Athena Renderer - 完全開発ドキュメント

📘 目次

概要

Athena Rendererは、グラフィックス研究のために設計されたDirectX 12ベースの軽量レンダリングフレームワークです。知恵の女神アテナの名を冠し、研究者が新しいレンダリング技術を探求し、実験し、論文化するための「知的な実験場」を提供します。

特徴

✓ モジュラーアーキテクチャ
 - レンダーグラフによる柔軟なパス構成
 - 基盤を変更せずに新機能を追加可能

✓ 研究指向設計
 - 迅速なプロトタイピング
 - 公平な比較実験
 - 論文執筆に最適

✓ 高性能
 - DirectX 12の最新機能をフル活用
 - GPU駆動レンダリング対応
 - 詳細なプロファイリング

✓ 使いやすさ

- ImGuiベースの実験制御UI
- 設定ファイルベースの実験管理
- 自動ベンチマーク機能

対象ユーザー

- グラフィックス研究者
- 大学院生・学部生
- ゲームエンジン開発者
- DirectX 12学習者

システム要件

OS: Windows 10/11 (64-bit)
GPU: DirectX 12対応GPU
- NVIDIA: GTX 1060以上推奨
- AMD: RX 5700以上推奨
- Intel: Arc A750以上推奨

RAM: 8GB以上 (16GB推奨)
VRAM: 4GB以上 (8GB推奨)
IDE: Visual Studio 2022
SDK: Windows SDK 10.0.22000.0以上

コンセプト

設計哲学

"研究者が基盤を気にせず、アイデアに集中できる環境"

【原則】
1. 関心の分離 (Separation of Concerns)
   - 各レイヤーは独立した責任
   - 低レベルと高レベルの明確な分離

2. 開放閉鎖原則 (Open-Closed Principle)
   - 拡張に対して開いている
   - 修正に対して閉じている

3. 依存性注入 (Dependency Injection)
   - 依存関係は外部から注入
   - テスタビリティの向上

Athena Rendererの位置づけ

【比較】

Unity/Unreal Engine
├─ 完全なゲーム開発環境
├─ ブラックボックスが多い
└─ カスタマイズに制限

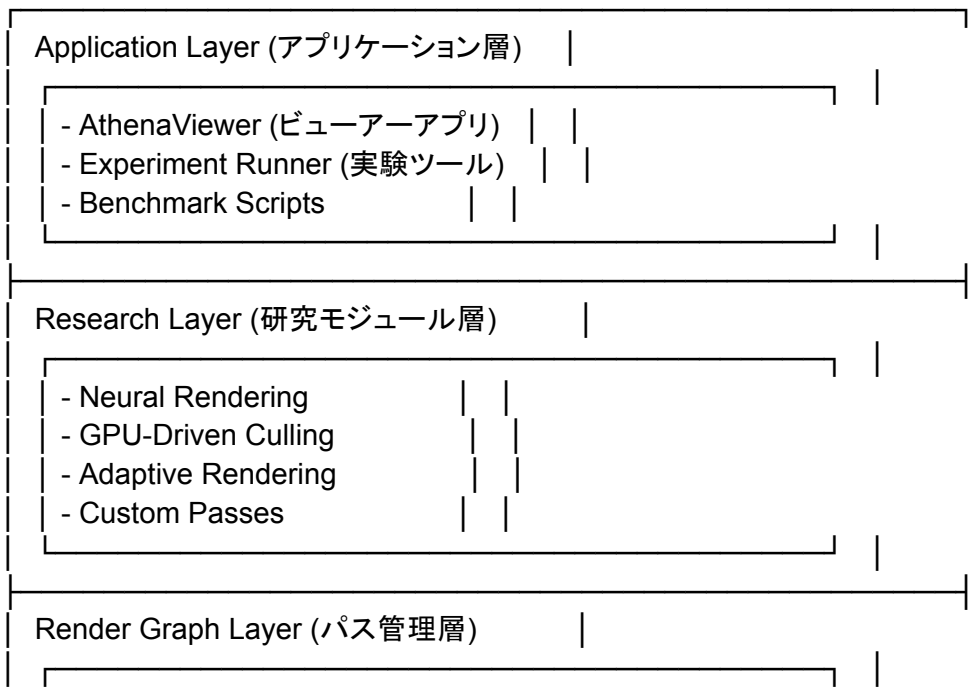
Falcor (NVIDIA)
├─ 研究用フレームワーク
├─ 大規模・機能豊富
└─ 学習コストが高い

