Modern OpenGL® Programming

Pursuing maximum performance/watt

Course Objectives

- **Leverage knowledge** of modern graphics hardware and APIs
- **Update perspectives** for those who still think in terms of OpenGL 1 or 2
- A glimpse of the **hardware/driver perspective**

Assumed Knowledge

- Modern graphics hardware
  - D3D9 / OpenGL 2 at a minimum
  - Will discuss OpenGL 4.4 features as well
- A modern graphics API
  - Graphics resource management concepts
  - Shader language concepts
- Intermediate real-time rendering knowledge
  - Will not motivate why data from one pass or shader stage would be used in another

Outline

- Quick start
  - The history, OpenGL core profile and debug context
- Shaders: programming the GPU
  - Pipeline overview, the geometry/tessellation/ compute shader introduction, the program pipeline and uniform buffer
- Textures and buffers
  - New texture interface, image-load-store and texture views.
- Last but not least
  - New draw calls, memory and command synchronization

Cross Platform and "Open"

- ARB and Khronos
- Specifications are public and free
  - http://www.opengl.org/registry
  - http://www.opengl.org/documentation/glsl
- Conformance Tests (past/future)
- Extensions
  - Vendor-specific
  - Multi-vendor
  - ARB

OpenGL History

- GL (SGI)
- OpenGL 1: Fixed Function
- OpenGL 2: Programmable
- OpenGL 3: DX10, Cleanup
- OpenGL 4: DX11(.1)

- OpenGL ES 1
- OpenGL ES 2
- OpenGL ES 3

- WebGL

Today: OpenGL 4.5

Contexts and Profiles

- Profiles: capability (almost like a device)
- Contexts: device context, a state collection

- Core profile
- Compatibility profile
- Debug context

Contexts and Profiles

- Profiles: capability (almost like a device)
- Contexts: a state collection

```
// If WGL_ARB_create_context is supported
const int attribs[] = {
   WGL_CONTEXT_MAJOR_VERSION_ARB, 4,
   WGL_CONTEXT_MINOR_VERSION_ARB, 3,
   WGL_CONTEXT_FLAGS_PROFILE_MASK_ARB,
          WGL_CONTEXT_CORE_PROFILE_BIT_ARB,
   WGL_CONTEXT_FLAGS_ARB, 0,
   0
};

HGLRC hGLRC = wglCreateContextAttribsARB( hDC, 0, attribs );
```

Core Contexts

- Introduced with OpenGL 3.2
- Core contexts *remove* legacy functionality
  - Substantial simplification
- Core contexts restrict OpenGL to features actually supported by hardware
- The driver should perform less error checking
  - Am I in the middle of a begin-end sequence?
  - Am I compiling a display list?
  - Do I need to validate my fixed-function shader?

Not In Core

- Immediate Mode
- Fixed-Function
  - glLight, glFog, glColor and etc
- GL Display Lists (not quite the same as DX)
- N-Gons
- Selection + Feedback, accumulation buffers, stipples, …

- Similar to OpenGL ES 2

Debug Context and Output

- Separate debug context for deliberate cost
- Basic use is a callback (no more glError)

```
void callback( enum source, enum type, uint id, enum
        severity, sizei length, const char *message,
        void *userParam );
```

- Or log per context
- Filtering and labelling
- Synchronous (easier) or asynchronous (fetch the log message)

Debug Context and Output

```
// If WGL_ARB_create_context is supported
const int attribs[] = {
   WGL_CONTEXT_MAJOR_VERSION_ARB, 4,
   WGL_CONTEXT_MINOR_VERSION_ARB, 3,
   WGL_CONTEXT_FLAGS_PROFILE_MASK_ARB, WGL_CONTEXT_CORE_PROFILE_BIT_ARB,
   WGL_CONTEXT_FLAGS_ARB, WGL_CONTEXT_DEBUG_BIT_ARB,
   0
};

// Create the context and make the context current
// …

if (/* ARB_debug_output is supported */)
{
    glDebugMessageCallbackARB(debugCallbackARB, NULL);
    glEnable(GL_DEBUG_OUTPUT_SYNCHRONOUS_ARB);
}
else if (/*AMD_debug_output is supported */)
{
    glDebugMessageCallbackAMD(debugCallbackAMD, NULL);
}
```

Object Label (since 4.3)

- Motivation: hard to identify OpenGL objects with their integer names.
- Solution: to assign a string name to OpenGL objects, i.e., FBO, texture, buffer, etc

```
void glObjectLabel(GLenum identifier, GLuint name,
    GLsizei length, const char* label);
```

Development Advice

Develop using core profile with debugging

- Try different vendor debug contexts
- Core gives you platform flexibility
- Core gives you higher performance
- Compatibility is still an option

Quick Start

- Context creation
  - OS-specific (WGL, …)
- Extensions
  - Needed for exposing newer versions too
  - OS-specific
  - Third Party (glew, gl3w, glee)
- Frameworks
  - freeglut
  - glfw
  - Game Engines (e.g., Ogre)
  - Rendering engines (e.g., OpenSceneGraph, Irrlicht)

Put It All Together

```
PIXELFORMATDESCRIPTOR pfd =
{
    sizeof (PIXELFORMATDESCRIPTOR),
    1,
    PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,
    PFD_TYPE_RGBA,
    32,                    // RGBA bits
    0, 0, 0, 0, 0, 0,
    0,                     // Alpha Buffer
    0,                     // Shift Bit Ignored
    0,                     // No Accumulation Buffer
    0, 0, 0, 0,            // Accumulation Bits Ignored
    24,                    // Z-Buffer (Depth Buffer)
    8,                     // Stencil Buffer
    0,                     // No Auxiliary Buffer
    PFD_MAIN_PLANE,        // Main Drawing Layer
    0,                     // Reserved
    0, 0, 0                // Layer Masks Ignored
};
int pixelFormat = ChoosePixelFormat(hDC, &pfd); // return false when returned pixelFormat is 0
SetPixelFormat(hDC, pixelFormat, &pfd);    // return false when SetPixelFormat fails
HGLRC tempContext = wglCreateContext(hDC); // return false when tempContext is 0
wglMakeCurrent(hDC, tempContext);

glewInit();   // return false when GLEW init fails
```

Put It All Together

```
if (wglewIsSupported("WGL_ARB_create_context"))
{
    wglMakeCurrent(NULL, NULL);
    wglDeleteContext(tempContext);
    ReleaseDC(hWnd, hDC);
    DestroyWindow(hWnd);
    // Recreate the window (renew hWnd and hDC)

    int attributes[] =
    {
        WGL_DRAW_TO_WINDOW_ARB, GL_TRUE,
        WGL_SUPPORT_OPENGL_ARB, GL_TRUE,
        WGL_ACCELERATION_ARB, WGL_FULL_ACCELERATION_ARB,
        WGL_COLOR_BITS_ARB, 32,
        WGL_ALPHA_BITS_ARB, 0,
        WGL_DEPTH_BITS_ARB, 24,
        WGL_STENCIL_BITS_ARB, 8,
        WGL_DOUBLE_BUFFER_ARB, GL_TRUE,
        WGL_SAMPLE_BUFFERS_ARB, GL_TRUE,
        WGL_SAMPLES_ARB, 4,
        0, 0
    };
    float fattributes[] = { 0, 0 };
    UINT numFormats;
    wglChoosePixelFormatARB(hdC, attributes, fattributes, 1, &pixelFormat, &numFormats);
    SetPixelFormat(hDC, pixelFormat, &pfd);
```

Put It All Together

```
const int attribs[] =
{
   WGL_CONTEXT_MAJOR_VERSION_ARB, 4,
   WGL_CONTEXT_MINOR_VERSION_ARB, 3,
   WGL_CONTEXT_FLAGS_PROFILE_MASK_ARB, WGL_CONTEXT_CORE_PROFILE_BIT_ARB,
   WGL_CONTEXT_FLAGS_ARB, WGL_CONTEXT_DEBUG_BIT_ARB,
   0
};

HGLRC hGLRC = wglCreateContextAttribsARB( hDC, 0, attribs );

extern void WINAPI debugCallbackAMD(GLenum id, GLenum category, GLenum severity,
       GLsizei  length, const GLchar* message, GLvoid* userParam);
extern void WINAPI debugCallbackARB(GLenum source, GLenum type, GLenum id,
       GLenum severity, GLsizei length, const GLchar* message, GLvoid* userParam);


if (/* ARB_debug_output is supported */)
{
    glDebugMessageCallbackARB(debugCallbackARB, NULL);
    glEnable(GL_DEBUG_OUTPUT_SYNCHRONOUS_ARB);
}
else if (/*AMD_debug_output is supported */)
{
    glDebugMessageCallbackAMD(debugCallbackAMD, NULL);
}
```

