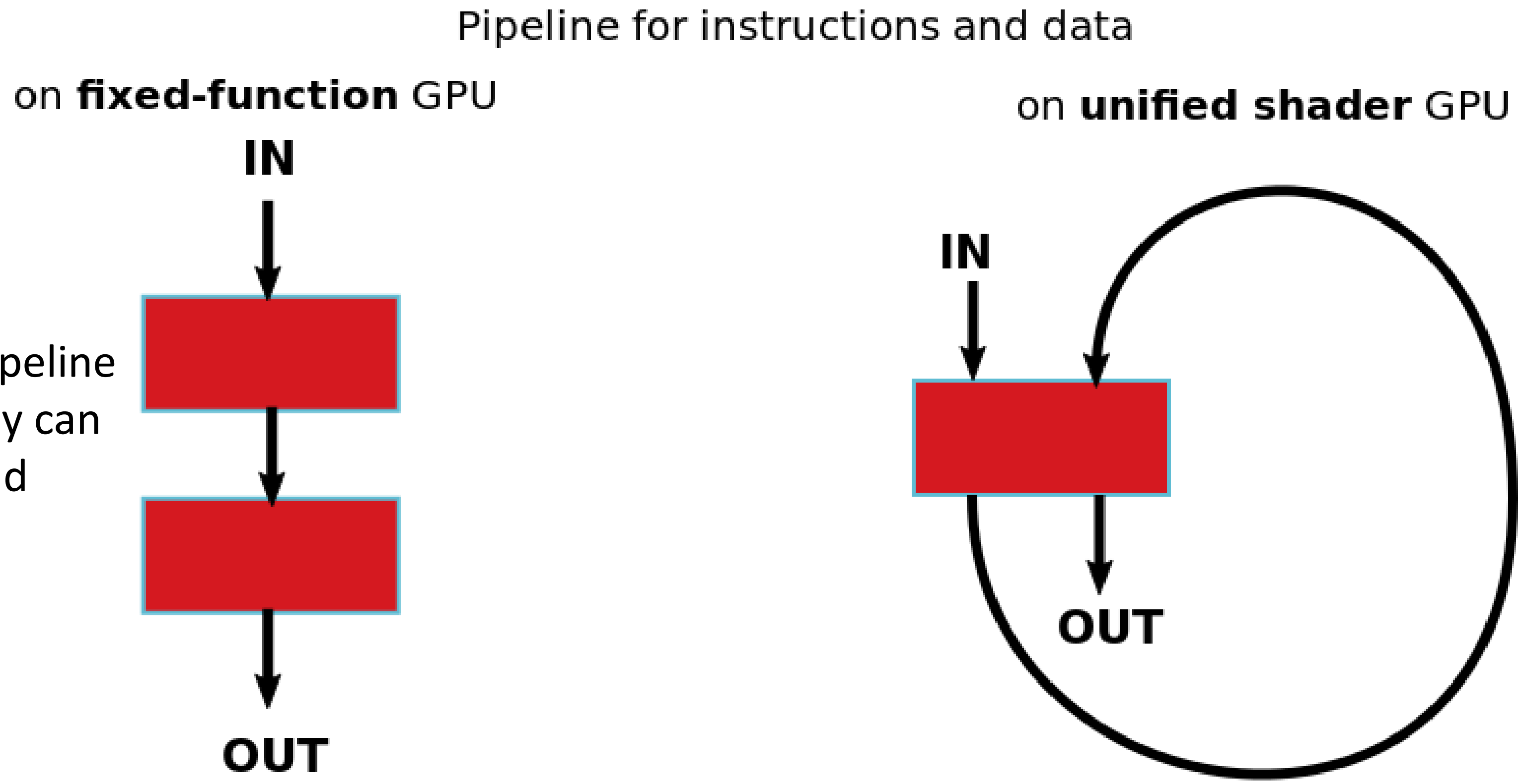
- What is a shader?
- Popular shading languages
- GLSL
- HLSL
- GLSL vs HLSL

