Diligent Engine: Building a Modern Graphics Abstraction Layer

Egor Yusov, Diligent Graphics

Presented at the Khronos Vulkanised 2023 Conference
Agenda

- Motivation
- Diligent API Quick Overview
- **Resource Binding System**
- Pipeline State Packaging
- Testing
The Graphics API Diagram

Convenience (Hello Δ LOC)

Flexibility

1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0

OpenGL
Microsoft® DirectX
Microsoft® DirectX
Vulkan

Vulkanised 2023 | February 7-9, Munich, Germany
Graphics API vs GPU API

```cpp
layout(set=X, binding=Y) uniform sampler2D Texture;
```

Vulkan:

- Assign non-conflicting X and Y in the shader
- Create a descriptor set layout that uses the exact same binding
- Create pipeline descriptor that uses the set layout
- Create pipeline with the layout
- Create descriptor set from the set layout
- Write descriptor in the set at binding Y
- Bind the set at index X
Graphics API vs GPU API

```
layout(set=X, binding=Y) uniform sampler2D Texture;

Diligent:

pPSO->CreateShaderResourceBinding(&pSRB);

pSRB->GetVariableByName(SHADER_TYPE_PIXEL, "Texture")->Set(TextureSRV);

pCtx->CommitShaderResources(pSRB, RESOURCE_STATE_TRANSITION_MODE_TRANSITION);
```

- Encompasses descriptor sets and other binding details
- Accesses shader resource by name
- Binds all sets and executes barriers if necessary
Diligent API – Key Concepts

● Convenience
  ○ Resources are accessed by names, no explicit set/bindings
  ○ Default implementation for common operations

● Modern features
  ○ Multithreading, bindless, command queues, async compute, ray tracing, mesh shaders, VRS, sparse resources, ...

● Flexibility
  ○ Application can take full control where needed
Diligent API Example – PSO Initialization

GraphicsPipelineStateCreateInfo PSOCreateInfo;

auto& PSODesc    = PSOCreateInfo.PSODesc;
auto& GraphicsPipeline = PSOCreateInfo.GraphicsPipeline;

PSODesc.Name        = "My PSO";
PSODesc.PipelineType = PIPELINE_TYPE_GRAPHICS;

GraphicsPipeline.NumRenderTargets                    = 1;
GraphicsPipeline.RTVFormats[0]                       = TEX_FORMAT_RGBA8_UNORM_SRGB;
GraphicsPipeline.DSVFormat                            = TEX_FORMAT_D32_FLOAT;
GraphicsPipeline.PrimitiveTopology                    = PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
GraphicsPipeline.RasterizerDesc.CullMode             = CULL_MODE_BACK;
GraphicsPipeline.DepthStencilDesc.DepthEnable        = true;
Diligent API Example – Creating Shaders

ShaderCreateInfo ShaderCI;

ShaderCI.pShaderSourceStreamFactory = pShaderSourceFactory;

ShaderCI.SourceLanguage = SHADER_SOURCE_LANGUAGE_HLSL;
ShaderCI.Desc.ShaderType = SHADER_TYPE_VERTEX;
ShaderCI.EntryPoint = "main";
ShaderCI.Desc.Name = "Cube VS";
ShaderCI.FilePath = "cube.vsh";
pDevice->CreateShader(ShaderCI, &pVS);

PSOCreateInfo.pVS = pVS;
PSOCreateInfo.pPS = pPS;
Diligent API Example – Input Layout

```c
LayoutElement LayoutElems[] =
{
    // Attribute 0 - vertex position
    LayoutElement{0, 0, 3, VT_FLOAT32},
    // Attribute 1 - texture coordinates
    LayoutElement{1, 0, 2, VT_FLOAT32}
};

PSOCreateInfo.GraphicsPipeline.InputLayout.LayoutElements = LayoutElems;
PSOCreateInfo.GraphicsPipeline.InputLayout.NumElements = 2;
```

```c
struct VSInput
{
    float3 Pos : ATTRIB0;
    float2 UV : ATTRIB1;
};

VSOutput main(in VSInput VSIn)
```
Diligent API Example – Resource Layout

```cpp
ShaderResourceVariableDesc Vars[] =
{
    {SHADER_TYPE_PIXEL, "Texture", SHADER_RESOURCE_VARIABLE_TYPE_MUTABLE}
};
PSOCreateInfo.PSODesc.ResourceLayout.Variables = Vars;
PSOCreateInfo.PSODesc.ResourceLayout.NumVariables = 1;

pDevice->CreateGraphicsPipelineState(PSOCreateInfo, &pPSO);
```
Diligent API Example – Shader Resource Binding