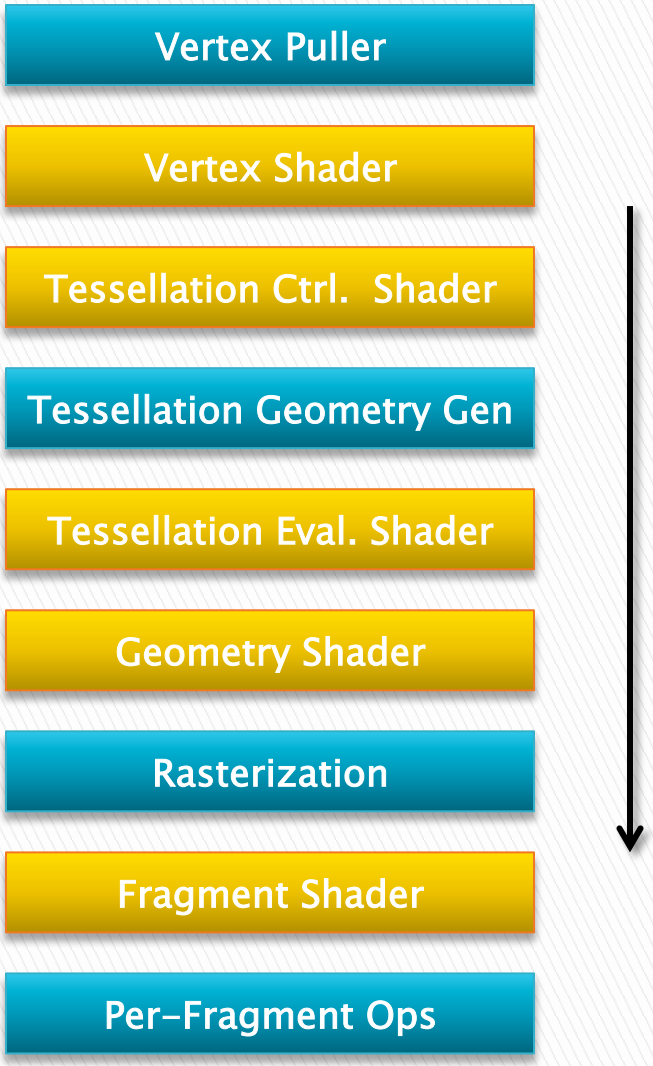
Shaders

- Programming the GPU

Shader Types

- Vertex(2)
- Tessellation Control (4)
- Tessellation Evaluation
- Geometry (3)
- Fragment(2)
- Compute (4.3)

Compute Shader



Data Types

- int, uint, float, double
- vec3
- mat4x3
- struct
- Opaque types
  - sampler
  - image
  - atomic_uint (counter)

An Old GLSL Vertex Shader

```
void main()
{
    // ftransform()
    gl_Position = gl_ProjectionMatrix * gl_ModelViewMatrix *
        gl_Vertex; // glMatrixMode() and glVertex3f().
    gl_TexCoord[0] = gl_TextureMatrix[0] * gl_MultiTexCoord0;
}
```

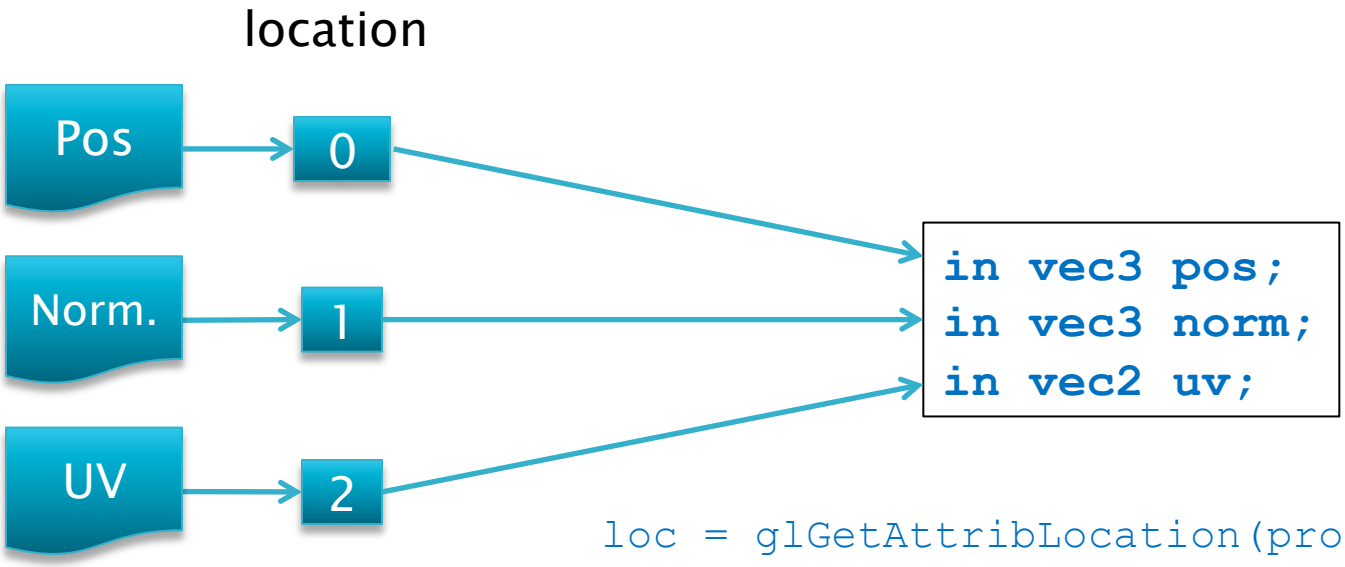
An Old GLSL Fragment Shader

```
uniform sampler2D sDiffuse;

void main(void)
{
    gl_FragColor = texture2D(sDiffuse, gl_TexCoord[0].st);
}
```

A New GLSL Vertex Shader

```
#version 430 core

layout (location=0) attribute vec3 Position; // glVertexAttribPointer()
layout (location=1) attribute vec2 TexCoord;

out gl_PerVertex
{
    vec4 gl_Position;
};

out block
{
    vec2 vTexCoord;
} vs_output;

uniform mat4x4 mVP;
void main(void)
{
    gl_Position = mVP * vec4(Position, 1.0);
    vs_output.vTexCoord = TexCoord;
}
```

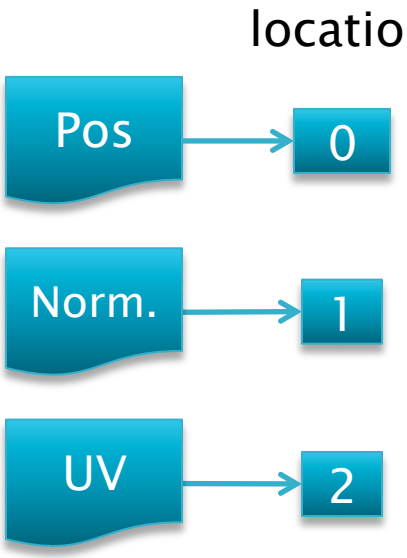
Vertex Attributes




```
loc = glGetAttribLocation(program, name);
glBindAttribLocation(program, loc, name);
```

```
glVertexAttribPointer
    (loc, …, stride, …,
offset);
```

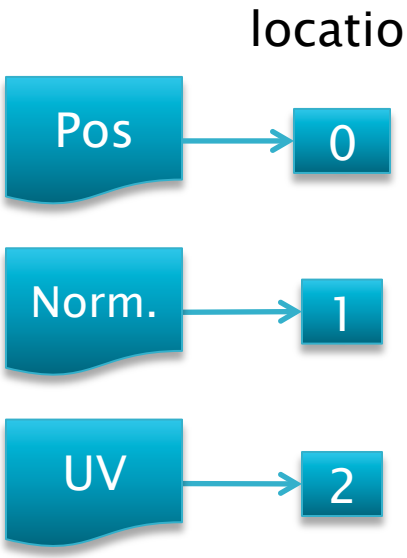
Vertex Attrib (4.2)




```
layout(location = 0) in vec3 pos;
layout(location = 1) in vec3 norm;
layout(location = 2) in vec2 uv;
```

```
glVertexAttribPointer(loc, …, stride, …,
offset);
```

Vertex Attrib 4.3




```
layout(location = 0) in vec3 pos;
layout(location = 1) in vec3 norm;
layout(location = 2) in vec2 uv;
```

```
//Separates format from the array.
glVertexAttribBinding(format, loc);
```

Vertex output

- Need to explicitly declare the vertex built-in output variables.

```
out gl_PerVertex {
 vec4 gl_Position;
 float gl_PointSize;
 float gl_ClipDistance[];
};
```

- Use **`out`** structure to declare “varying” variables for vertex output.