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

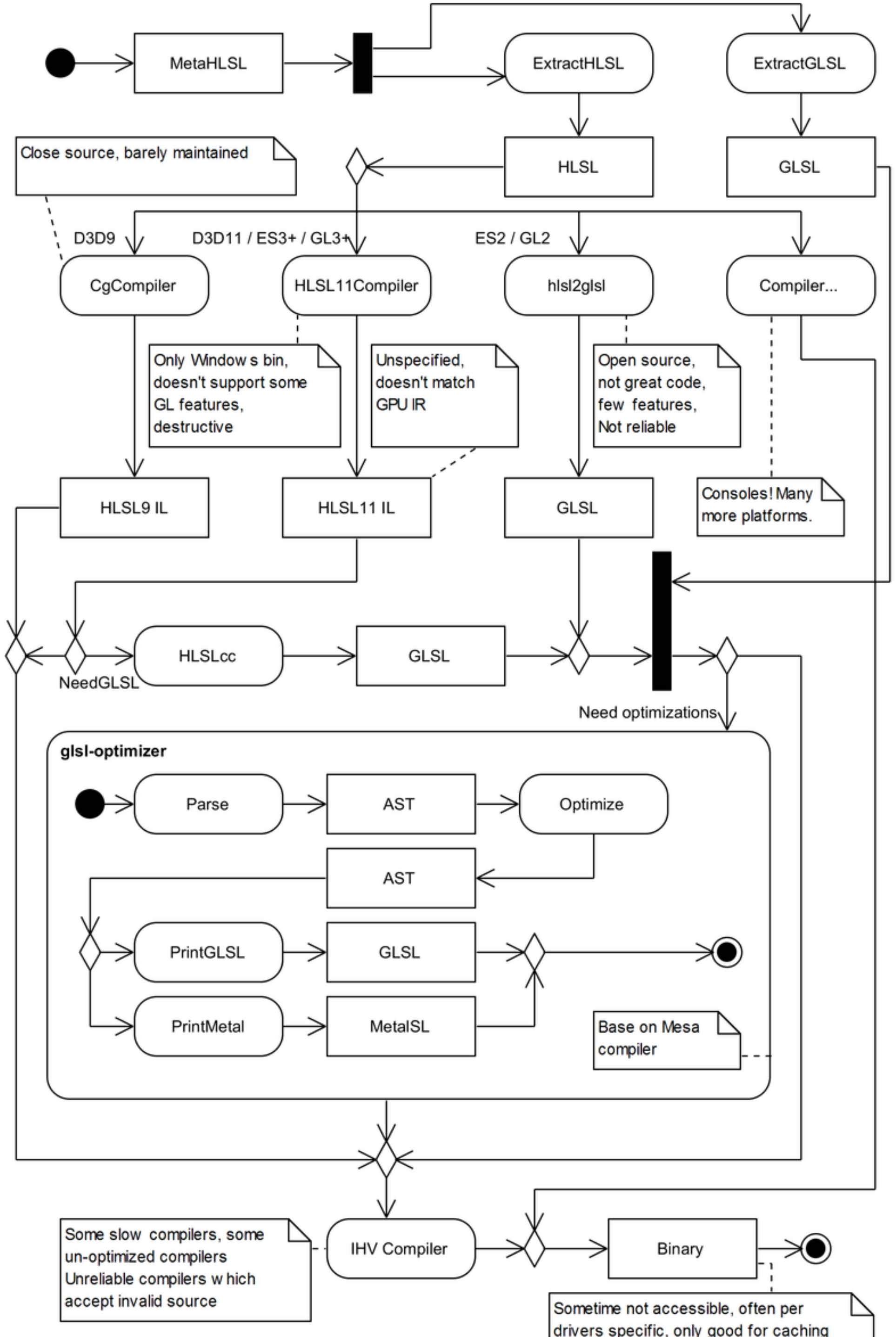
All stages of in the rendering pipeline have the same capabilities. They can All read textures and buffers and Instructions sets are identical.



Pipeline for instructions and data

Please note, that the "3D engines" in unified-shader GPUs do still contain fixed function units, e.g. for tesselation! 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!



MetaHLSL

ExtractHLSL

ExtractGLSL

HLSL

GLSL

Close source, barely maintained

D3D9

D3D11 / ES3+ / GL3+

ES2 / GL2

CgCompiler

HLSL11Compiler

hlsl2glsl

Compiler...

Only Window s bin,
doesn't support some
GL features,
destructive

Unspecified,
doesn't match
GPU IR

Open source,
not great code,
few features,
Not reliable

HLSL9 IL

HLSL11 IL

GLSL

Consoles! Many
more platforms.

