Rogue Robots

(stort spelprojekt)

by “Disorganized Games (DOG)”

Nadhif Ginola

Primary responsibility:
Rendering **backend, frontend, dev-tooling**

Contributions/Responsibilities Outline

- Render backend abstraction (Direct3D 12)
- (***) Render Graph
- Render frontend (ie. tech, front-facing dev-tools, etc.)
- Memory allocators
- Assisting render tech implementations by other contributors
- Solely responsible for the maintaining the renderer module(s)

(***) : Primary focus area

- Scrum Master (2 week period)
- Product Owner (2 week period)

Render Backend

- Handle based interface
  - 64 bit keys with generational counter
  - Keeps internal data structures fully hidden and flexible
- Bindless GPU resources
- Forced vertex pulling for geometry data (caveat: IB for vertex caching)
- Minimized API *surface area* by reducing available API primitives by **simplification** (i.e sync receipts) and **specialization** (i.e direct descriptor access)
- Sensible defaults for trivial API primitive construction (i.e pipelines)

```
class HandlePool
{
public:
    HandlePool();

    uint64_t allocate_handle();
    void free_handle(uint64_t handle);
```

```
Swapchain* CreateSwapchain(void* hwnd, u8 numBuffers);
Buffer CreateBuffer(const BufferDesc& desc, MemoryPool = {});
Texture CreateTexture(const TextureDesc& desc, MemoryPool = {});
Pipeline CreateGraphicsPipeline(const GraphicsPipelineDesc& desc);
Pipeline CreateComputePipeline(const ComputePipelineDesc& desc);
RenderPass CreateRenderPass(const RenderPassDesc& desc);
BufferView CreateView(Buffer buffer, const BufferViewDesc& desc);
TextureView CreateView(Texture texture, const TextureViewDesc& desc);
MemoryPool CreateMemoryPool(const MemoryPoolDesc& desc);

u32 GetGlobalDescriptor(BufferView view) const;
u32 GetGlobalDescriptor(TextureView view) const;

CommandList AllocateCommandList(QueueType queue = QueueType::Graphics);
void WaitForGPU(SyncReceipt receipt);

std::optional<SyncReceipt> SubmitCommandLists(
    std::span<CommandList> lists,
    QueueType queue = QueueType::Graphics,
    std::optional<SyncReceipt> incoming_sync = std::nullopt,        // Synchronize with receipt prior to execution
    bool generate_sync = false);                                    // Generate sync after command list execution
```

```
struct Buffer { u64 handle{ 0 }; };
struct Texture { u64 handle{ 0 }; };
struct Pipeline { u64 handle{ 0 }; };
struct RenderPass { u64 handle{ 0 }; };

struct SyncReceipt { u64 handle{ 0 }; };
struct BufferView { u64 handle{ 0 }; };
struct TextureView { u64 handle{ 0 }; };

struct CommandList {  u64 handle{ 0 }; };
struct MemoryPool { u64 handle{ 0 }; };
```

```
struct ShaderArgs
{
    static constexpr u32 MAX_CONSTANTS = 15;
    std::array<u32, MAX_CONSTANTS> constants{};
    u8 numConstants{ 0 };
    ShaderArgs& AppendConstant(u32 constant) { assert(numConstants < 15); constants[numConstants++] = constant; return *this; }
    ShaderArgs& SetPrimaryCBV(Buffer buf, u32 offset) { mainCBV = buf; mainCBVOffset = offset; return *this; };
```

```
void Cmd_UpdateShaderArgs(CommandList list,
    QueueType targetQueue,
    const ShaderArgs& args);
```

```
auto meshVS = m_sclr->CompileFromFile("MainVS.hlsl", ShaderType::Vertex);
auto meshPS = m_sclr->CompileFromFile("MainPS.hlsl", ShaderType::Pixel);
m_meshPipe = m_rd->CreateGraphicsPipeline(GraphicsPipelineBuilder()
    .SetShader(meshVS.get())
    .SetShader(meshPS.get())
    .AppendRTFormat(DXGI_FORMAT_R16G16B16A16_FLOAT)
    .SetDepthFormat(DepthFormat::D32)
    .SetDepthStencil(DepthStencilBuilder().SetDepthEnabled(true).SetDepthWriteMask(D3D12_DEPTH_WRITE_MASK_ZERO).SetDepthFunc(D3D12_COMPARISON_FUNC_EQUAL))
    .Build());
```