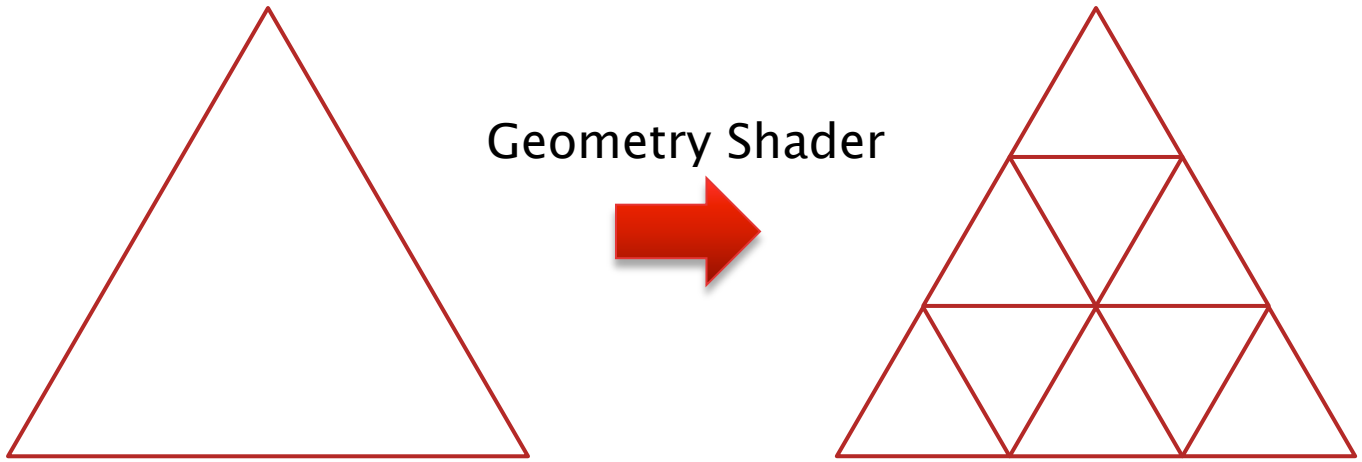
```
out block
{
    vec2 vTexCoord;
} vs_output;
```

A New GLSL Fragment Shader

```
#version 430 core

in fsInput
{
    vec2 vTexCoord;
} fs_input;

uniform sampler2D sDiffuse;

layout (location = 0, index = 0) out vec4 color_out;

void main(void)
{
    color_out = texture2D(sDiffuse, fs_input.vTexCoord);
}
```

Basic Geometry Shader

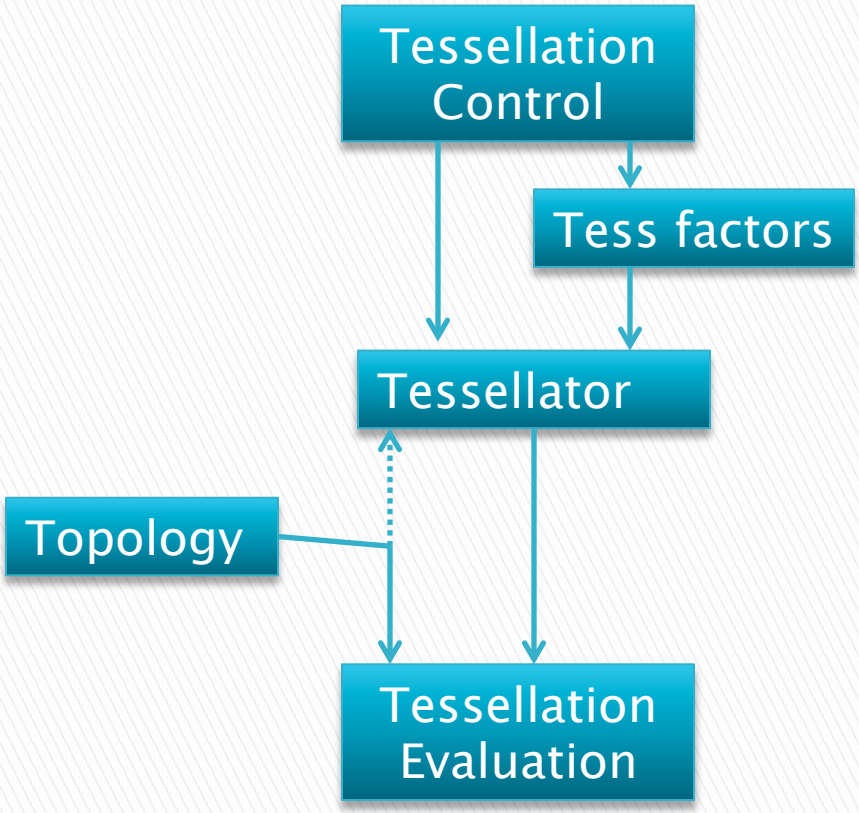
- Introduced in OpenGL 3
- Enhanced vertex shader
  - Can change the topology of primitives
  - Can change the number of primitives



Basic Geometry Shader

```
#version 430 core
layout (lines) in;
layout (triangle_strip, max_vertices = 4) out;
in block
{
    float TexCoord;
} In[];
layout (stream = 0) out block
{
    vec2 UV;
} Out;
void main(void)
{
    gl_Position = gl_in[0].gl_Position - vec4(1.0, 0.0, 0.0, 0.0);
    Out.UV = vec2(0, In[0].TexCoord, Out.UV.x); EmitVertex();
    gl_Position = gl_in[0].gl_Position + vec4(1.0, 0.0, 0.0, 0.0);
    Out.UV = vec2(1, In[0].TexCoord, Out.UV.x); EmitVertex();
    gl_Position = gl_in[1].gl_Position - vec4(1.0, 0.0, 0.0, 0.0);
    Out.UV = vec2(0, In[0].TexCoord, Out.UV.x); EmitVertex();
    gl_Position = gl_in[1].gl_Position + vec4(1.0, 0.0, 0.0, 0.0);
    Out.UV = vec2(1, In[0].TexCoord, Out.UV.x); EmitVertex();

    EndPrimitive();
}
```

Tessellation



Tessellation

```
// Tessellation Control
layout (vertices = 4) out;
void TCS(void)
{
   if (gl_InvocationID == 0)
   {
      gl_TessLevelInner[0] = 2.0;
      …
```

```
// Tessellation Evaluation
layout (quads, cw, equal_spacing) in
void TES(void)
{
   …
```

Tessellation Control

```
out patch float tessFactor;

void main(void)
{
   if (gl_InvocationID == 0)
   {
      gl_TessLevelInner[0] = 2.0;
      …
      tessFactor = 2.0;
   }
   barrier();

   DoSomeWork(tessFactor, gl_InvocationID);
```

Tessellation rate can be set by any instance

Values can be communicated across threads

OpenGL Compute

- Introduced in OpenGL 4.3
- Simplify the GPGPU using OpenGL
  - No more geometry setup
  - No more FBO binding and viewport control
  - No more co-operate with OpenCL/Cuda
- Better performance
  - Shorter pipeline

OpenGL Compute

```
#version 430
layout(r32i) image1D linearOutput;
shared int var;

layout(local_size_x = 64, local_size_y = 1, local_size_z = 1)
void main()
{
    const uvec3 localIdx = gl_LocalInvocationID;
    const uvec3 globalIdx = gl_GlobalInvocationID;
    if(localId.x == 0)
        var = 1;

    barrier();

    imageStore(linearOutput, globalIdx.x, var);
}
```

OpenCL Compute

```
layout(r32i) image1D linearOutput;

__kernel void main(__global uint* linearOutput)
{
    __local int var;

    const uint localIdx = get_local_id(0);
    const uint globalIdx = get_global_id(0);
    if(localIdx == 0)
        var = 1;

    barrier(CLK_GLOBAL_MEM_FENCE);

    linearOutput[globalIdx] = var;
}
```

OpenGL Compute

- Create a OpenGL compute shader
- Dispatch the OpenGL compute shader

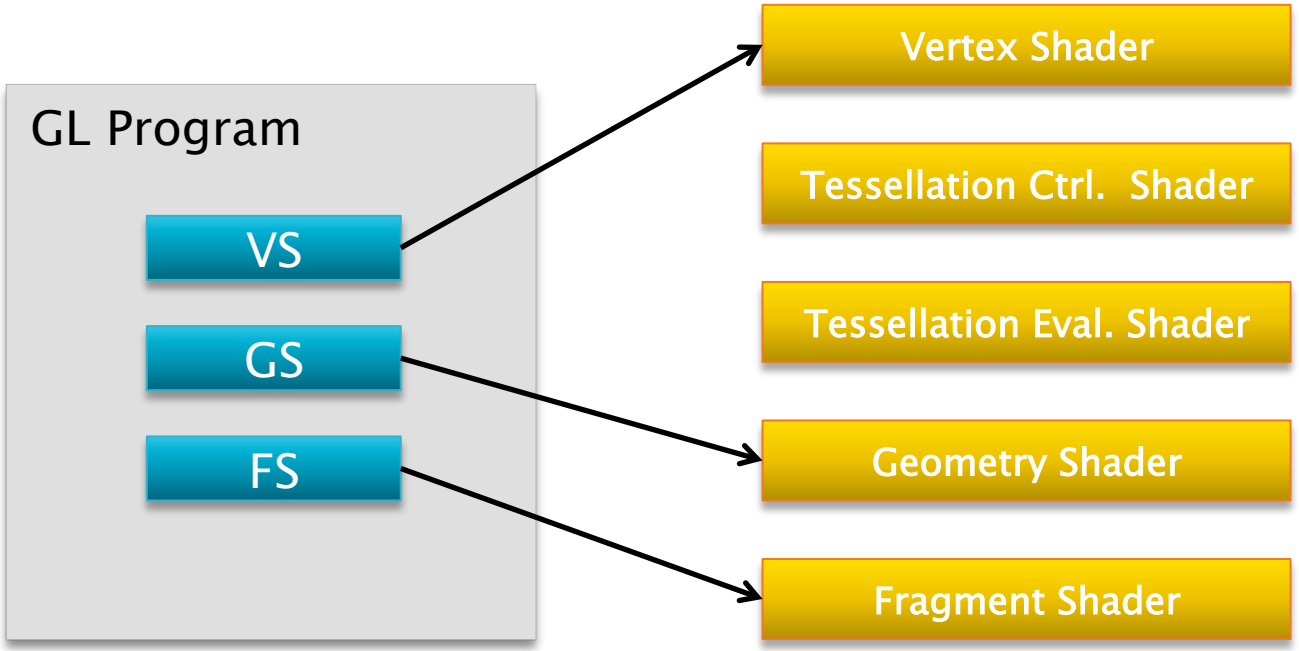
```
// Load and compile compute shader
GLuint prog = glCreateProgram();
GLuint shr = glCreateShader(GL_COMPUTE_SHADER);
glShaderSource(shr, 1, str, NULL);
glCompileShader(shr);
glAttachShader(prog, shr);
glLinkProgram(prog);

// Dispatch a compute shader
glUseProgram(prog);
// There are 256x256 working groups
glDispatchCompute(256, 256, 1);
```