HLSLcc

GLSL

NeedGLSL

Need optimizations

glsl-optimizer

Parse

AST

Optimize

AST

PrintGLSL

GLSL

PrintMetal

MetalSL

Base on Mesa
compiler

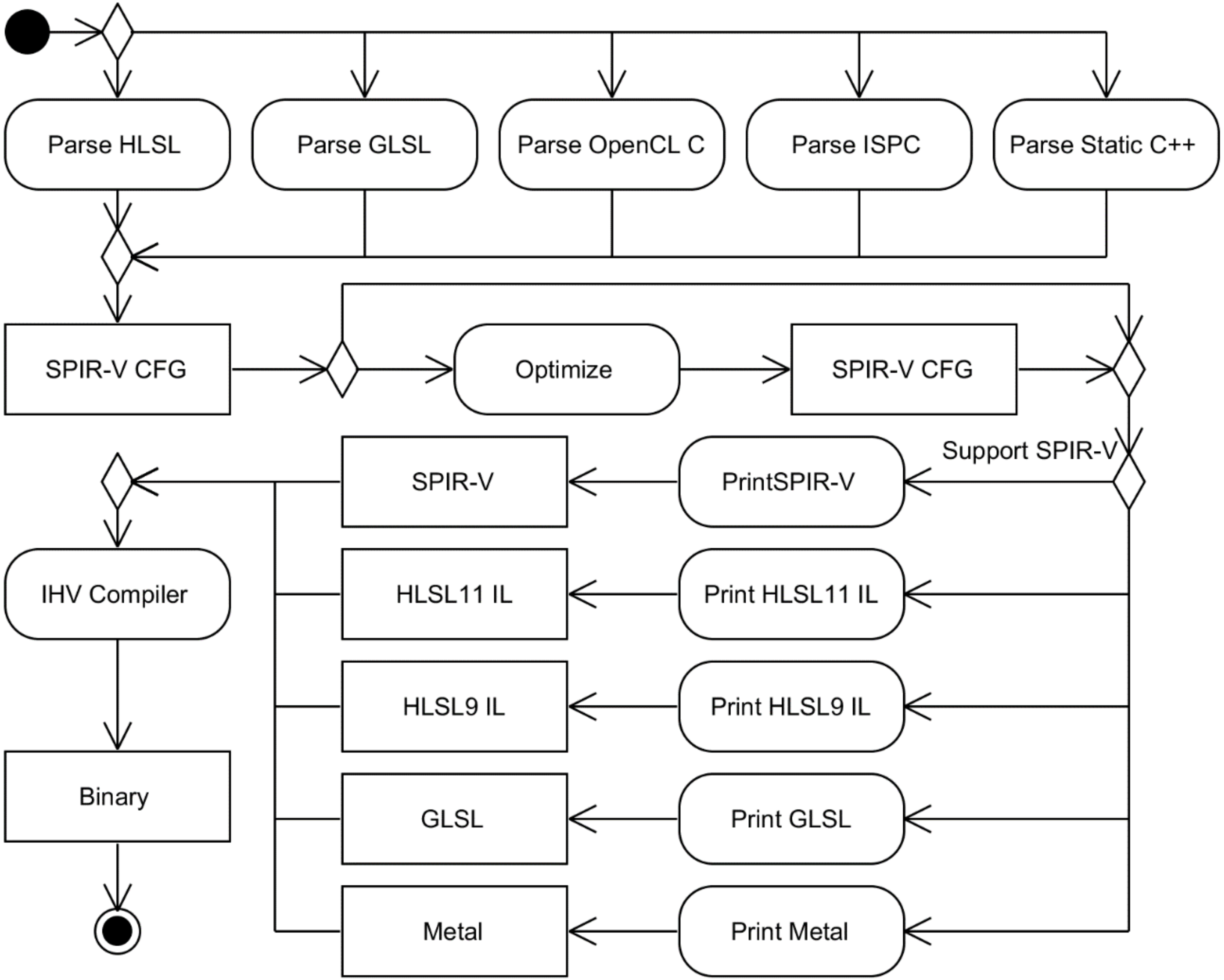
Some slow compilers, some
un-optimized compilers
Unreliable compilers w hich
accept invalid source

IHV Compiler

Binary

Sometime not accessible, often per
drivers specific, only good for caching

https://www.g-truc.net/post-0714.html


Parse HLSL
Parse GLSL
Parse OpenCL C
Parse ISPC
Parse Static C++
SPIR-V CFG
Optimize
SPIR-V CFG
Support SPIR-V
SPIR-V
PrintSPIR-V
IHV Compiler
HLSL11 IL
Print HLSL11 IL
HLSL9 IL
Print HLSL9 IL
Binary
GLSL
Print GLSL
Metal
Print Metal