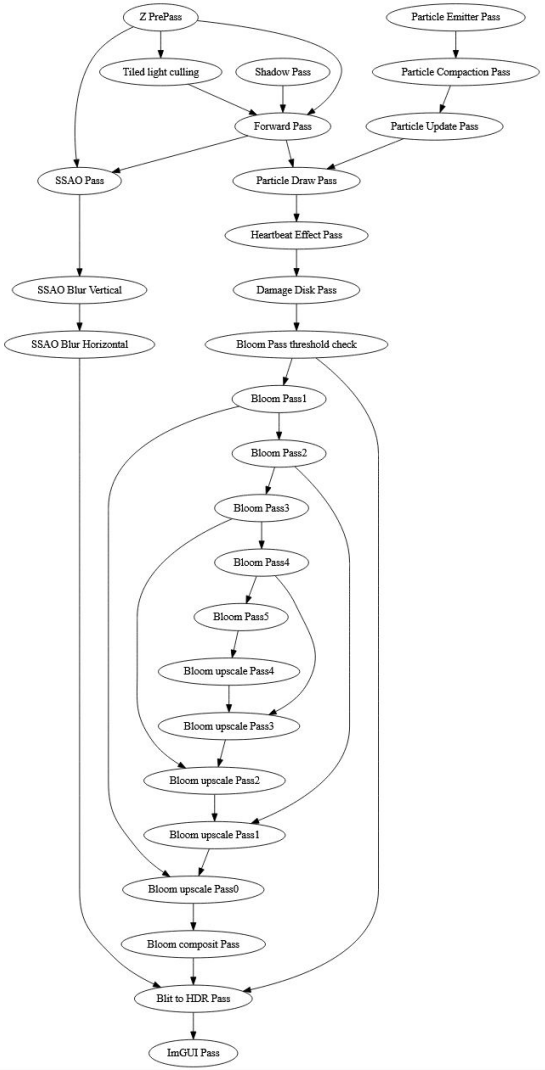
(***) Render Graph

My primary focus area in 'stort spelprojekt'

- In a nutshell → a **Task Graph** (Directed Acyclic Graph) with some domain-specific caveats

- Automatic barriers (transitions/UAV/aliasing)
- ***Graph-Level Resource Aliasing*** resolves cyclic dependencies (i.e write-then-write on a resource X)
- Parallelizable command list recording
    - I.e fork-join on tasks with the same depth
- **Memory aliasing** (naive)

- On-demand graph rebuilding
- Caveat
    - Single Queue
    - Design relies on Direct Descriptor Access
- Resources re-created only on graph rebuild



(***) Render Graph

My primary focus area in 'stort spelprojekt'

- Declarative API
  - Resource/usage declaration pass
  - Execution pass
- String-identified resources
- Allows externally **imported** resources
- RT/DS auto-setup internally for the declared pass

- Minimized developer burden while still providing relevant low-level control
- User-defined per-pass data

- Contributors can focus more on the *important parts* with render tech implementations😎

Shadow Pass example

```
rg.AddPass<ShadowPassData>("Shadow Pass",
    [&](ShadowPassData&, RenderGraph::PassBuilder& builder)
    {
        builder.DeclareTexture(RG_RESOURCE(ShadowDepth), RGTextureDesc::DepthWrite2D(DepthFormat::D32, 1024, 1024, m_shadowMapCapacity));
        builder.WriteDepthStencil(RG_RESOURCE(ShadowDepth), RenderPassAccessType::ClearPreserve,
            TextureViewDesc(ViewType::DepthStencil, TextureViewDimension::Texture2D_Array, DXGI_FORMAT_D32_FLOAT)
            .SetArrayRange(0, m_shadowMapCapacity));

    },
    [&, shadowDrawFunc = shadowDrawSubmissions](const ShadowPassData&, RenderDevice* rd, CommandList cmdl, RenderGraph::PassResources&) mutable
    {
        rd->Cmd_SetViewports(cmdl, Viewports().Append(0.f, 0.f, 1024.f, 1024.f));
        rd->Cmd_SetScissorRects(cmdl, ScissorRects().Append(0, 0, 1024, 1024));

        rd->Cmd_SetIndexBuffer(cmdl, m_globalEffectData.meshTable->GetIndexBuffer());

        // Fills shadowmaps chronologically
        rd->Cmd_SetPipeline(cmdl, m_shadowPipe);
        u32 nextMap = 0;
        for (u32 i = 0; i < m_activeSpotlights.size(); ++i)
        {
            if (m_activeSpotlights[i].shadow)
                shadowDrawFunc(rd, cmdl, m_singleSidedShadowDraws[m_activeShadowCasters[i].singleSidedBucket], nextMap++, m_activeSpotlights[i].shadow.value());
        }
    });
```