Shader Usage – Old style

```
GLuint shader1 = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(…);
glCompileShader();

GLuint shader2 = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(…);
glCompileShader();

//…

uint program = glCreateProgram();
glAttachShader(program, shader1);
glAttachShader(program, shader2);
glLinkProgram(program);
glUseProgram(program);
```

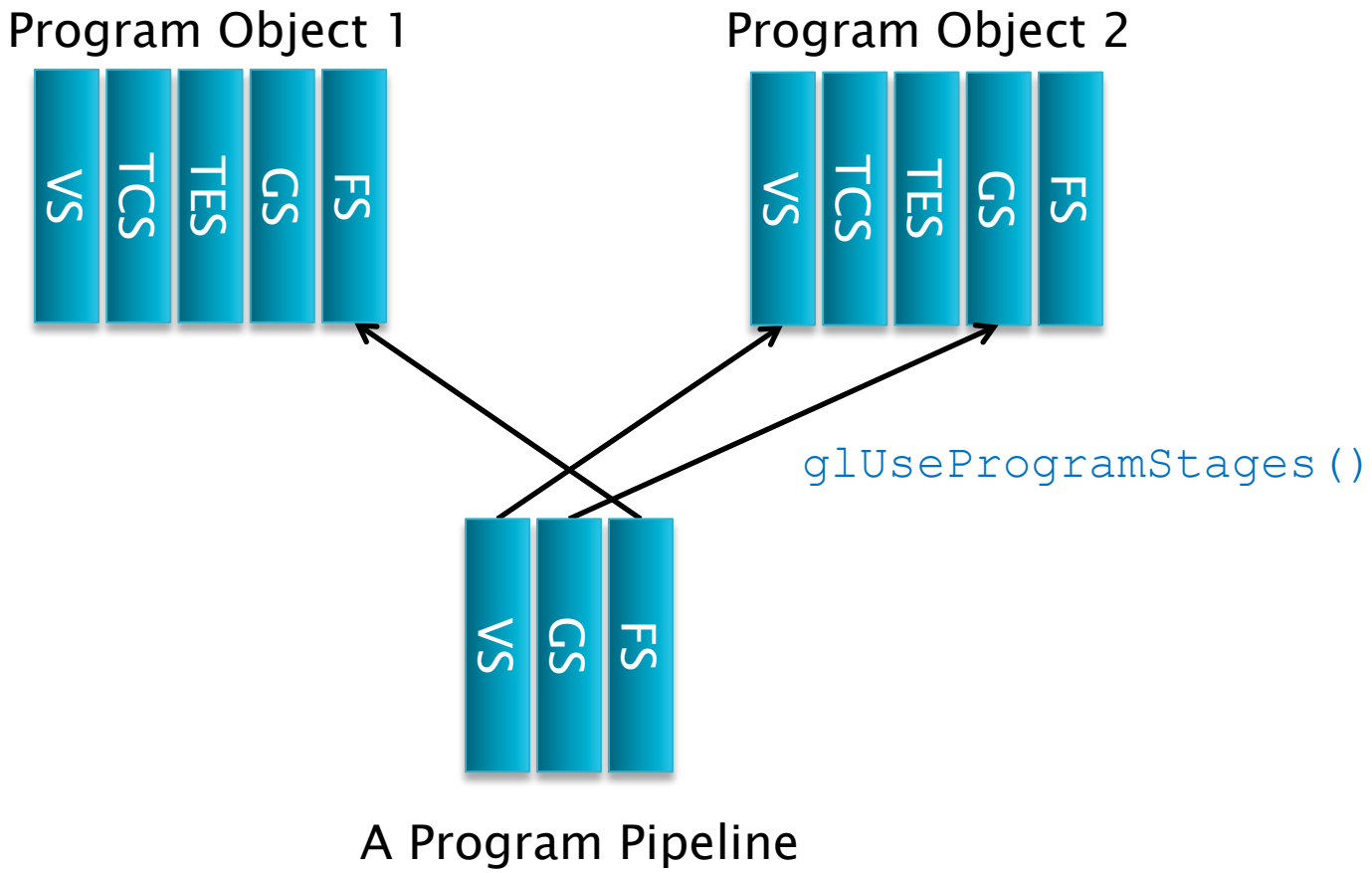
Shader Usage

- Hardware perspective



40

Program Pipeline



Program Pipeline – New Style

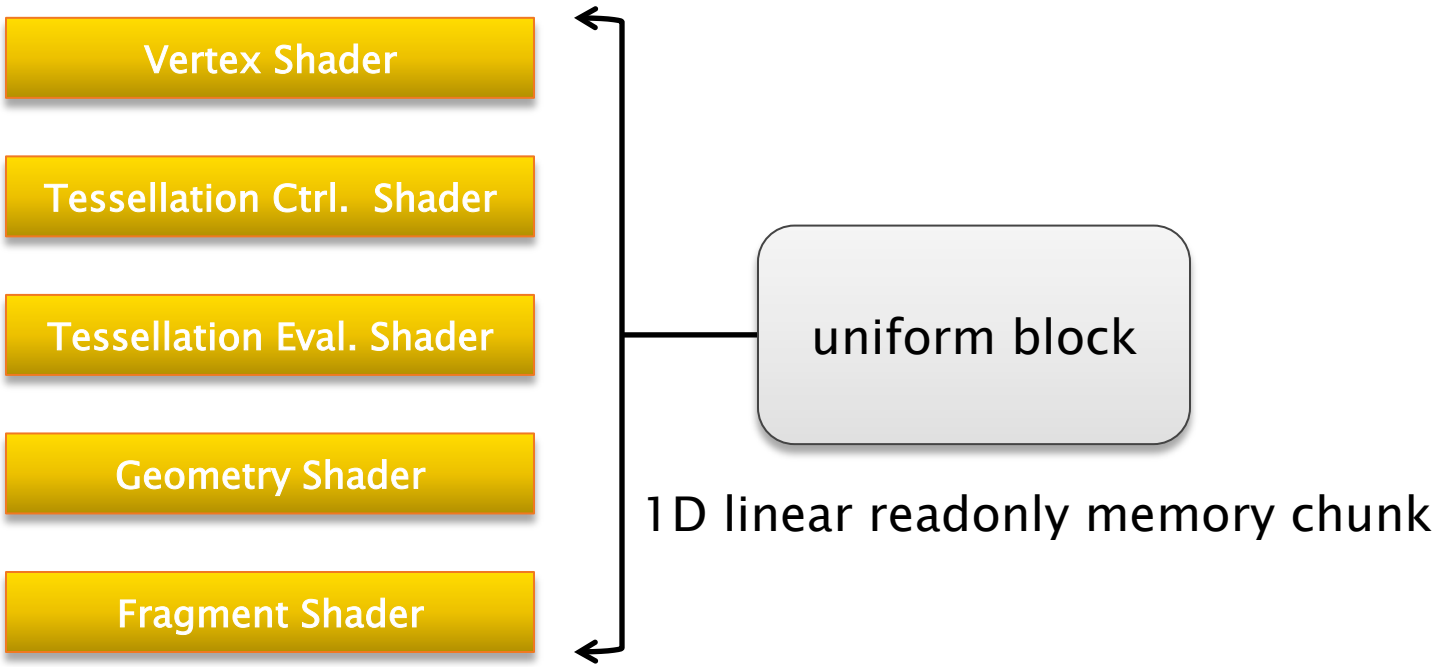
```
GLuint shader1 = glCreateShaderProgramv(GL_VERTEX_SHADER, 1, &vertSrc);
GLuint shader2 = glCreateShaderProgramv(GL_FRAGMENT_SHADER, 1, &fragSrc);

//…


Gluint pipeline;
glGenProgramPipeline(1, &pipeline);
glUseProgramStages(pipeline, GL_VERTEX_SHADER_BIT, shader1);
glUseProgramStages(pipeline, GL_FRAGMENT_SHADER_BIT, shader2);

glBindProgramPipeline(pipeline);
```