Static resources are set in the PSO

```cpp
pPSO->GetStaticVariableByName(SHADER_TYPE_VERTEX, "Constants")->Set(pVSConstants);
pPSO->CreateShaderResourceBinding(&pSRB, true);

TextureLoadInfo loadInfo;
loadInfo.IsSRGB = true;
RefPtr<ITexture> pTex;
CreateTextureFromFile("Texture.png", loadInfo, m_pDevice, &pTex);
auto* pTextureSRV = pTex->GetDefaultView(TEXTURE_VIEW_SHADER_RESOURCE);

pSRB->GetVariableByName(SHADER_TYPE_PIXEL, "Texture")->Set(pTextureSRV);
```

Shade Resource Binding object is created from the Pipeline

Mutable and dynamic resources are set in the SRB
Diligent API Example – Rendering

auto* pRTV = pSwapChain->GetCurrentBackBufferRTV();
auto* pDSV = pSwapChain->GetDepthBufferDSV();
pCtx->SetRenderTargets(1, &pRTV, pDSV, RESOURCE_STATE_TRANSITION_MODE_TRANSITION);

const float ClearColor[] = {0.35f, 0.35f, 0.35f, 1.0f};
pCtx->ClearRenderTarget(pRTV, ClearColor, RESOURCE_STATE_TRANSITION_MODE_TRANSITION);

pCtx->ClearDepthStencil(pDSV, CLEAR_DEPTH_FLAG, 1.f, 0, RESOURCE_STATE_TRANSITION_MODE_TRANSITION);

Tells the engine how to handle resource state transitions

Possible options are:
- None
- Transition
- Verify
Diligent API Example – Rendering

```c
const Uint64 offset = 0;
IBuffer* pBuffs[] = {pVertexBuffer};
pCtx->SetVertexBuffers(0, 1, pBuffs, &offset, RESOURCE_STATE_TRANSITION_MODE_TRANSITION);
pCtx->SetIndexBuffer(pIndexBuffer, 0, RESOURCE_STATE_TRANSITION_MODE_TRANSITION);

pCtx->SetPipelineState(pPSO);
pCtx->CommitShaderResources(pSRB, RESOURCE_STATE_TRANSITION_MODE_TRANSITION);

DrawIndexedAttribs DrawAttrs;
DrawAttrs.IndexType = VT_UINT32;
DrawAttrs.NumIndices = 36;
pCtx->DrawIndexed(DrawAttrs);
```

Binds descriptor sets
Shader Resource Binding Model 1.0

- Use reflection to get the list of resources
- Each resource is classified by the app as
  - Static - can only be set once in the PSO
  - Mutable - can only be set once in each SRB instance
  - Dynamic - can be set multiple times in each SRB instance
- Programmatically assign descriptor sets and bindings
- Patch SPIRV to make it consistent with bindings
Shader Resource Binding Model 1.0

- Two descriptor sets (if necessary)
  - Static/Mutable resources
  - Dynamic resources

- Within each set, 3 groups (total 6 groups)
  - Dynamic-offset uniform buffers
  - Dynamic-offset storage buffers
  - All other resources
for (const auto& ResDesc : PipelineResources) {
    const auto SetId = VarTypeToDescriptorSetId(ResDesc.VarType); // 0 - Static/Mutable, 1 - Dynamic
    const auto DescrType = GetDescriptorType(ResDesc);
    const auto CacheGroup = GetResourceCacheGroup(ResDesc); // [Static/Mutable, Dynamic] x [Dyn UB, Dyn SB, Other]

    AddDescriptor(Binding[CacheGroup], ResDesc.ArraySize, DescrType, SetId, CacheOffsets[CacheGroup]);

    Binding[CacheGroup] += 1;
}
uniform GlobalAttribsUB {}; // Static, no dyn offsets
uniform FrameTransformsUB {}; // Mutable, no dyn offsets
uniform MeshTransformsUB {}; // Mutable, dyn offsets
uniform SpecialAttribsUB {}; // Dynamic, dyn offsets

buffer LightsSB {}; // Static, dyn offsets

uniform sampler2D ShadowMap; // Static
uniform sampler2D BaseColor[8]; // Mutable
uniform sampler2D Normals[8]; // Mutable
uniform sampler2D ColorCorection; // Dynamic

### Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Binding</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeshTransformsUB</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LightsSB</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>GlobalAttribsUB</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>FrameTransformsUB</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>ShadowMap</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>BaseColor</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Normals</td>
<td>6</td>
<td>13</td>
</tr>
</tbody>
</table>