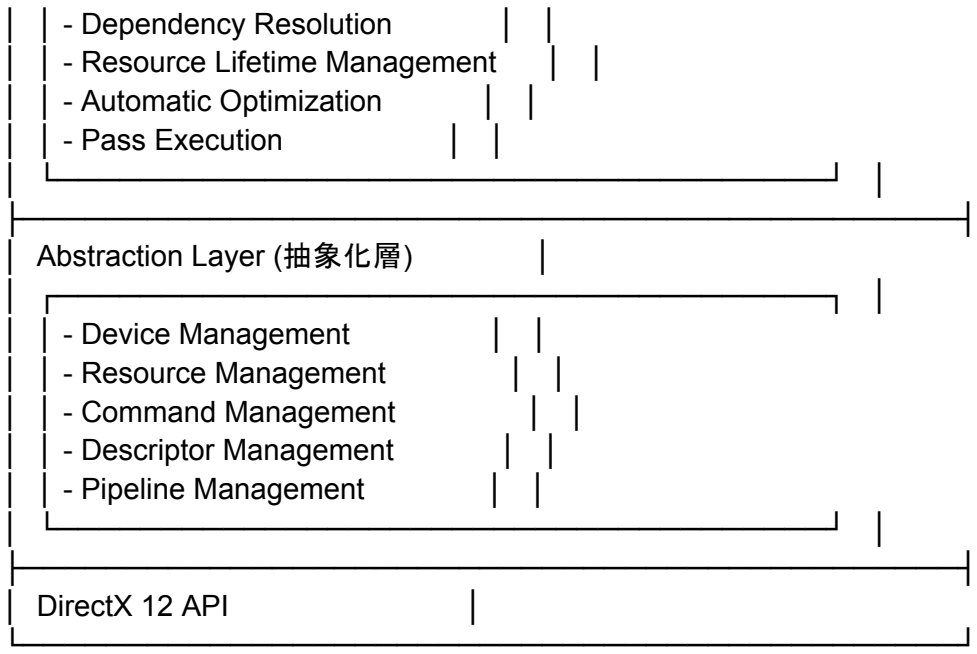
Athena Renderer ← あなたが作るもの
├─ 研究用の実験台
├─ 軽量・カスタマイズ重視
├─ 理解しやすい規模
└─ DirectX 12の深い理解

【用途】
✓ 新しいレンダリング技術の研究
✓ 学術論文の実装基盤
✓ DirectX 12の学習
✓ ポートフォリオ作品

システムアーキテクチャ

レイヤー構造





データフロー

初期化フェーズ:
Device → CommandQueue → SwapChain → DescriptorHeaps

レンダリングフェーズ:
RenderGraph.Compile()
↓
依存関係解決
↓
リソースライフタイム解析
↓
メモリ最適化
↓
RenderGraph.Execute()
↓
各Pass.Execute(context)
↓
SwapChain.Present()