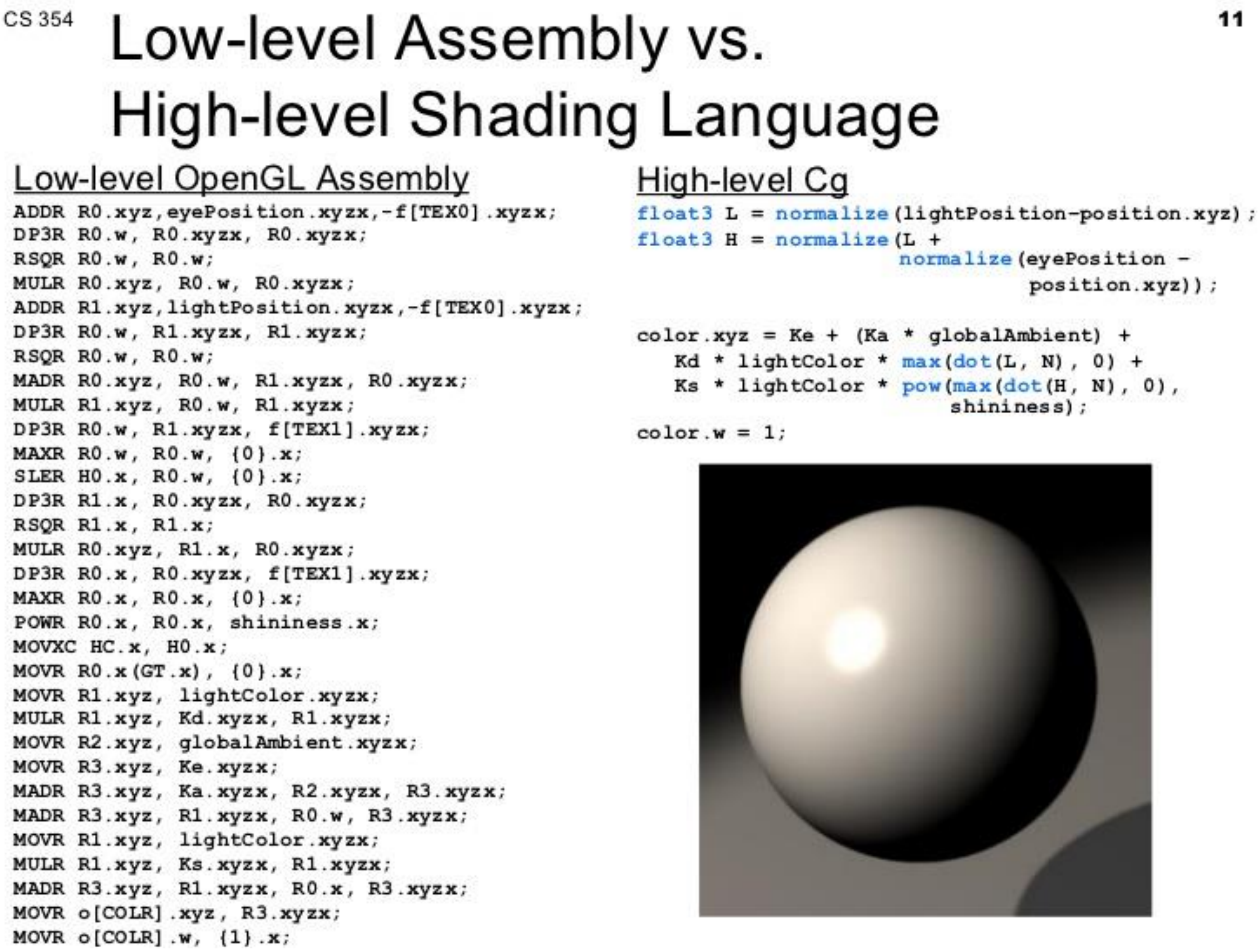
Popular shading languages

- 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 whitch is passed to graphics driver.
  - DirectX High-Level Shading Language – HLSL
  - Cg programming language – API independent. Compiles into GLSL and HLSL. Depricated since 2012.

Shader languages mostly resembles C language. (Except the assembly ones)



Low-level Assembly vs. High-level Shading Language

Low-level OpenGL Assembly

```
ADDR R0.xyz,eyePosition.xyzx,-f[TEX0].xyzx;
DP3R R0.w, R0.xyzx, R0.xyzx;
RSQR R0.w, R0.w;
MULR R0.xyz, R0.w, R0.xyzx;
ADDR R1.xyz,lightPosition.xyzx,-f[TEX0].xyzx;
DP3R R0.w, R1.xyzx, R1.xyzx;
RSQR R0.w, R0.w;
MADR R0.xyz, R0.w, R1.xyzx, R0.xyzx;
MULR R1.xyz, R0.w, R1.xyzx;
DP3R R0.w, R1.xyzx, f[TEX1].xyzx;
MAXR R0.w, R0.w, {0}.x;
SLER H0.x, R0.w, {0}.x;
DP3R R1.x, R0.xyzx, R0.xyzx;
RSQR R1.x, R1.x;
MULR R0.xyz, R1.x, R0.xyzx;
DP3R R0.x, R0.xyzx, f[TEX1].xyzx;
MAXR R0.x, R0.x, {0}.x;
POWR R0.x, R0.x, shininess.x;
MOVXC HC.x, H0.x;
MOVR R0.x(GT.x), {0}.x;
MOVR R1.xyz, lightColor.xyzx;
MULR R1.xyz, Kd.xyzx, R1.xyzx;
MOVR R2.xyz, globalAmbient.xyzx;
MOVR R3.xyz, Ke.xyzx;
MADR R3.xyz, Ka.xyzx, R2.xyzx, R3.xyzx;
MADR R3.xyz, R1.xyzx, R0.w, R3.xyzx;
MOVR R1.xyz, lightColor.xyzx;
MULR R1.xyz, Ks.xyzx, R1.xyzx;
MADR R3.xyz, R1.xyzx, R0.x, R3.xyzx;
MOVR o[COLR].xyz, R3.xyzx;
MOVR o[COLR].w, {1}.x;
```

High-level Cg

```
float3 L = normalize(lightPosition-position.xyz);
float3 H = normalize(L +
                normalize(eyePosition -
                          position.xyz));

color.xyz = Ke + (Ka * globalAmbient) +
   Kd * lightColor * max(dot(L, N), 0) +
   Ks * lightColor * pow(max(dot(H, N), 0),
                         shininess);

color.w = 1;
```