### Set 0 - Static / Mutable

<table>
<thead>
<tr>
<th>Name</th>
<th>Binding</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpecialAttribsUB</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ColorCorection</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
// Load SPIRV resources
spirv_cross::Parser parser{move(spirv_binary)};
parser.parse();

const auto ParsedIRSource = parser.get_parsed_ir().source;
spirv_cross::Compiler Compiler{std::move(parser.get_parsed_ir())};

spirv_cross::ShaderResources resources = Compiler.get_shader_resources();
for (const auto& UB : resources.uniform_buffers)
{
    Compiler.get_binary_offset_for_decoration(UB.id, spv::Decoration::DecorationBinding, &BindingDecorationOffset);
    Compiler.get_binary_offset_for_decoration(UB.id, spv::Decoration::DecorationDescriptorSet, &DescrSetDecorationOffset);
    // Save decorations
}

for (const auto& SB : resources.storage_buffers)
    // Process other resource types
// Patch SPIRV using the assigned sets and bindings
for (size_t s = 0; s < ShaderStages.size(); ++s) // There may be multiple shaders in the stage in case of ray tracing
{
    const auto& Shaders = ShaderStages[s].Shaders;
    auto& SPIRVs = ShaderStages[s].SPIRVs;
    for (size_t i = 0; i < Shaders.size(); ++i)
    {
        auto* pShader = Shaders[i];
        auto& SPIRV = SPIRVs[i];
        for (const auto& Res : pShader->GetShaderResources())
        {
            const auto& ResAttribs = Layout.GetResourceAttribs(Res.Name);
            SPIRV[Res.DescriptorSetDecorationOffset] = ResAttribs.DescriptorSet;
        }
    }
}
Shader Resource Binding Model 1.0

Pipeline State
- Static Cache
- Layout

Copy static resources

SRB
- Res Cache

Static/Mutable VkDescriptorSet
- pResource
- pResource

<table>
<thead>
<tr>
<th>&quot;BaseColor&quot;</th>
<th>&quot;Normals&quot;</th>
<th>&quot;ColorCorrection&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set=0</td>
<td>Set=0</td>
<td>Set=1</td>
</tr>
<tr>
<td>Bind=5</td>
<td>Bind=6</td>
<td>Bind=1</td>
</tr>
<tr>
<td>Offset=5</td>
<td>Offset=13</td>
<td>Offset=1</td>
</tr>
</tbody>
</table>

"BaseColor"
Set=0
Bind=5
Offset=5

"Normals"
Set=0
Bind=6
Offset=13

"ColorCorrection"
Set=1
Bind=1
Offset=1
Shader Resource Binding Model 1.0

Committing Resources

- Set Static/Mutable descriptor set from the SRB (if any)
- Allocate dynamic descriptor set (if any)
  - Copy all dynamic descriptors from the SRB
- Write dynamic offsets and commit descriptor sets
Shader Resource Binding Model 2.0

Limitations of version 1.0:

- Shader resource binding objects of different pipelines are incompatible
  - SRB of pipeline 1 can only be used with pipeline 2 when they use the exact same resources

- All pipeline resources are consolidated
  - Can’t split resources based on the frequency of change (e.g. frame / pass / object)
Shader Resource Binding Model 2.0

Pipeline Resource Signature

- Defines resource layout independently of any pipeline state
- Each pipeline state may use up to 8 signatures
  - Descriptor sets of each signature stack on top of previous sets
- SRBs can be reused between pipelines
- Pipeline states take bindings from the resource signatures
Pipeline Resource Signature

```cpp
PipelineResourceSignatureDesc PRSDesc;
PRSDesc.Name = "Ray tracing scene resources";

const PipelineResourceDesc Resources[] = {
    {SHADER_TYPE_COMPUTE, "g_TLAS", 1, ACCEL_STRUCT, VARIABLE_TYPE_STATIC},
    {SHADER_TYPE_COMPUTE, "g_Constants", 1, CONSTANT_BUFFER, VARIABLE_TYPE_MUTABLE},
    {SHADER_TYPE_COMPUTE, "g_VertexBuffer", 1, BUFFER_SRV, VARIABLE_TYPE_DYNAMIC},
    {SHADER_TYPE_COMPUTE, "g_IndexBuffer", 1, BUFFER_SRV, VARIABLE_TYPE_DYNAMIC},
    {SHADER_TYPE_COMPUTE, "g_Textures", 8, TEXTURE_SRV, VARIABLE_TYPE_MUTABLE},
    {SHADER_TYPE_COMPUTE, "g_Samplers", 2, SAMPLER, VARIABLE_TYPE_STATIC}
};

PRSDesc.BindingIndex = 0;
PRSDesc.Resources = Resources;
PRSDesc.NumResources = 6;

pDevice->CreatePipelineResourceSignature(PRSDesc, &pRayTracingSign);
```
Pipeline Resource Signature

```cpp
IPipelineResourceSignature* ppSignatures[] =
{
    pRayTracingSign,
    pScreenResourcesSign
};
PSOCreateInfo.ppResourceSignatures = ppSignatures;
PSOCreateInfo.ResourceSignaturesCount = 2;
pDevice->CreateComputePipelineState(PSOCreateInfo, &pRayTracingPSO);
```
Shader Resource Binding Model 2.0