Uniform Variables – Old Style

```
// In GLSL
uniform vec4 avariable;


// In application
GLuint loc = glGetUniformLocation(p, “avariable");
glUseProgram(p);
glUniform4f(loc, 0, 0, 0, 0);
```

Uniform Block

- At hardware perspective



Uniform Block

- Core since 3.1
- Share uniforms between different programs
- Quickly change the uniform values

```
layout(std140, column_major) uniform;

uniform transform
{
    mat MVP;
} Transform;

void main()
{
    gl_Position = Transform.MVP * vec4(Position, 1.0);
}
```

Uniform Block

```
// Initialize the buffer and upload buffer data.
struct Buffer
{
    float mat[4][4];
};
GLuint buffer;
glGenBuffers(1, &buffer);
glBindBuffer(GL_UNIFORM_BUFFER, buffer);
glBufferData(buffer, sizeof(Buffer), NULL, GL_DYNAMIC_DRAW);
glBindBuffer(GL_UNIFORM_BUFFER, 0);

// Render
GLuint uniformBlockIndex = glGetUniformBlockIndex(program, “transform”);
glUniformBlockBinding(program, uniformBlockIndex, 0);
glBindBufferRange(GL_UNIFORM_BUFFER, 0, buffer, 0, sizeof(Buffer));
```

Shader Storage

- Shader Storage only available with 4.3
- Shader Storage for more general read/write and atomics (but not counters)
- Last element can be unsized array (like DX structured buffers)
- Generally slower than uniforms but uniforms are read-only

Shader Storage Block (4.3)

```
layout (binding = 0, std140) buffer ColorTable
{
    vec3 colors[3]; //this is actually writeable
};

GLuint buffer;
glGenBuffers(1, & buffer);
glBindBuffer(GL_SHADER_STORAGE_BUFFER, buffer);
glBufferData(GL_SHADER_STORAGE_BUFFER, sizeof(colorTable),
             colorTable, GL_DYNAMIC_DRAW);
glBindBufferRange(GL_SHADER_STORAGE_BUFFER, 0, buffer,
             0, sizeof(colorTable));
```

Shader Storage Block

```
layout (binding = 0, std140) buffer ColorTable
{
    volatile vec3 colors[3];
    coherent uint ready;
    MyStruct data[]; // Last item can be unsized
};

colors[0] = vec3(1, 1, 1);
colors[1] = vec3(1, 1, 1);
memoryBarrier();

// over in some other threads
if(ready == 1)
{
    colors[1] = vec3(1, 0, 0);
}
// colors[0] will be white, while colors[1] is red in some
threads
```

Program Binaries

- ARB_get_program_binary (since core 4.1)

```
uint program = glCreateProgram();

if (/* Load program binary from the disk successfully */)
{
    glProgramBinary(program, binaryFormat, binaryData, dataLength);
}
else
{
    glCreateShader();
    glShaderSource(…);
    glCompileShader();

    glAttachShader(program, shader);
    glLinkProgram(program);
    glGetProgramBinary(program,…,format,pBinaryOut);

    /* Dump binary file to the disk. */
}
```

Textures and Buffers

- New Interface of Textures and Buffers

Buffer Targets

GL Name	Typical Purpose	DX Equivalent
ARRAY	Vertices	VERTEX
ELEMENT_ARRAY	Indices	INDEX
UNIFORM	Read-only vars	CONSTANT
TEXTURE_BUFFER	Buffer-as-texture	CONSTANT (tbuffer)
SHADER_STORAGE	Read/write	SHADER_RESOURCE
TRANSFORM_FEEDBACK		Stream out
DRAW_INDIRECT	indirect draw	DRAWINDIRECT
ATOMIC_COUNTER	Global counter var	UAV_FLAG_COUNTER
COPY_READ, _WRITE	Copying (optional)	Staging?
PIXEL_PACK, _UNPACK	GPU <-> CPU	Staging?

Texture Sampling – old style

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

- Better to decouple sampling parameters from the texture

Sampler

- Reuse the sampling parameters
- Change the sampling of a group of textures at one time

```
glGenSamplers(1, &sampler);

glSamplerParameteri(sampler, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glSamplerParameteri(sampler, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glSamplerParameteri(sampler, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glSamplerParameteri(sampler, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture1);
glBindSampler(0, sampler);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texutre2);
glBindSampler(1, sampler);
```

Sampler

- No counterpart as in DirectX that can declare sampler inside GLSL

```
SamplerState sampler
{
    Filter   = MIN_MAG_MIP_LINEAR;
    AddressU = Clamp;
    AdddresV = Clamp;
};
```

Texture Binding in Shader

- Old style

```
GLint loc = glGetUniformLocation(program, "texture");
glUseProgram(program);
glUniform1i(loc, 0);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture1);
```

- New style

```
// texture is bound to texture unit 0
layout (binding = 0) uniform sampler2D texture;
```

Bindless Texture

- GL_ARB_bindless_texture
  - No need to bind texture to texture unit
  - Unlimited textures accessible in shader

```
Gluint tex;
glGenTextures(1, &tex);
// Fill the texture data and set the texture parameters
// …
GLuint64 texHandle = glGetTextureHandleARB(tex);
glMakeTextureResidentARB(texHandle);
glProgramUniformHandleui64ARB(loc, texHandle);
```

Images

- Texture as an indexable array of data
- It's a shader bind, not an object.
- Not sampled
- Reads and **scattered** writes (like UAV in DX)
- Atomic operations

```
imageLoad(nDIntAddress);
imageStore(nDIntAddress,value);
```

Images

gimage1D	GL_TEXTURE_1D
gimage2D	GL_TEXTURE_2D
gimage3D	GL_TEXTURE_3D
gimageCube	GL_TEXTURE_CUBE_MAP
gimage2DRect	GL_TEXTURE_RECTANGLE
gimage1DArray	GL_TEXTURE_1D_ARRAY
gimage2DArray	GL_TEXTURE_2D_ARRAY
gimageCubeArray	GL_TEXTURE_CUBE_MAP_ARRAY
gimageBuffer	GL_TEXTURE_BUFFER
gimage2DMS	GL_TEXTURE_2D_MULTISAMPLE
gimage2DMSArray	GL_TEXTURE_2D_MULTISAMPLE_ARRAY

Images

- A few examples

```
layout (binding = 0, rgba8) coherent uniform image2D Diffuse;
layout (binding = 1, rgba32ui) coherent uniform uimage2D Bitmap;
layout (binding = 2, rgba32f) coherent uniform image1D ColorMap;
```

- Image load and store

```
in fsInput
{
    vec2 UV;
} fs_input;

void main()
{
    vec4 c = imageLoad(Diffuse, ivec2(fs_input.UV * imageSize(Diffuse)));
    imageStore(Bitmap, ivec2(fs_input.UV * imageSize(Bitmap)), c);
}
```

Images

- Usage

```
glGenTextures(1, &image);
glBindTexture(GL_TEXTURE_2D, image);
glTexStorage2D(GL_TEXTURE_2D, 1, GL_R32UI, 128, 128);
glTexSubImage2D(…);

glBindImageTexture(0, image, 0, GL_FALSE, 0, GL_READ_WRITE, GL_R32UI);
```

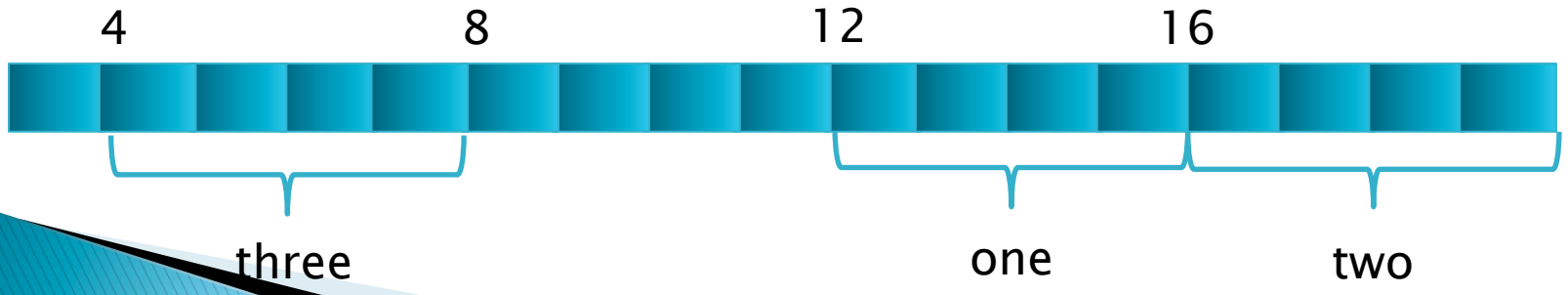
Images – Atomic Instructions

uint imageAtomicAdd(), int imageAtomicAdd()
uint imageAtomicMin(), int imageAtomicMin()
uint imageAtomicMax(), int imageAtomicMax()
uint imageAtomicAnd(), int imageAtomicAnd()
uint imageAtomicOr(), int imageAtomicOr()
uint imageAtomicXor(), int imageAtomicXor()
uint imageAtomicExchange(), int imageAtomicExchange()
uint imageAtomicCompSwap(), int imageAtomicCompSwap()