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

```
sampler2D tex0;

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

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

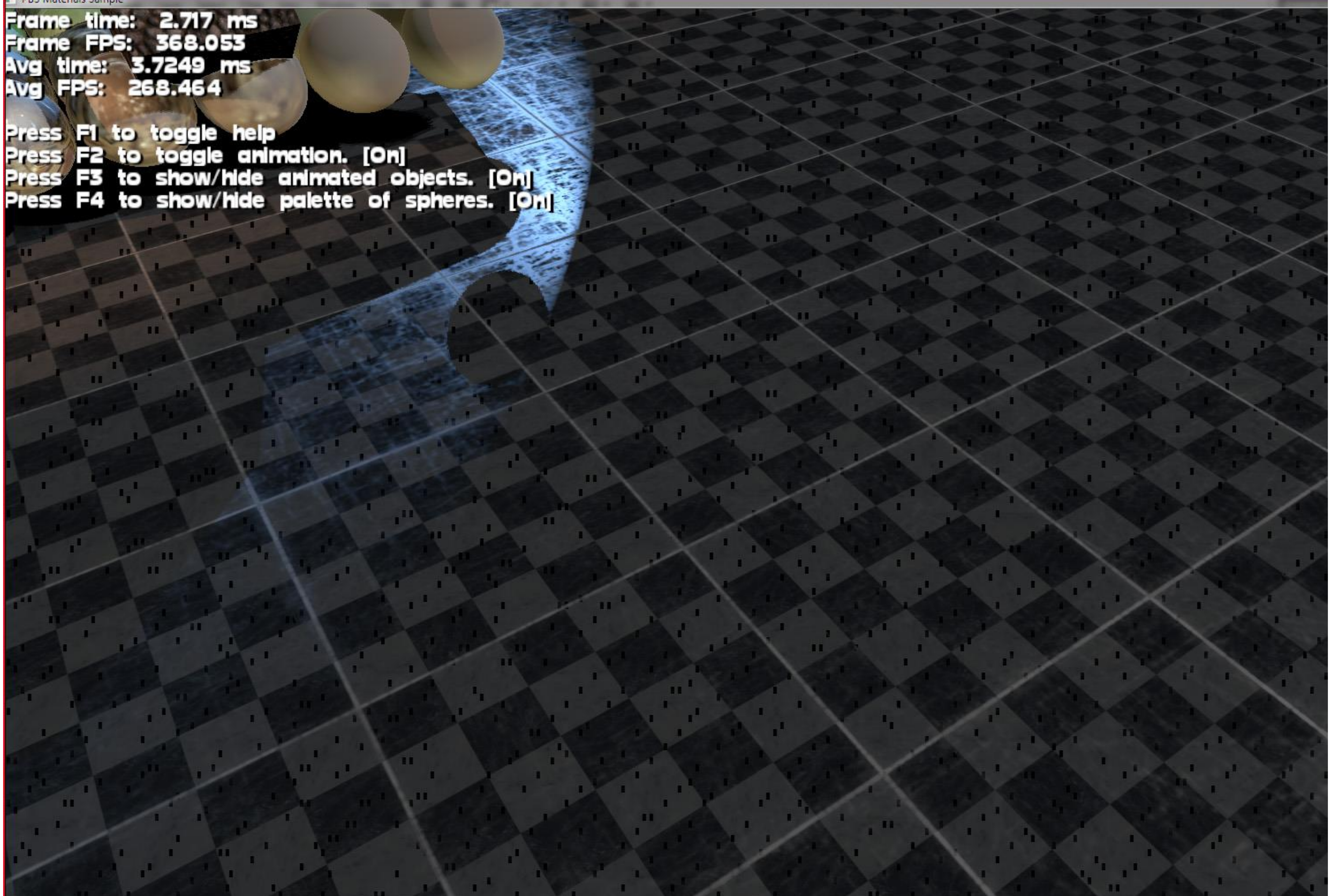
http://www.gamasutra.com/view/feature/1812/creating_a_postprocessing_.php?print=1

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

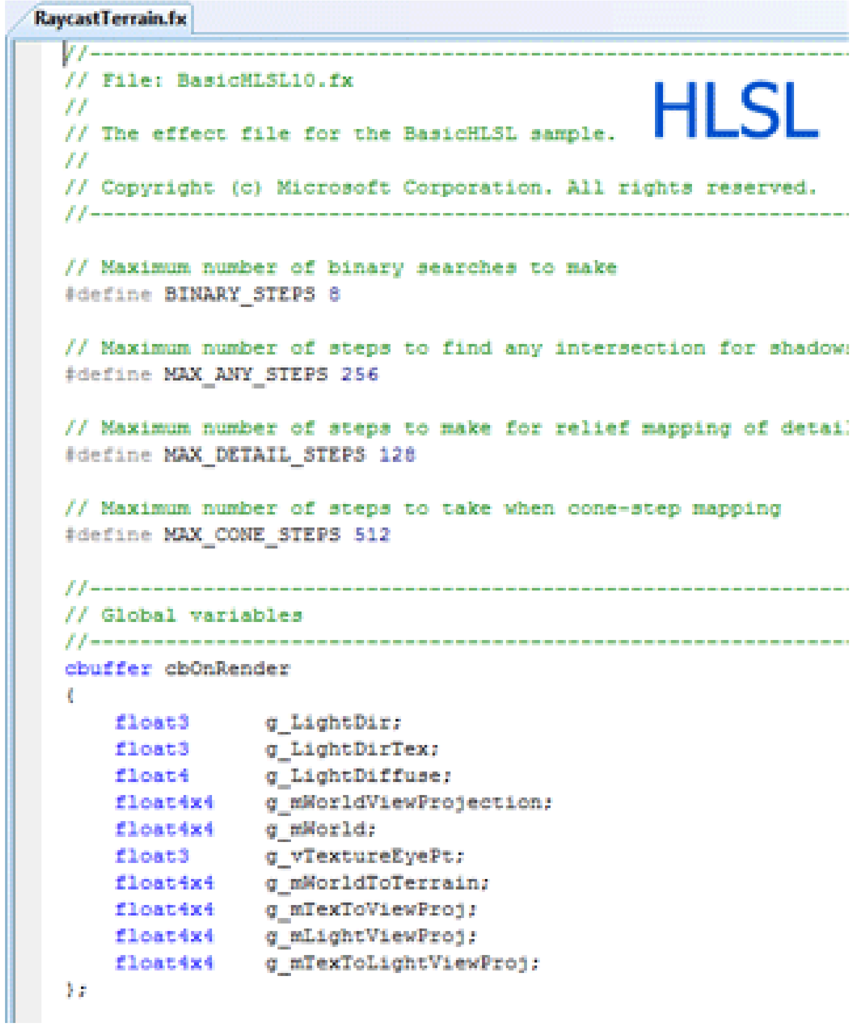
- GL gives lower access to synchronization
- HL code is faster because the fxc compilator is very aggressive
- HL is sometimes too much optimized (precision errors)

- 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

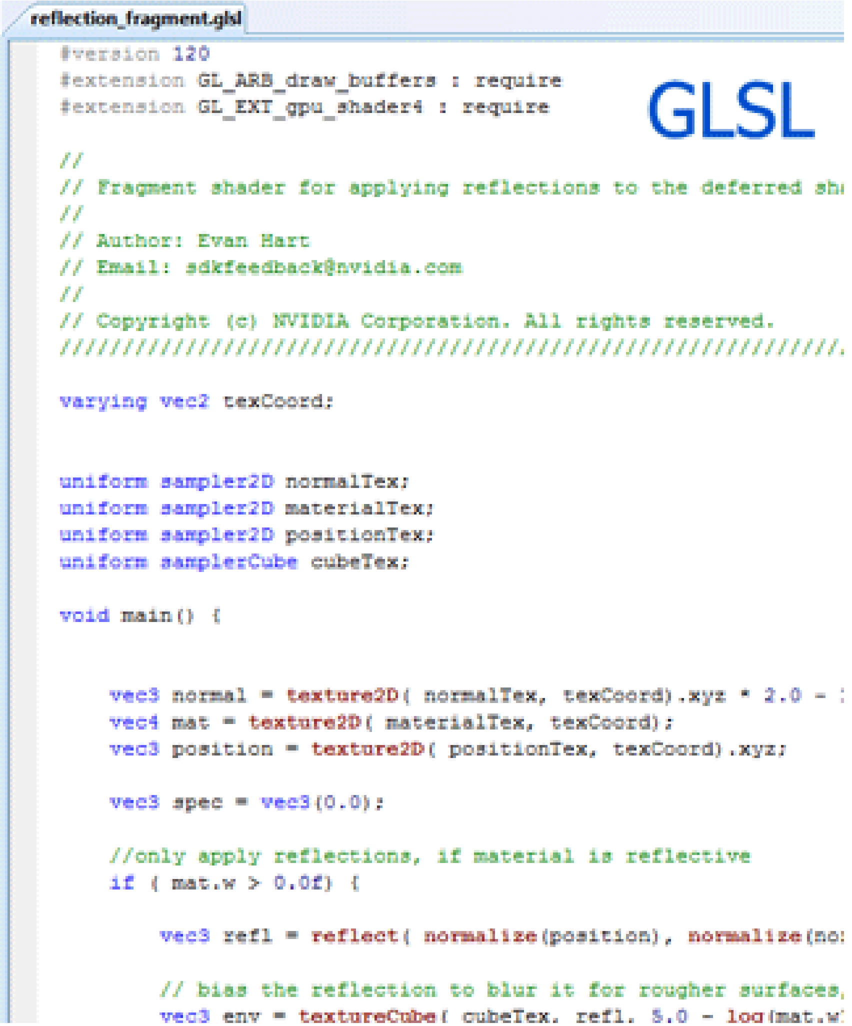