プロジェクト構成

ディレクトリ構造

```
athena-renderer/
├─ AthenaRenderer.sln              # Visual Studioソリューション
│
```

```
├─ AthenaCore/                    # コアライブラリプロジェクト
│  ├─ AthenaCore.vcxproj
│  ├─ AthenaCore.vcxproj.filters
│  │
│  ├─ include/
│  │  └─ Athena/
│  │     ├─ Core/
│  │     │  ├─ Device.h            # デバイス管理
│  │     │  ├─ CommandQueue.h      # コマンドキュー
│  │     │  ├─ CommandList.h       # コマンドリスト
│  │     │  ├─ SwapChain.h         # スワップチェーン
│  │     │  ├─ DescriptorHeap.h    # デスクリプタ管理
│  │     │  └─ Fence.h             # 同期機構
│  │     │
│  │     ├─ Resources/
│  │     │  ├─ Buffer.h            # バッファ抽象化
│  │     │  ├─ Texture.h           # テクスチャ抽象化
│  │     │  ├─ ResourceManager.h   # リソース管理
│  │     │  └─ UploadContext.h     # アップロード管理
│  │     │
│  │     ├─ Rendering/
│  │     │  ├─ Shader.h            # シェーダー管理
│  │     │  ├─ Pipeline.h          # PSO管理
│  │     │  ├─ RootSignature.h     # ルートシグネチャ
│  │     │  └─ RenderContext.h     # レンダリングコンテキスト
│  │     │
│  │     ├─ RenderGraph/
│  │     │  ├─ RenderGraph.h       # グラフ本体
│  │     │  ├─ RenderPass.h        # パス基底クラス
│  │     │  ├─ RenderGraphBuilder.h # グラフ構築
│  │     │  ├─ ResourceHandle.h    # リソースハンドル
│  │     │  └─ PassRegistry.h      # パス登録
│  │     │
│  │     ├─ Scene/
│  │     │  ├─ Mesh.h              # メッシュ
│  │     │  ├─ Material.h          # マテリアル
│  │     │  ├─ Camera.h            # カメラ
│  │     │  ├─ Light.h             # ライト
│  │     │  └─ Scene.h             # シーン管理
│  │     │
│  │     ├─ Utils/
│  │     │  ├─ Math.h              # 数学ライブラリ
│  │     │  ├─ Timer.h             # タイマー
│  │     │  ├─ Logger.h            # ログ
│  │     │  └─ Profiler.h          # プロファイラ
│  │     │
│  │     └─ Athena.h               # 統合ヘッダー
│  │
```

```
│   └── src/
│       └── Athena/
│           ├── Core/
│           │   ├── Device.cpp
│           │   ├── CommandQueue.cpp
│           │   ├── CommandList.cpp
│           │   ├── SwapChain.cpp
│           │   ├── DescriptorHeap.cpp
│           │   └── Fence.cpp
│           │
│           ├── Resources/
│           │   ├── Buffer.cpp
│           │   ├── Texture.cpp
│           │   ├── ResourceManager.cpp
│           │   └── UploadContext.cpp
│           │
│           ├── Rendering/
│           │   ├── Shader.cpp
│           │   ├── Pipeline.cpp
│           │   ├── RootSignature.cpp
│           │   └── RenderContext.cpp
│           │
│           ├── RenderGraph/
│           │   ├── RenderGraph.cpp
│           │   ├── RenderPass.cpp
│           │   ├── RenderGraphBuilder.cpp
│           │   ├── ResourceHandle.cpp
│           │   └── PassRegistry.cpp
│           │
│           ├── Scene/
│           │   ├── Mesh.cpp
│           │   ├── Material.cpp
│           │   ├── Camera.cpp
│           │   ├── Light.cpp
│           │   └── Scene.cpp
│           │
│           └── Utils/
│               ├── Timer.cpp
│               ├── Logger.cpp
│               └── Profiler.cpp
│
├── AthenaViewer/                  # ビューアーアプリ
│   ├── AthenaViewer.vcxproj
│   ├── main.cpp                   # エントリーポイント
│   ├── ViewerApp.h
│   ├── ViewerApp.cpp              # メインアプリケーション
│   │
│   ├── UI/
```

```
│   │   ├── ImGuiManager.h
│   │   ├── ImGuiManager.cpp          # ImGui管理
│   │   ├── DebugPanel.h
│   │   ├── DebugPanel.cpp            # デバッグUI
│   │   ├── PerformancePanel.h
│   │   └── PerformancePanel.cpp      # パフォーマンスUI
│   │
│   ├── Scenes/
│   │   ├── TestScene.h
│   │   ├── TestScene.cpp             # テストシーン
│   │   ├── BenchmarkScene.h
│   │   └── BenchmarkScene.cpp        # ベンチマーク用
│   │
│   └── Passes/                       # 標準レンダーパス
│       ├── GeometryPass.h
│       ├── GeometryPass.cpp
│       ├── LightingPass.h
│       ├── LightingPass.cpp
│       ├── ToneMappingPass.h
│       └── ToneMappingPass.cpp
│
├── AthenaResearch/                   # 研究モジュール(オプション)
│   ├── AthenaResearch.vcxproj
│   │
│   ├── NeuralRendering/
│   │   ├── NeuralGIPass.h
│   │   ├── NeuralGIPass.cpp
│   │   └── shaders/
│   │       └── neural_inference.hlsl
│   │
│   ├── GPUDriven/
│   │   ├── GPUCullingPass.h
│   │   ├── GPUCullingPass.cpp
│   │   └── shaders/
│   │       └── culling.hlsl
│   │
│   └── AdaptiveRendering/
│       ├── MotionVectorPass.h
│       ├── MotionVectorPass.cpp
│       └── shaders/
│           └── motion.hlsl
│
├── shaders/                          # 共通シェーダー
│   ├── Common.hlsli                  # 共通ヘッダー
│   ├── VertexLayouts.hlsli           # 頂点レイアウト
│   ├── Lighting.hlsli                # ライティング関数
│   ├── BasicVS.hlsl                  # 基本頂点シェーダー
│   ├── BasicPS.hlsl                  # 基本ピクセルシェーダー
```

```
│   ├── DeferredLightingPS.hlsl       # ディファードライティング
│   ├── ToneMappingPS.hlsl            # トーンマッピング
│   └── FullscreenVS.hlsl             # フルスクリーンクワッド
│
├── configs/                          # 設定ファイル
│   ├── standard.json                 # 標準パイプライン
│   ├── deferred.json                 # ディファード
│   ├── forward_plus.json             # Forward+
│   ├── neural_gi.json                # ニューラルGI実験
│   └── gpu_driven.json               # GPU駆動実験
│
├── experiments/                      # 実験スクリプト
│   ├── benchmark_runner.cpp          # ベンチマーク実行
│   ├── experiment_config.json        # 実験設定
│   └── results/                      # 実験結果保存先
│       ├── csv/
│       ├── graphs/
│       └── screenshots/
│
├── external/                         # 外部ライブラリ
│   ├── imgui/                        # Dear ImGui (submodule)
│   ├── stb/                          # STB (submodule)
│   └── json/                         # nlohmann/json (submodule)
│
├── assets/                           # アセット
│   ├── models/                       # 3Dモデル
│   │   ├── sponza/
│   │   ├── bistro/
│   │   └── test_objects/
│   ├── textures/                     # テクスチャ
│   └── fonts/                        # フォント
│       └── Roboto-Regular.ttf
│
├── docs/                             # ドキュメント
│   ├── GettingStarted.md             # 入門ガイド
│   ├── Architecture.md               # アーキテクチャ
│   ├── RenderGraph.md                # レンダーグラフ
│   ├── AddingPasses.md               # パス追加方法
│   ├── Benchmarking.md               # ベンチマーク
│   └── API/                          # API詳細
│       ├── Core.md
│       ├── RenderGraph.md
│       └── Resources.md
│
├── AthenaCommon.props                # 共通プロパティシート
├── .gitignore
├── .gitmodules                       # Submodule設定
├── README.md
```