Atomic counter

- A GLSL variable that uses a buffer object's memory
  - unsigned integer only
  - Increase and decrement by 1 only
  - Coherent, no need to call barrier to synchronize counter access

```
layout(binding = 0, offset = 12) uniform atomic_uint one;
layout(binding = 0) uniform atomic_uint two;
layout(binding = 0, offset = 4) uniform atomic_uint three;
```



Atomic counter

```
glBindBuffer(GL_ATOMIC_COUNTER_BUFFER, _buffer);
glBufferData(GL_ATOMIC_COUNTER_BUFFER, sizeof(GLuint), NULL, GL_DYNAMIC_COPY);
glBindBuffer(GL_ATOMIC_COUNTER_BUFFER, 0);

GLuint* ptr = (GLuint*)glMapBufferRange(
        GL_ATOMIC_COUNTER_BUFFER,
        0,
        sizeof(GLuint),
        GL_MAP_WRITE_BIT | GL_MAP_INVALIDATE_BUFFER_BIT);
*ptr = 0;
glUnmapBuffer(GL_ATOMIC_COUNTER_BUFFER);

glBindBufferBase(GL_ATOMIC_COUNTER_BUFFER, location, _buffer);
```

```
layout(binding = 0) uniform atomic_uint one;

void main()
{
    // Increase the counter and return old value
    uint c = atomicCounterIncrement(one);
}
```

OpenGL Texture Views (4.3)

- Initialize a texture as a data alias of another texture's data store
- Two textures share the same memory address but with different data type expression

```
void glTextureView(GLuint texture,
        GLenum target,
        GLuint origtexture,
        GLenum internalformat,
        GLuint minlevel,
        GLuint numlevels,
        GLuint minlayer,
        GLuint numlayers);
```

Buffer Texture

- Texture interface with a buffer behind it
- 1D
- No filter support (always GL_NEAREST)
- `texelFetch(sampler,oneDIntAddress);`
- Pretty much infinite in size (currently 1GB or so in practice)
- Example use: accessing vertex data

Buffer Texture

- Example: read the vertex data as a texture

```
GLuint vertexBuffer;
glGenBuffers(1, &vertexBuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
glBufferData(GL_ARRAY_BUFFER, vertexBufferSize,
    vertexBufferData, GL_STATIC_READ);
glBindBuffuer(GL_ARRAY_BUFFER, 0);

GLuint tex;
glGenTextures(1, &tex);
glBindTexture(GL_TEXTURE_BUFFER, tex);
// Create the texture storage with an existing buffer.
glTexBuffer(GL_TEXTURE_BUFFER, GL_RGBA32F, vertexBuffer);
```

Last But Not Least

- Draw calls, Transform Feedback and Synchronization

Draw Calls

- glDrawArrays
- glDrawArraysInstanced
- glDrawArraysInstancedBaseInstance
- glDrawArraysIndirect
- glMultiDrawArrays
- glMultiDrawArraysIndirect
- glDrawElements
- …and so forth

glDrawArraysIndirect

- Render primitives from array data, taking parameters from memory

```
void glDrawArraysIndirect(GLenum mode, const void *indirect);
```

- The data can be from GPU memory, i.e., GL_DRAW_INDIRECT_BUFFER

glDrawArraysIndirect

```
typedef struct
{
    uint count;
    uint instanceCount;
    uint first;
    uint baseInstance;
} DrawArraysIndirectCommand;

DrawArraysIndirectCommand command = {…};

glGenBuffers(1, &buffer);
glBindBuffer(GL_DRAW_INDIRECT_BUFFER, buffer);
glBufferData(GL_DRAW_INDIRECT_BUFFER, sizeof(Command), command)

glDrawArraysIndirect(GL_TRIANGLES, 0);
```

glDrawArraysIndirect

- Pure GPU implementation
  - Bind buffer to GL_SHADER_STORAGE_BUFFER and update the buffer in shader.

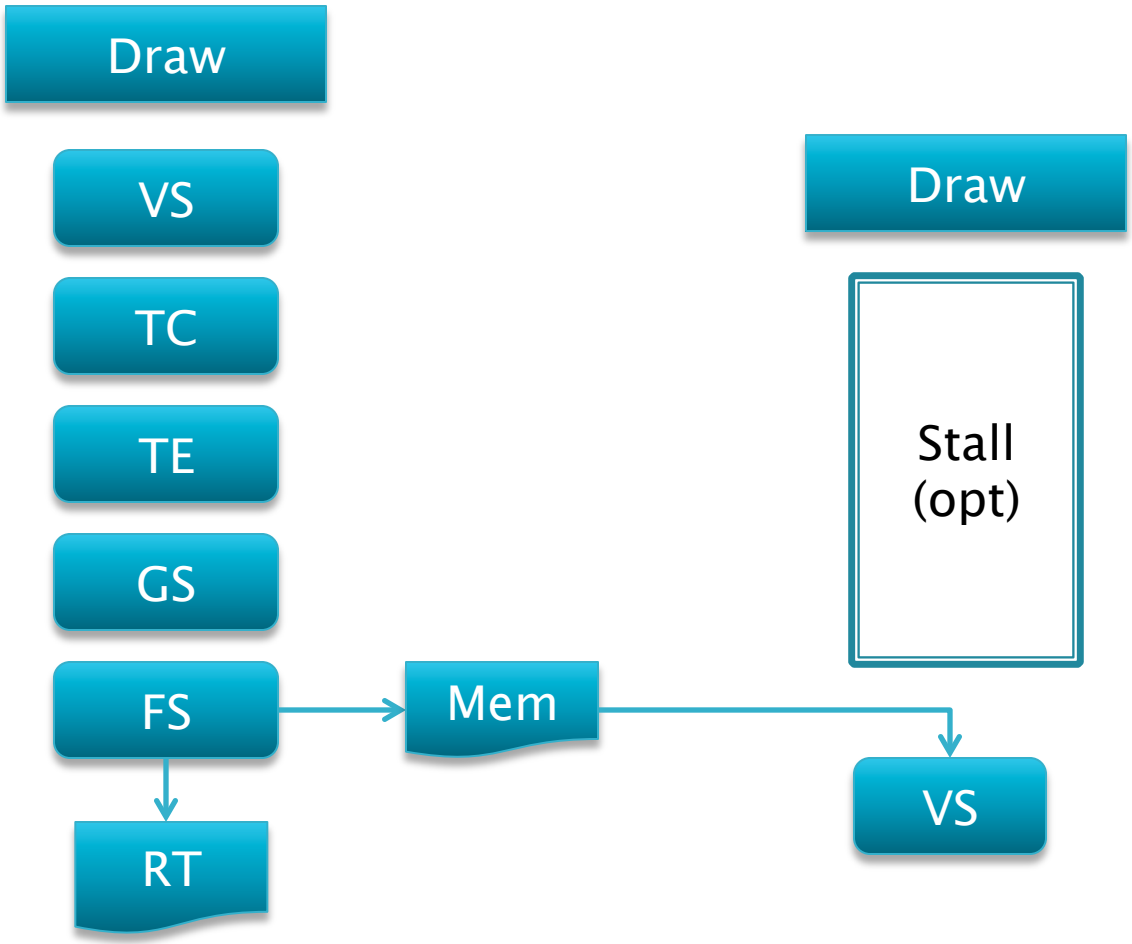
```
typedef struct
{
    uint count;
    uint instanceCount;
    uint first;
    uint baseInstance;
} DrawCommand;

GLuint cmdBuf = 0;
glGenBuffers(1, &cmdBuf);
glBindBuffer(GL_SHADER_STORAGE_BUFFER, cmdBuf);
glBufferData(GL_SHADER_STORAGE_BUFFER,
    sizeof(DrawCommand), 0, GL_DYNAMIC_COPY);
glBindBuffer(GL_SHADER_STORAGE_BUFFER, 0);
```