```
Frame time:  2.717 ms
Frame FPS:  368.053
Avg time:  3.7249 ms
Avg FPS:  268.464

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]
```

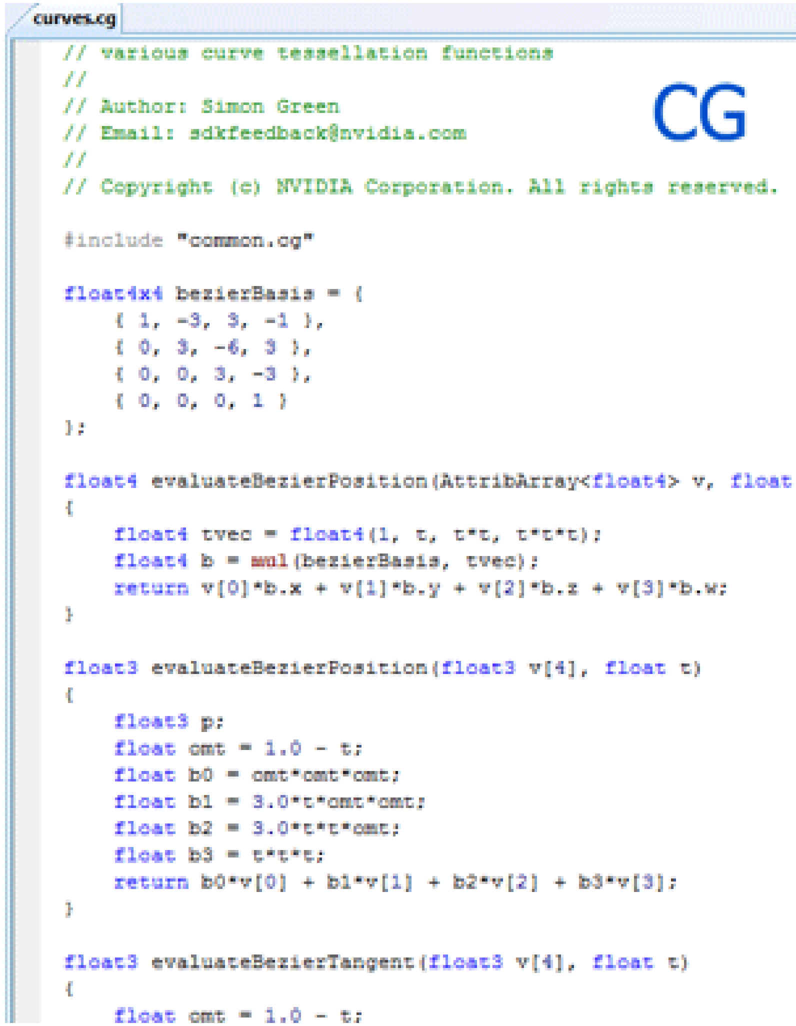
- https://anteru.net/blog/2016/mapping-between-hlsl-and-glsl/