Example

// Signature 0 - Frame Constants
uniform FrameConstantsUB{}; // Static
uniform sampler2D SunShadowMap; // Static

// Signature 1 - Render Pass Constants
uniform CameraTransformUB{}; // Mutable
uniform sampler2D AmbientOcclusion; // Mutable

// Signature 2 - Object constants
uniform ObjectTransformUB {}; // Mutable
buffer LightsSB {}; // Dynamic
uniform sampler2D ObjectShadowMap; // Mutable
uniform sampler2D BaseColor; // Mutable
uniform sampler2D Normals; // Mutable

Signature 0 - Set 0 (0)
FrameConstantsUB
SunShadowMap

Signature 1 - Set 0 (1)
CameraTransformUB
AmbientOcclusion

Signature 2 - Set 0 (2)
ObjectTransformUB
ObjectShadowMap
BaseColor
Normals

Signature 2 - Set 1 (3)
LightsSB
Shader Resource Binding Model 2.0

Example

```cpp
pFrameSign->CreateShaderResourceBinding(&pFrameSRB, true);  
pCtx->CommitShaderResources(pFrameSRB);

for (size_t pass = 0; pass < PassCount; ++pass)
{
    pCtx->CommitShaderResources(pPassSRB);
    for (size_t obj = 0; obj < NumObjects; ++obj)
    {
        pCtx->SetPipelineState(pObjectPSO);
        pCtx->CommitShaderResources(pObjectSRB);
        pCtx->DrawIndexed();
    }
}
```

Uses `pFrameSign`, `pPassSign`, `pObjectSign`
Diligent Render State Notation

- JSON-based state description language
  - Shaders
  - Resource signatures
  - Pipeline states

- Can be parsed at run-time or off-line

- Off-line archiver tool processes all states and packages them into archive
  - Compiles shaders, defines layouts, patches byte code
  - Run-time loading is very fast
Diligent Render State Notation

"PSODesc": {
  "Name": "G-Buffer PSO",
  "ResourceLayout": {
    "Variables": [
      {
        "Name": "cbConstants",
        "ShaderStages": "PIXEL",
        "Type": "STATIC"
      }
    ]
  },
  "GraphicsPipeline": {
    "PrimitiveTopology": "TRIANGLE_LIST",
    "RasterizerDesc": {
      "CullMode": "NONE"
    },
    "DepthStencilDesc": {
      "DepthEnable": false
    }
  }
},

"pVS": {
  "Desc": {
    "Name": "Screen Triangle VS"
  },
  "FilePath": "screen_tri.vsh",
  "EntryPoint": "main"
},

"pPS": {
  "Desc": {
    "Name": "G-Buffer PS"
  },
  "FilePath": "g_buffer.psh",
  "EntryPoint": "main"}
Testing

- Render reference image using native API, compare with what Diligent renders
  - Robust to driver updates, API differences, different GPUs
- Intercept errors, fail tests if any detected
  - Run special tests to simulate error situations
- Run tests on each commit
  - Linux -> Vulkan/OpenGL
  - Windows -> Direct3D11/Direct3D12
- Run golden image tests for Tutorial and Sample applications
Links

- Vulkan Backend
- Resource Binding:
  - PipelineResourceSignatureVkImpl.cpp
  - PipelineStateVkImpl.cpp
  - ShaderResourceBindingVkImpl.cpp
  - SPIRVShaderResources.cpp
Thank you!
Backup
Shader Resource Binding

Immutable Samplers

- Backed into the pipeline state
- Don’t require sampler at run time
- More efficient

```c
ImmutableSamplerDesc ImtblSamplers[] = {
    {SHADER_TYPE_PIXEL, "Texture", LinearClamp}
};
ResourceLayout ImmutableSamplers = ImtblSamplers;
ResourceLayout NumImmutableSamplers = 1;
```
Shader Resource Binding

Other features

- Run-time resource arrays
  - PIPELINE_Resource_FLAG_RUNTIME_ARRAY
- Disallow dynamic offsets
  - PIPELINE_RESOURCE_FLAG_NO_DYNAMIC_BUFFERS
- Update mutable variables
  - SET_SHADERRESOURCE_FLAG_ALLOW_OVERWRITE

uniform sampler2D BaseColor[];
Multithreaded Command Recording

- Create one deferred context per thread
- Record commands in each thread using the RESOURCE_STATE_TRANSITION_MODE_NONE to disable automatic state handling
  - Manually issue synchronization commands
- Get command lists from each deferred context and execute them in immediate context
Multiple Command Queues

- Immediate context = command queue + command list + state (e.g., dynamic memory)
- Multiple command queues = multiple immediate contexts
- An application may run commands directly in each immediate context or record them through deferred contexts and execute
- Manual synchronization between contexts is done through fences