Compute example

```
rg.AddPass<PassData>("SSAO Pass",
    [&](PassData& passData, RenderGraph::PassBuilder& builder)
    {
        // Compute read access
        passData.depth = builder.ReadResource(RG_RESOURCE(MainDepth), D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE,
            TextureViewDesc(ViewType::ShaderResource, TextureViewDimension::Texture2D, DXGI_FORMAT_R32_FLOAT));
        passData.nor = builder.ReadResource(RG_RESOURCE(MainNormals), D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE,
            TextureViewDesc(ViewType::ShaderResource, TextureViewDimension::Texture2D, DXGI_FORMAT_R16G16B16A16_FLOAT));
        passData.noise = builder.ReadResource(RG_RESOURCE(NoiseSSAO), D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE,
            TextureViewDesc(ViewType::ShaderResource, TextureViewDimension::Texture2D, DXGI_FORMAT_R32G32B32A32_FLOAT));
        passData.samples = builder.ReadResource(RG_RESOURCE(SamplesSSAO), D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE,
            BufferViewDesc(ViewType::ShaderResource, 0, sizeof(DirectX::SimpleMath::Vector4), 64));

        // screen space AO texture
        builder.DeclareTexture(RG_RESOURCE(AmbientOcclusion), RGTextureDesc::ReadWrite2D(DXGI_FORMAT_R16G16B16A16_FLOAT, m_renderWidth, m_renderHeight));
        passData.aoOut = builder.ReadWriteTarget(RG_RESOURCE(AmbientOcclusion),
            TextureViewDesc(ViewType::UnorderedAccess, TextureViewDimension::Texture2D, DXGI_FORMAT_R16G16B16A16_FLOAT));
    },
    [&](const PassData& passData, RenderDevice* rd, CommandList cmdl, RenderGraph::PassResources& resources)
    {
        if (m_graphicsSettings.ssao)
        {
            rd->Cmd_SetPipeline(cmdl, m_ssaoPipe);
            auto args = ShaderArgs()
                .AppendConstant(m_globalEffectData.globalDataDescriptor)
                .AppendConstant(m_currPfDescriptor)
                .AppendConstant(m_renderWidth)
                .AppendConstant(m_renderHeight)
                .AppendConstant(resources.GetView(passData.aoOut))
                .AppendConstant(resources.GetView(passData.depth))
                .AppendConstant(resources.GetView(passData.nor))
                .AppendConstant(resources.GetView(passData.noise))
                .AppendConstant(resources.GetView(passData.samples));
            rd->Cmd_UpdateShaderArgs(cmdl, QueueType::Compute, args);
            rd->Cmd_ClearUnorderedAccessFLOAT(cmdl,
                resources.GetTextureView(passData.aoOut), { 0.f, 0.f, 0.f, 1.f }, ScissorRects().Append(0, 0, m_renderWidth, m_renderHeight));
            auto xGroup = (u32)std::ceilf(m_renderWidth / 32.f);
            auto yGroup = (u32)std::ceilf(m_renderHeight / 32.f);
            rd->Cmd_Dispatch(cmdl, xGroup, yGroup, 1);
```

```
struct PassData
{
    RGResourceView noise, samples;
    RGResourceView depth;
    RGResourceView nor;
    RGResourceView aoOut;
};
```

Render Frontend

- PBR (Metallic-Roughness workflow) + Emissive
- Normal-mapping

→ *Game vision changed to low-poly* ***after..***

- Screen Space Ambient Occlusion
- Z prepass
- Static and Dynamic point & spotlights
- Damage indicator (Gameplay)
- Low health indicator (Gameplay)

Render Frontend Miscellaneous

- **Assisted** renderer-related implementations by **other contributors**: Shadow mapping, Forward+, Particles, Bloom, Skeletal Animation, UI

- Memory allocators (ring, buffer, pool)
  - Virtual allocators for GPU resources included
- Helpers to..
  - Trivially enqueue in-flight resources for safe deletion
  - Trivially upload data to device-local memory
- Scene geometry stored in a single VB/IB pair

- BC7 textures & mipmap-gen (DXTex)
- ImGUI
- Tracy profiler

Other

- Setup material prefabs in Lua for instantiation
- Grenade explosion effect through Lua
- Created systems to hook the visual effects (i.e damage/low-hp indicator) to gameplay

- Contribute to and discuss overarching architecture