```
//-----------------------------------------------------------------
// File: BasicHLSL10.fx
//
// The effect file for the BasicHLSL sample.
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//-----------------------------------------------------------------

// Maximum number of binary searches to make
#define BINARY_STEPS 8

// Maximum number of steps to find any intersection for shadow:
#define MAX_ANY_STEPS 256

// Maximum number of steps to make for relief mapping of detai:
#define MAX_DETAIL_STEPS 128

// Maximum number of steps to take when cone-step mapping
#define MAX_CONE_STEPS 512

//-----------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------
cbuffer cbOnRender
{
    float3      g_LightDir;
    float3      g_LightDirTex;
    float4      g_LightDiffuse;
    float4x4    g_mWorldViewProjection;
    float4x4    g_mWorld;
    float3      g_vTextureEyePt;
    float4x4    g_mWorldToTerrain;
    float4x4    g_mTexToViewProj;
    float4x4    g_mLightViewProj;
    float4x4    g_mTexToLightViewProj;
};
```

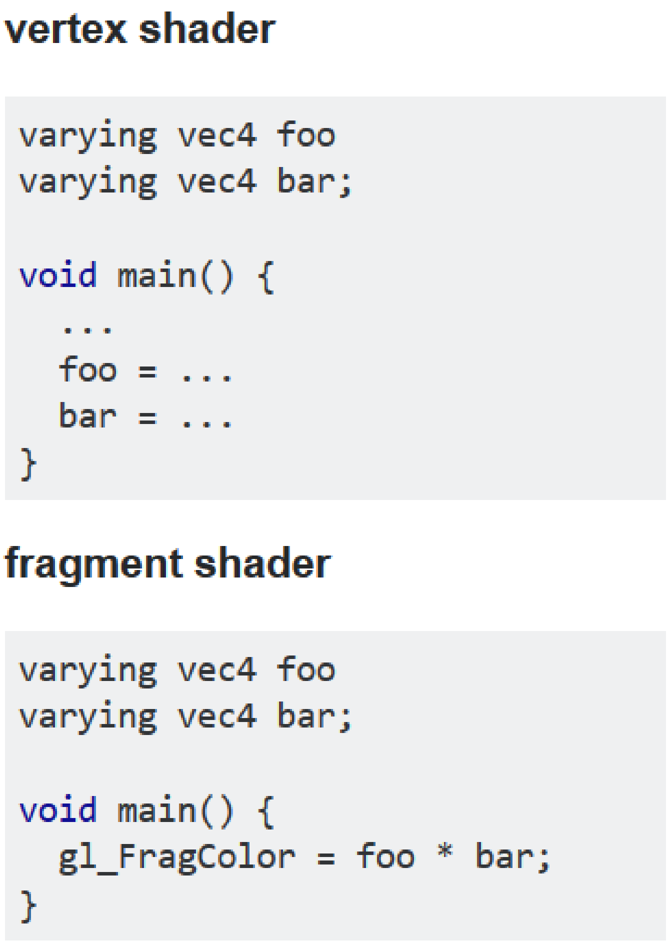


```
#version 120
#extension GL_ARB_draw_buffers : require
#extension GL_EXT_gpu_shader4 : require

//
// Fragment shader for applying reflections to the deferred sh
//
// Author: Evan Hart
// Email: sdkfeedback@nvidia.com
//
// Copyright (c) NVIDIA Corporation. All rights reserved.
//////////////////////////////////////////////////////////////////.

varying vec2 texCoord;


uniform sampler2D normalTex;
uniform sampler2D materialTex;
uniform sampler2D positionTex;
uniform samplerCube cubeTex;

void main() {


    vec3 normal = texture2D( normalTex, texCoord).xyz * 2.0 - :
    vec4 mat = texture2D( materialTex, texCoord);
    vec3 position = texture2D( positionTex, texCoord).xyz;

    vec3 spec = vec3(0.0);

    //only apply reflections, if material is reflective
    if ( mat.w > 0.0f) {

        vec3 refl = reflect( normalize(position), normalize(no:

        // bias the reflection to blur it for rougher surfaces,
        vec3 env = textureCube( cubeTex, refl, 5.0 - log(mat.w:
```



```
// various curve tessellation functions
//
// Author: Simon Green
// Email: sdkfeedback@nvidia.com
//
// Copyright (c) NVIDIA Corporation. All rights reserved.