└─ LICENSE

環境構築

必要なソフトウェア

1. Visual Studio 2022
   - ワークロード: C++によるデスクトップ開発
   - コンポーネント:
     ✓ MSVC v143
     ✓ Windows 10/11 SDK (最新)
     ✓ C++ CMake tools (オプション)

2. Windows SDK
   - バージョン: 10.0.22000.0以上
   - DirectX 12 Agility SDK推奨

3. Git
   - Submoduleで外部ライブラリを管理

4. オプション:
   - PIX for Windows (GPUデバッグ)
   - RenderDoc (GPUデバッグ)
   - NVIDIA Nsight Graphics (NVIDIA GPU)

リポジトリのセットアップ

```
# リポジトリをクローン
git clone https://github.com/yourname/athena-renderer.git
cd athena-renderer

# Submoduleを初期化
git submodule update --init --recursive

# Visual Studioでソリューションを開く
start AthenaRenderer.sln
```

外部ライブラリの設定

NuGetパッケージ

Visual Studioで以下をインストール:

ツール → NuGetパッケージマネージャー →

ソリューションのNuGetパッケージの管理

推奨パッケージ:
- Microsoft.Direct3D.D3D12 (Agility SDK)
- DirectXTex_Desktop_2019
- Assimp (オプション)
- nlohmann.json

Git Submodule
Dear ImGui
cd external
git submodule add https://github.com/ocornut/imgui.git imgui

STB (画像ローダー)
git submodule add https://github.com/nothings/stb.git stb

nlohmann/json
git submodule add https://github.com/nlohmann/json.git json

プロパティシートの設定

AthenaCommon.propsを作成:

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup Label="UserMacros">
    <AthenaRootDir>$(SolutionDir)</AthenaRootDir>
    <AthenaIncludeDir>$(AthenaRootDir)AthenaCore\include</AthenaIncludeDir>
    <ExternalDir>$(AthenaRootDir)external</ExternalDir>
    <BinDir>$(AthenaRootDir)bin\$(Platform)\$(Configuration)\</BinDir>
    <ObjDir>$(AthenaRootDir)obj\$(Platform)\$(Configuration)\$(ProjectName)\</ObjDir>
  </PropertyGroup>

  <PropertyGroup>
    <OutDir>$(BinDir)</OutDir>
    <IntDir>$(ObjDir)</IntDir>
  </PropertyGroup>

  <ItemDefinitionGroup>
    <ClCompile>
      <LanguageStandard>stdcpp20</LanguageStandard>
      <LanguageStandard_C>stdc17</LanguageStandard_C>
      <MultiProcessorCompilation>true</MultiProcessorCompilation>
      <WarningLevel>Level4</WarningLevel>
      <AdditionalIncludeDirectories>
        $(AthenaIncludeDir);
```

```
      $(ExternalDir)\imgui;
      $(ExternalDir)\imgui\backends;
      $(ExternalDir)\stb;
      $(ExternalDir)\json\include;
      %(AdditionalIncludeDirectories)
    </AdditionalIncludeDirectories>
    <PreprocessorDefinitions>
      WIN32;
      _WINDOWS;
      UNICODE;
      _UNICODE;
      %(PreprocessorDefinitions)
    </PreprocessorDefinitions>
  </ClCompile>
 </ItemDefinitionGroup>

 <ItemDefinitionGroup Condition="'$(Configuration)'=='Debug'">
  <ClCompile>
   <Optimization>Disabled</Optimization>

<PreprocessorDefinitions>_DEBUG;%(PreprocessorDefinitions)</PreprocessorDefinitions>
    <RuntimeLibrary>MultiThreadedDebugDLL</RuntimeLibrary>
  </ClCompile>
 </ItemDefinitionGroup>

 <ItemDefinitionGroup Condition="'$(Configuration)'=='Release'">
  <ClCompile>
   <Optimization>MaxSpeed</Optimization>