glDrawArraysIndirect

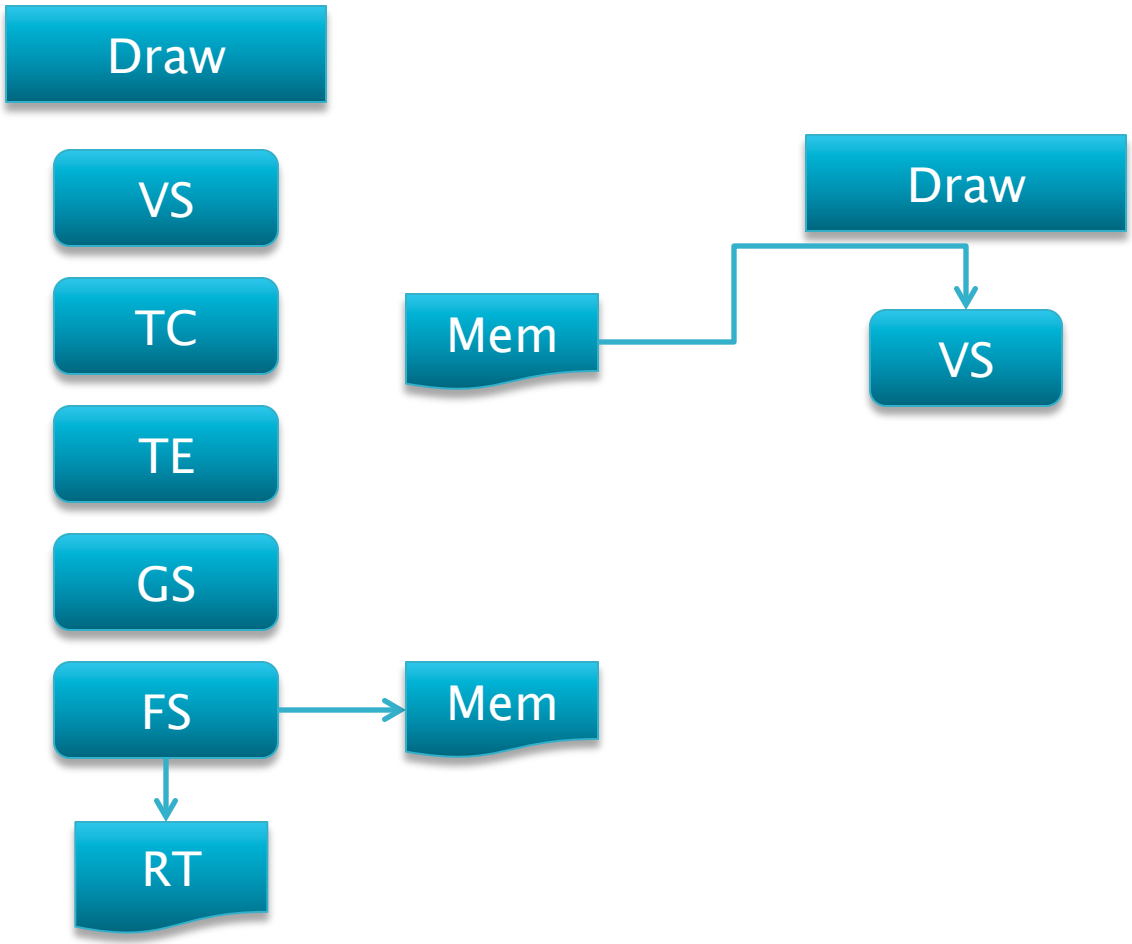
```
GLunit blockId = 0;
blockId = glGetProgramResourceIndex(program,
    GL_SHADER_STORAGE_BLOCK, “Cmd");
glShaderStorageBlockBinding(program, blockId,
    2);


layout (std430, binding=2) buffer Cmd
{
    uint count;
    uint instanceCount;
    uint first;
    uint baseInstance;
}

void main()
{
    count = 0;
    // …
}
```