#include "common.cg"

float4x4 bezierBasis = {
    { 1, -3, 3, -1 },
    { 0, 3, -6, 3 },
    { 0, 0, 3, -3 },
    { 0, 0, 0, 1 }
};

float4 evaluateBezierPosition(AttribArray<float4> v, float
{
    float4 tvec = float4(1, t, t*t, t*t*t);
    float4 b = mul(bezierBasis, tvec);
    return v[0]*b.x + v[1]*b.y + v[2]*b.z + v[3]*b.w;
}

float3 evaluateBezierPosition(float3 v[4], float t)
{
    float3 p;
    float omt = 1.0 - t;
    float b0 = omt*omt*omt;
    float b1 = 3.0*t*omt*omt;
    float b2 = 3.0*t*t*omt;
    float b3 = t*t*t;
    return b0*v[0] + b1*v[1] + b2*v[2] + b3*v[3];
}

float3 evaluateBezierTangent(float3 v[4], float t)
{
    float omt = 1.0 - t;
```

- https://docs.microsoft.com/en-us/windows/uwp/gaming/glsl-to-hlsl-reference

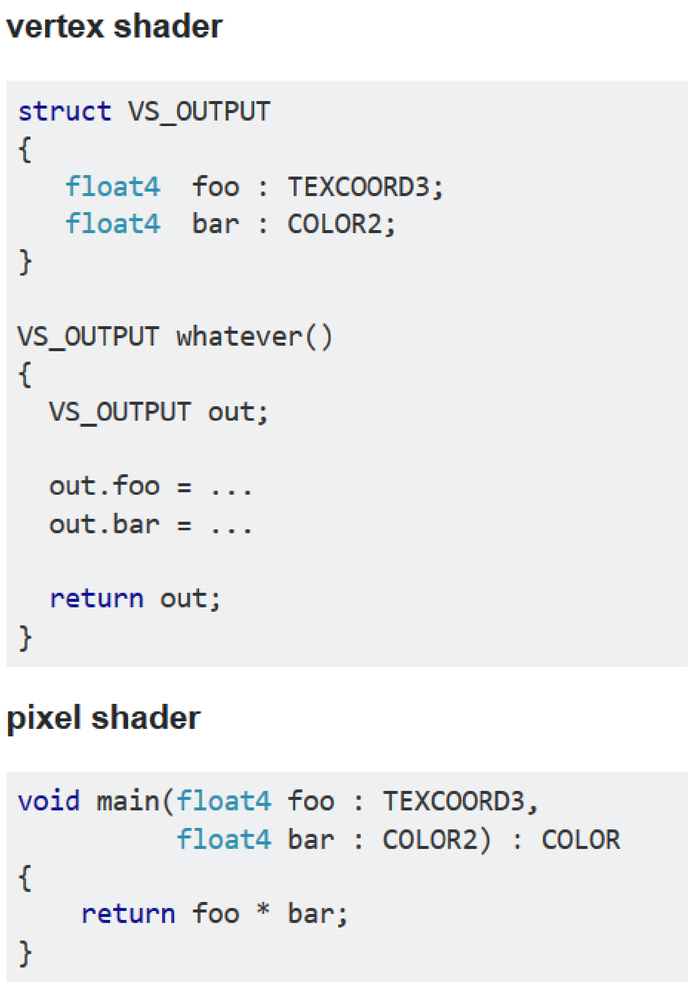


vertex shader

```
varying vec4 foo
varying vec4 bar;

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

fragment shader

```
varying vec4 foo
varying vec4 bar;

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

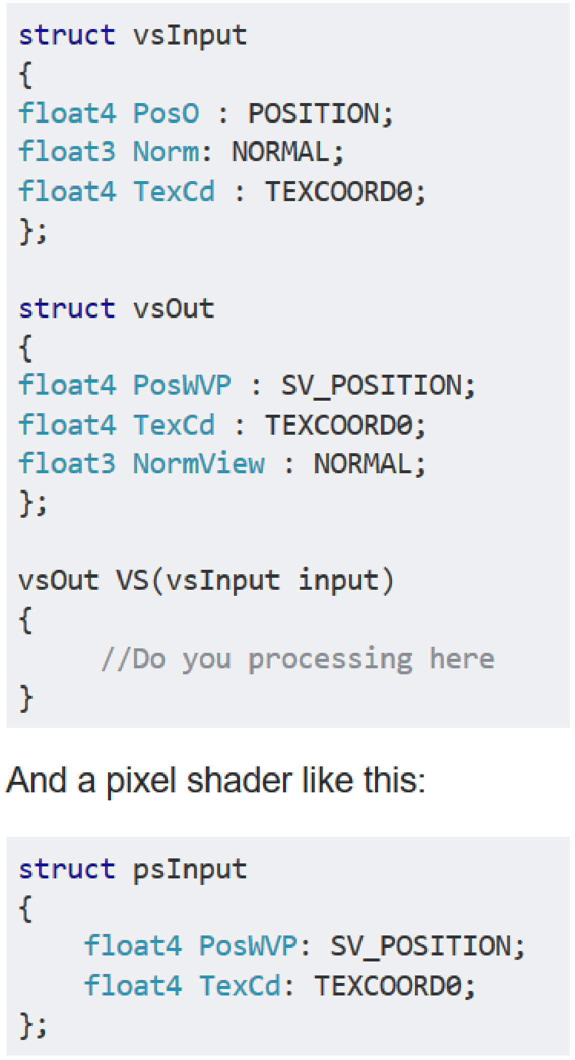


vertex shader

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

VS_OUTPUT whatever()
{
  VS_OUTPUT out;

  out.foo = ...
  out.bar = ...

  return out;
}
```

pixel shader

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



```
struct vsInput
{
float4 PosO : 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:

```
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://takinginitiative.wordpress.com/2011/01/12/directx10-tutorial-9-the-geometry-shader/
- https://github.com/walbourn/directx-sdk-samples/tree/master/FluidCS11
- http://www.rastertek.com/dx11tut38.html
- http://www.humus.name/Articles/Persson_LowlevelShaderOptimization.pdf