<PreprocessorDefinitions>NDEBUG;%(PreprocessorDefinitions)</PreprocessorDefinitions>
    <RuntimeLibrary>MultiThreadedDLL</RuntimeLibrary>
    <FunctionLevelLinking>true</FunctionLevelLinking>
    <IntrinsicFunctions>true</IntrinsicFunctions>
  </ClCompile>
  <Link>
   <EnableCOMDATFolding>true</EnableCOMDATFolding>
   <OptimizeReferences>true</OptimizeReferences>
  </Link>
 </ItemDefinitionGroup>
</Project>
```

ビルド設定

AthenaCore プロジェクト
プロジェクトの種類: Static Library (.lib)

【全般】

- 構成の種類: スタティックライブラリ (.lib)
- C++言語標準: C++20
- Windows SDK バージョン: 最新

【C/C++ → プリプロセッサ】
- 追加:
  ATHENA_CORE_EXPORTS

【ライブラリアン → 全般】
- 出力ファイル: $(OutDir)$(TargetName)$(TargetExt)

【ビルドイベント → ビルド後】
- コマンドライン:
  xcopy /Y /D "$(ProjectDir)include*" "$(OutDir)include\" /S /I

AthenaViewer プロジェクト
プロジェクトの種類: Application (.exe)

【全般】
- 構成の種類: アプリケーション (.exe)
- サブシステム: Windows

【C/C++】
- 追加のインクルードディレクトリ:
  $(SolutionDir)AthenaCore\include;
  %(AdditionalIncludeDirectories)

【リンカー → 入力】
- 追加の依存ファイル:
  AthenaCore.lib;
  d3d12.lib;
  dxgi.lib;
  d3dcompiler.lib;
  dxguid.lib;
  %(AdditionalDependencies)

【リンカー → 全般】
- 追加のライブラリディレクトリ:
  $(OutDir);
  %(AdditionalLibraryDirectories)

【ビルドイベント → ビルド後】
- コマンドライン:
  xcopy /Y /D "$(SolutionDir)shaders*.*" "$(OutDir)shaders\" /S /I
  xcopy /Y /D "$(SolutionDir)configs*.*" "$(OutDir)configs\" /S /I
  xcopy /Y /D "$(SolutionDir)assets*.*" "$(OutDir)assets\" /S /I

Dear ImGuiのビルド

AthenaViewerプロジェクトに以下のファイルを追加:

外部ライブラリ (フィルター作成)
├─ imgui/
  ├─ imgui.cpp
  ├─ imgui_demo.cpp
  ├─ imgui_draw.cpp
  ├─ imgui_tables.cpp
  ├─ imgui_widgets.cpp
  └─ backends/
    ├─ imgui_impl_win32.cpp
    └─ imgui_impl_dx12.cpp

コア実装

1. Device (デバイス管理)

include/Athena/Core/Device.h

```
#pragma once
#include <d3d12.h>
#include <dxgi1_6.h>
#include <wrl/client.h>

namespace Athena {

using Microsoft::WRL::ComPtr;

class Device {
public:
    Device();
    ~Device();

    // 初期化・終了
    void Initialize(bool enableDebugLayer = true);
    void Shutdown();

    // アクセサ
    ID3D12Device5* GetD3D12Device() const { return device.Get(); }
    IDXGIFactory6* GetDXGIFactory() const { return factory.Get(); }
    IDXGIAdapter4* GetAdapter() const { return adapter.Get(); }

    // 機能サポート確認
```

```
    bool SupportsMeshShaders() const { return supportMeshShaders; }
    bool SupportsRaytracing() const { return supportRaytracing; }
    bool SupportsVariableRateShading() const { return supportVRS; }

private:
    ComPtr<ID3D12Device5> device;
    ComPtr<IDXGIFactory6> factory;
    ComPtr<IDXGIAdapter4> adapter;

    bool supportMeshShaders = false;
    bool supportRaytracing = false;
    bool supportVRS = false;

    void EnableDebugLayer();
    void SelectBestAdapter();
    void CheckFeatureSupport();
};

} // namespace Athena
```

src/Athena/Core/Device.cpp

```
#include "Athena/Core/Device.h"
#include "Athena/Utils/Logger.h"
#include <stdexcept>

namespace Athena {

Device::Device() = default;
Device::~Device() = default;

void Device::Initialize(bool enableDebugLayer) {
    Logger::Info("Initializing Device...");

    if (enableDebugLayer) {
        EnableDebugLayer();
    }

    // DXGIファクトリ作成
    UINT flags = 0;
    if (enableDebugLayer) {
        flags = DXGI_CREATE_FACTORY_DEBUG;
    }

    HRESULT hr = CreateDXGIFactory2(flags, IID_PPV_ARGS(&factory));
    if (FAILED(hr)) {
        throw std::runtime_error("Failed to create DXGI factory");
```

```
    }

    // 最適なアダプター選択
    SelectBestAdapter();

    // デバイス作成
    hr = D3D12CreateDevice(
        adapter.Get(),
        D3D_FEATURE_LEVEL_12_0,
        IID_PPV_ARGS(&device)
    );

    if (FAILED(hr)) {
        throw std::runtime_error("Failed to create D3D12 device");
    }

    // 機能サポートチェック
    CheckFeatureSupport();