OpenGL Command Queue

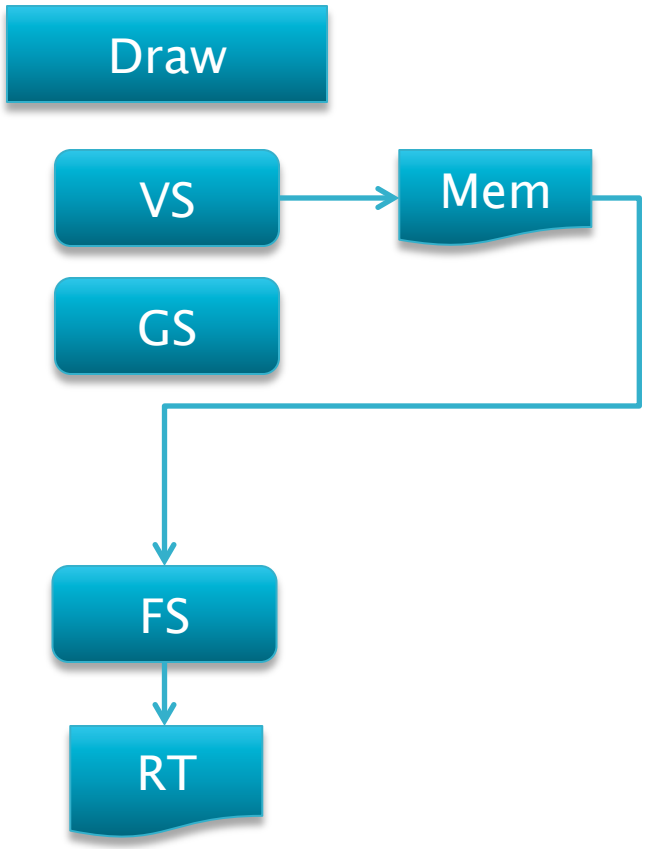
Memory Coherence



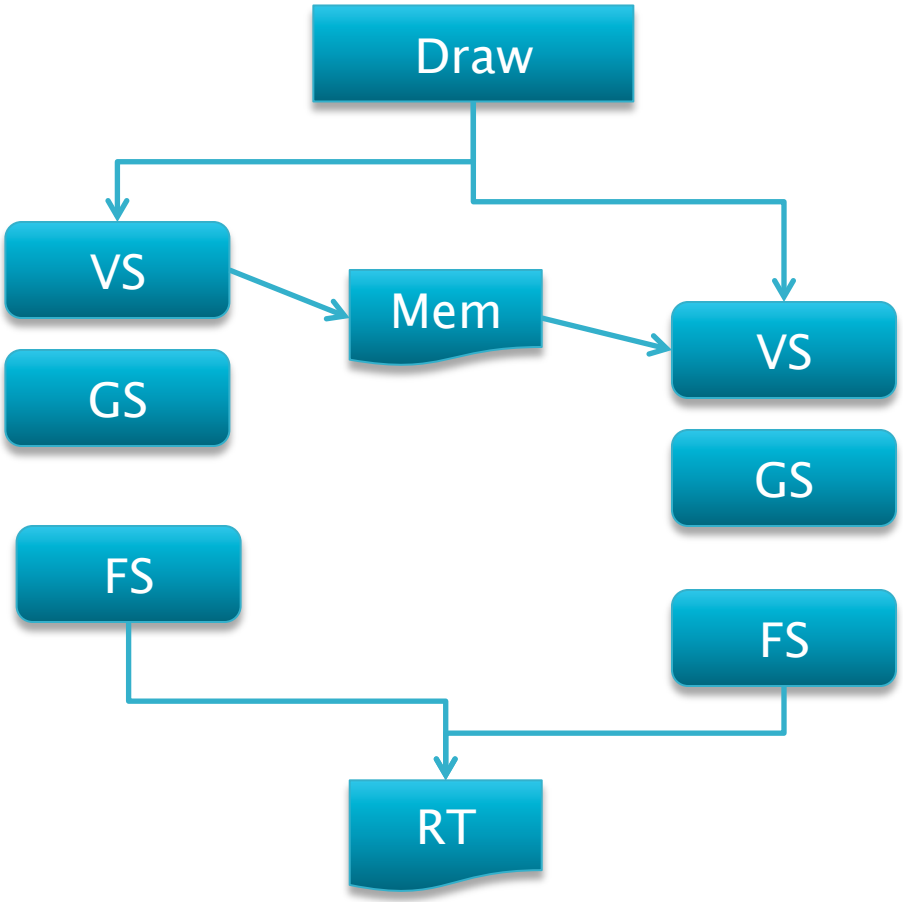
Memory Coherence



Memory Coherence



Memory Coherence



OpenGL Compute

- Flush the cache written by image-load-store

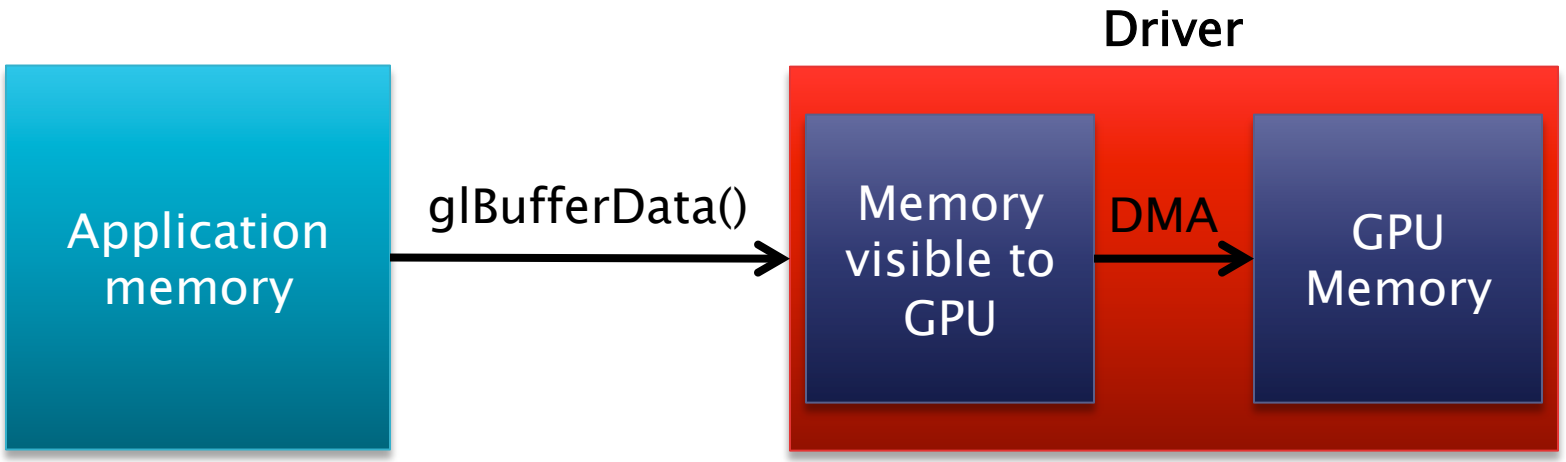
```
void glMemoryBarrier(GLbitfield barriers);
```

- Caches

```
GL_VERTEX_ATTRIB_ARRAY_BARRIER_BIT
GL_ELEMENT_ARRAY_BARRIER_BIT
GL_UNIFORM_BARRIER_BIT
GL_TEXTURE_FETCH_BARRIER_BIT
GL_SHADER_IMAGE_ACCESS_BARRIER_BIT
GL_COMMAND_BARRIER_BIT
GL_PIXEL_BUFFER_BARRIER_BIT
GL_TEXTURE_UPDATE_BARRIER_BIT
GL_BUFFER_UPDATE_BARRIER_BIT
GL_QUERY_BUFFER_BARRIER_BIT
GL_CLIENT_MAPPED_BUFFER_BARRIER_BIT
GL_FRAMEBUFFER_BARRIER_BIT
GL_TRANSFORM_FEEDBACK_BARRIER_BIT
GL_ATOMIC_COUNTER_BARRIER_BIT
GL_SHADER_STORAGE_BARRIER_BIT
```

Data Transfer: Synchronous Copy

```
void glBufferData(GLenum target, Glsizeiptr size, const Glvoid*
data, GLenum usage);
void glBufferSubData(GLenum target, Glintptr offset, Glsizeptr
size, const GLvoid* data);
```



Data Transfer: Asynchronous Copy

- Asynchronous copy from CPU to GPU using GL_MAP_UNSYNCHORNIZED_BIT in glMapBuffer

```
// At each frame
const int buffer_number = frame_number++ % 3;
GLenum result = glClientWaitSync(fences [buffer_number], 0, TIMEOUT );
glDeleteSync(fences[buffer_number]);
glBindBuffer(GL_ARRAY_BUFFER , buffers[buffer_number]);
void *ptr = glMapBufferRange(GL_ARRAY_BUFFER, offset, size,
GL_MAP_WRITE_BIT | GL_MAP_UNSYNCHRONIZED_BIT);
// Fill ptr with useful data
glUnmapBuffer(GL_ARRAY_BUFFER);
// Use buffer in draw operation
// Put fence into command queue
fences[buffer_number] = glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE, 0);
```

courtesy of Ladislav Hrabcak

GL_pinned_memory_AMD

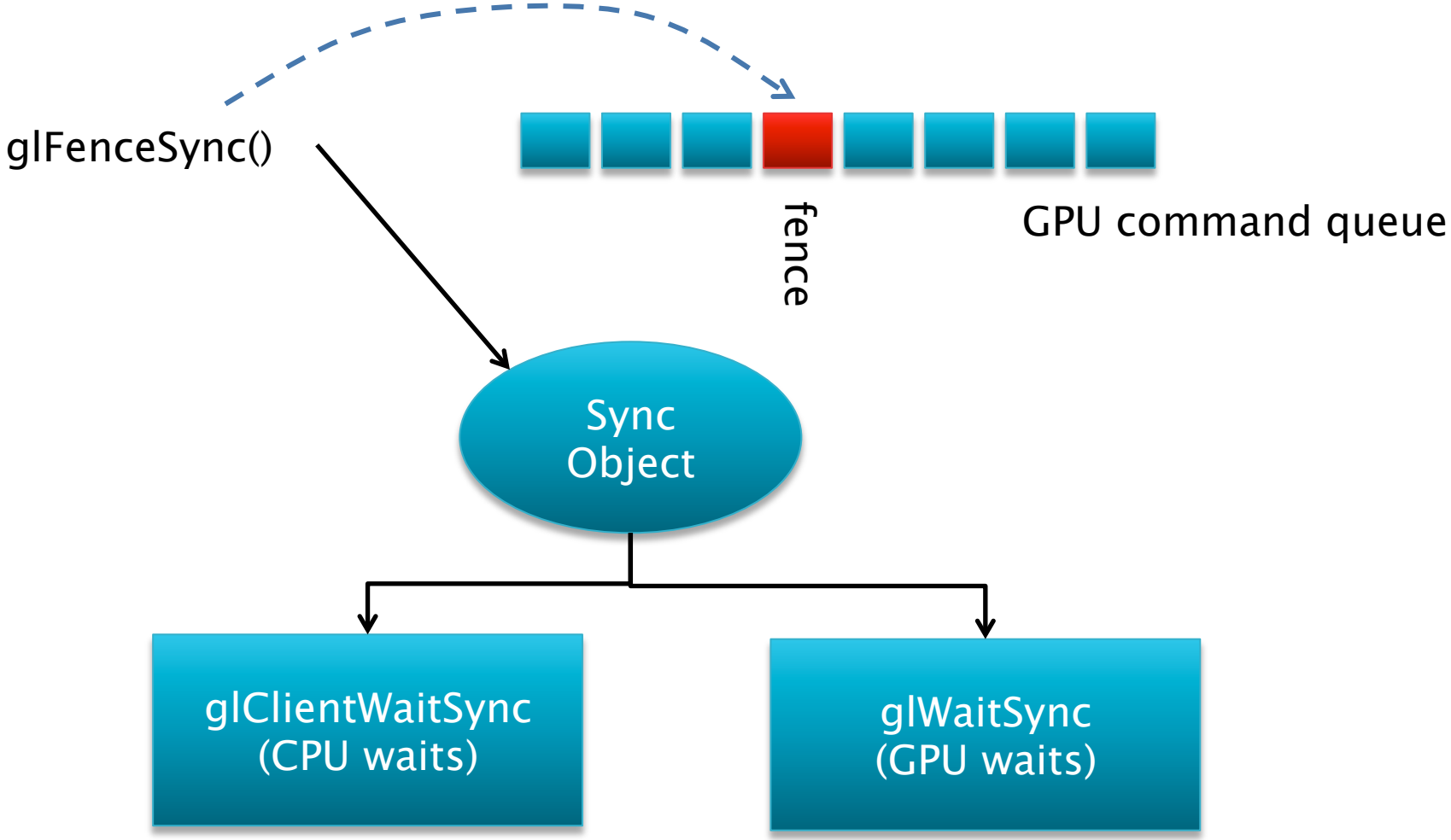
- Can use user provided memory to be the buffer storage.
  - No GL call overhead
  - No memory management overhead in driver
  - No data synchronization control from driver
- Since Catalyst 11.5

```
char*_pinned_ptr = new char[buffer_size + 0x1000 ];
char*_pinned_ptr_aligned = reinterpret_cast <char *>(unsigned
(_pinned_ptr + 0xfff) & (~0xfff));
// Copy data to this aligned user memory.
glBindBuffer(GL_EXTERNAL_VIRTUAL_MEMORY_AMD, buffer);
glBufferData(GL_EXTERNAL_VIRTUAL_MEMORY_AMD, buffer_size,
    _pinned_ptr_aligned, GL_STREAM_READ);
glBindBuffer(GL_EXTERNAL_VIRTUAL_MEMORY_AMD, 0);
```

Command Synchronization

- Old style
  - `glFinish, glFlush`
- Sync object and fence (since 4.1)
  - `glFenceSync` – insert a sync command to the command queue
  - `glClientWaitSync` – block the CPU clock till the specific sync command has been executed
  - `glWaitSync` – block the GPU till the specific sync command has been executed

Command Synchronization



References

- OpenGL.org
  - Language Specs (OpenGL and GLSL)
  - Extension registry
  - Links to tutorials, etc.
  - Wiki
- OpenGL Programming Guide, etc.
- OpenGL Insights, Christophe Riccio

Thanks

- Advanced GPU Technology Initiatives Team
  - Jason Yang
  - Karl Hillesland
- Graham Sellers
- Reviewers