    Logger::Info("Device initialized successfully");
}

void Device::Shutdown() {
    Logger::Info("Shutting down Device...");

    device.Reset();
    adapter.Reset();
    factory.Reset();
}

void Device::EnableDebugLayer() {
#ifdef _DEBUG
    ComPtr<ID3D12Debug> debugController;
    if (SUCCEEDED(D3D12GetDebugInterface(IID_PPV_ARGS(&debugController)))) {
        debugController->EnableDebugLayer();
        Logger::Info("D3D12 Debug Layer enabled");
    }
#endif
}

void Device::SelectBestAdapter() {
    ComPtr<IDXGIAdapter1> tempAdapter;
    SIZE_T maxVideoMemory = 0;

    for (UINT i = 0;
         factory->EnumAdapters1(i, &tempAdapter) != DXGI_ERROR_NOT_FOUND;
         ++i) {
```

```
        DXGI_ADAPTER_DESC1 desc;
        tempAdapter->GetDesc1(&desc);

        // ソフトウェアアダプターをスキップ
        if (desc.Flags & DXGI_ADAPTER_FLAG_SOFTWARE) {
            continue;
        }

        // D3D12対応チェック
        if (SUCCEEDED(D3D12CreateDevice(
            tempAdapter.Get(),
            D3D_FEATURE_LEVEL_12_0,
            __uuidof(ID3D12Device), nullptr))) {

            // 最大VRAMのアダプターを選択
            if (desc.DedicatedVideoMemory > maxVideoMemory) {
                maxVideoMemory = desc.DedicatedVideoMemory;
                tempAdapter.As(&adapter);
            }
        }
    }

    if (!adapter) {
        throw std::runtime_error("No compatible adapter found");
    }

    DXGI_ADAPTER_DESC1 desc;
    adapter->GetDesc1(&desc);
    Logger::Info("Selected GPU: %S", desc.Description);
    Logger::Info("VRAM: %.2f GB", desc.DedicatedVideoMemory / (1024.0 * 1024.0 *
1024.0));
}

void Device::CheckFeatureSupport() {
    // Mesh Shader対応
    D3D12_FEATURE_DATA_D3D12_OPTIONS7 options7 = {};
    if (SUCCEEDED(device->CheckFeatureSupport(
        D3D12_FEATURE_D3D12_OPTIONS7,
        &options7, sizeof(options7)))) {
        supportMeshShaders = (options7.MeshShaderTier >=
D3D12_MESH_SHADER_TIER_1);
    }

    // Raytracing対応
    D3D12_FEATURE_DATA_D3D12_OPTIONS5 options5 = {};
    if (SUCCEEDED(device->CheckFeatureSupport(
        D3D12_FEATURE_D3D12_OPTIONS5,
        &options5, sizeof(options5)))) {
```

```
        supportRaytracing = (options5.RaytracingTier >= D3D12_RAYTRACING_TIER_1_0);
    }

    // VRS対応
    D3D12_FEATURE_DATA_D3D12_OPTIONS6 options6 = {};
    if (SUCCEEDED(device->CheckFeatureSupport(
        D3D12_FEATURE_D3D12_OPTIONS6,
        &options6, sizeof(options6)))) {
        supportVRS = (options6.VariableShadingRateTier >=
D3D12_VARIABLE_SHADING_RATE_TIER_1);
    }

    Logger::Info("Feature Support:");
    Logger::Info("  Mesh Shaders: %s", supportMeshShaders ? "Yes" : "No");
    Logger::Info("  Raytracing: %s", supportRaytracing ? "Yes" : "No");
    Logger::Info("  VRS: %s", supportVRS ? "Yes" : "No");
}

} // namespace Athena
```

2. CommandQueue (コマンドキュー)

include/Athena/Core/CommandQueue.h

```
#pragma once
#include <d3d12.h>
#include <wrl/client.h>
#include <cstdint>

namespace Athena {

using Microsoft::WRL::ComPtr;

class CommandQueue {
public:
    CommandQueue();
    ~CommandQueue();

    void Initialize(ID3D12Device* device, D3D12_COMMAND_LIST_TYPE type);
    void Shutdown();

    // コマンド実行
    void ExecuteCommandLists(ID3D12CommandList* const* commandLists, uint32_t count);

    // 同期
    void WaitForGPU();
    void Signal();
```

```
    uint64_t GetCompletedValue() const;

    // アクセサ
    ID3D12CommandQueue* GetD3D12CommandQueue() const { return queue.Get(); }

private:
    ComPtr<ID3D12CommandQueue> queue;
    ComPtr<ID3D12Fence> fence;
    HANDLE fenceEvent = nullptr;
    uint64_t fenceValue = 0;
};

} // namespace Athena
```

src/Athena/Core/CommandQueue.cpp

```
#include "Athena/Core/CommandQueue.h"
#include "Athena/Utils/Logger.h"
#include <stdexcept>

namespace Athena {

CommandQueue::CommandQueue() = default;

CommandQueue::~CommandQueue() {
    if (fenceEvent) {
        CloseHandle(fenceEvent);
    }
}

void CommandQueue::Initialize(ID3D12Device* device, D3D12_COMMAND_LIST_TYPE
type) {
    // コマンドキュー作成
    D3D12_COMMAND_QUEUE_DESC queueDesc = {};
    queueDesc.Type = type;
    queueDesc.Priority = D3D12_COMMAND_QUEUE_PRIORITY_NORMAL;
    queueDesc.Flags = D3D12_COMMAND_QUEUE_FLAG_NONE;
    queueDesc.NodeMask = 0;

    HRESULT hr = device->CreateCommandQueue(&queueDesc, IID_PPV_ARGS(&queue));
    if (FAILED(hr)) {
        throw std::runtime_error("Failed to create command queue");
    }

    // フェンス作成
    hr = device->CreateFence(0, D3D12_FENCE_FLAG_NONE, IID_PPV_ARGS(&fence));
    if (FAILED(hr)) {
```

```
        throw std::runtime_error("Failed to create fence");
    }

    // イベント作成
    fenceEvent = CreateEvent(nullptr, FALSE, FALSE, nullptr);
    if (!fenceEvent) {
        throw std::runtime_error("Failed to create fence event");
    }

    Logger::Info("CommandQueue initialized");
}

void CommandQueue::Shutdown() {
    WaitForGPU();

    if (fenceEvent) {
        CloseHandle(fenceEvent);
        fenceEvent = nullptr;
    }

    fence.Reset();
    queue.Reset();
}

void CommandQueue::ExecuteCommandLists(ID3D12CommandList* const* commandLists,
uint32_t count) {
    queue->ExecuteCommandLists(count, commandLists);
}

void CommandQueue::Signal() {
    ++fenceValue;
    queue->Signal(fence.Get(), fenceValue);
}

void CommandQueue::WaitForGPU() {
    Signal();

    if (fence->GetCompletedValue() < fenceValue) {
        fence->SetEventOnCompletion(fenceValue, fenceEvent);
        WaitForSingleObject(fenceEvent, INFINITE);
    }
}

uint64_t CommandQueue::GetCompletedValue() const {
    return fence->GetCompletedValue();
}

} // namespace Athena
```

3. SwapChain (スワップチェーン)

include/Athena/Core/SwapChain.h

```
#pragma once
#include <d3d12.h>
#include <dxgi1_6.h>
#include <wrl/client.h>
#include <cstdint>
#include <vector>

namespace Athena {

using Microsoft::WRL::ComPtr;

class SwapChain {
public:
    static constexpr uint32_t BufferCount = 3; // トリプルバッファ

    SwapChain();
    ~SwapChain();

    void Initialize(
        IDXGIFactory6* factory,
        ID3D12CommandQueue* commandQueue,
        HWND hwnd,
        uint32_t width,
        uint32_t height
    );

    void Shutdown();

    // バッファ操作
    void Present(bool vsync = true);
    uint32_t GetCurrentBackBufferIndex() const;
    ID3D12Resource* GetCurrentBackBuffer() const;
    ID3D12Resource* GetBackBuffer(uint32_t index) const;

    // リサイズ
    void Resize(uint32_t width, uint32_t height);

    // アクセサ
    IDXGISwapChain4* GetDXGISwapChain() const { return swapChain.Get(); }
    uint32_t GetWidth() const { return width; }
    uint32_t GetHeight() const { return height; }
    DXGI_FORMAT GetFormat() const { return format; }
```

```
private:
    ComPtr<IDXGISwapChain4> swapChain;
    std::vector<ComPtr<ID3D12Resource>> backBuffers;

    uint32_t width = 0;
    uint32_t height = 0;
    DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM;

    void CreateBackBuffers();
};

} // namespace Athena
```

src/Athena/Core/SwapChain.cpp

```
#include "Athena/Core/SwapChain.h"
#include "Athena/Utils/Logger.h"
#include <stdexcept>

namespace Athena {

SwapChain::SwapChain() = default;
SwapChain::~SwapChain() = default;

void SwapChain::Initialize(
    IDXGIFactory6* factory,
    ID3D12CommandQueue* commandQueue,
    HWND hwnd,
    uint32_t width,
    uint32_t height) {

    this->width = width;
    this->height = height;

    // スワップチェーン作成
    DXGI_SWAP_CHAIN_DESC1 swapChainDesc = {};
    swapChainDesc.Width = width;
    swapChainDesc.Height = height;
    swapChainDesc.Format = format;
    swapChainDesc.Stereo = FALSE;
    swapChainDesc.SampleDesc.Count = 1;
    swapChainDesc.SampleDesc.Quality = 0;
    swapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
    swapChainDesc.BufferCount = BufferCount;
    swapChainDesc.Scaling = DXGI_SCALING_STRETCH;
    swapChainDesc.SwapEffect = DXGI_SWAP_EFFECT_FLIP_DISCARD;
```

```
    swapChainDesc.AlphaMode = DXGI_ALPHA_MODE_UNSPECIFIED;
    swapChainDesc.Flags = DXGI_SWAP_CHAIN_FLAG_ALLOW_TEARING;

    ComPtr<IDXGISwapChain1> swapChain1;
    HRESULT hr = factory->CreateSwapChainForHwnd(
        commandQueue,
        hwnd,
        &swapChainDesc,
        nullptr,
        nullptr,
        &swapChain1
    );

    if (FAILED(hr)) {
        throw std::runtime_error("Failed to create swap chain");
    }

    // IDXGISwapChain4にキャスト
    swapChain1.As(&swapChain);

    // Alt+Enterの無効化
    factory->MakeWindowAssociation(hwnd, DXGI_MWA_NO_ALT_ENTER);

    // バックバッファ作成
    CreateBackBuffers();

    Logger::Info("SwapChain initialized (%ux%u)", width, height);
}

void SwapChain::Shutdown() {
    backBuffers.clear();
    swapChain.Reset();
}

void SwapChain::Present(bool vsync) {
    UINT syncInterval = vsync ? 1 : 0;
    UINT flags = vsync ? 0 : DXGI_PRESENT_ALLOW_TEARING;

    swapChain->Present(syncInterval, flags);
}

uint32_t SwapChain::GetCurrentBackBufferIndex() const {
    return swapChain->GetCurrentBackBufferIndex();
}

ID3D12Resource* SwapChain::GetCurrentBackBuffer() const {
    return backBuffers[GetCurrentBackBufferIndex()].Get();
}
```

```
ID3D12Resource* SwapChain::GetBackBuffer(uint32_t index) const {
    return backBuffers[index].Get();
}

void SwapChain::Resize(uint32_t width, uint32_t height) {
    if (this->width == width && this->height == height) {
        return;
    }

    this->width = width;
    this->height = height;

    // バックバッファ解放
    backBuffers.clear();

    // リサイズ
    HRESULT hr = swapChain->ResizeBuffers(
        BufferCount,
        width,
        height,
        format,
        DXGI_SWAP_CHAIN_FLAG_ALLOW_TEARING
    );

    if (FAILED(hr)) {
        throw std::runtime_error("Failed to resize swap chain");
    }

    // バックバッファ再作成
    CreateBackBuffers();

    Logger::Info("SwapChain resized to %ux%u", width, height);
}

void SwapChain::CreateBackBuffers() {
    backBuffers.resize(BufferCount);

    for (uint32_t i = 0; i < BufferCount; ++i) {
        HRESULT hr = swapChain->GetBuffer(i, IID_PPV_ARGS(&backBuffers[i]));
        if (FAILED(hr)) {
            throw std::runtime_error("Failed to get back buffer");
        }
    }
}

} // namespace Athena
```

4. DescriptorHeap (デスクリプタヒープ)

include/Athena/Core/DescriptorHeap.h

```
#pragma once
#include <d3d12.h>
#include <wrl/client.h>
#include <cstdint>
#include <vector>

namespace Athena {

using Microsoft::WRL::ComPtr;

struct DescriptorHandle {
    D3D12_CPU_DESCRIPTOR_HANDLE cpu;
    D3D12_GPU_DESCRIPTOR_HANDLE gpu;
    uint32_t index = 0;

    bool IsValid() const { return cpu.ptr != 0; }
    bool IsShaderVisible() const { return gpu.ptr != 0; }
};

class DescriptorHeap {
public:
    DescriptorHeap();
    ~DescriptorHeap();

    void Initialize(
        ID3D12Device* device,
        D3D12_DESCRIPTOR_HEAP_TYPE type,
        uint32_t capacity,
        bool shaderVisible = false
    );

    void Shutdown();

    // デスクリプタ割り当て
    DescriptorHandle Allocate();
    void Free(const DescriptorHandle& handle);

    // アクセサ
    ID3D12DescriptorHeap* GetD3D12DescriptorHeap() const { return heap.Get(); }
    uint32_t GetDescriptorSize() const { return descriptorSize; }

private:
    ComPtr<ID3D12DescriptorHeap> heap;
    D3D12_DESCRIPTOR_HEAP_TYPE type;
```

```
    uint32_t capacity = 0;
    uint32_t descriptorSize = 0;
    bool shaderVisible = false;

    D3D12_CPU_DESCRIPTOR_HANDLE cpuStart = {};
    D3D12_GPU_DESCRIPTOR_HANDLE gpuStart = {};

    std::vector<bool> allocated;
    uint32_t nextFreeIndex = 0;
};

} // namespace Athena
```

src/Athena/Core/DescriptorHeap.cpp

```
#include "Athena/Core/DescriptorHeap.h"
#include "Athena/Utils/Logger.h"
#include <stdexcept>

namespace Athena {

DescriptorHeap::DescriptorHeap() = default;
DescriptorHeap::~DescriptorHeap() = default;

void DescriptorHeap::Initialize(
    ID3D12Device* device,
    D3D12_DESCRIPTOR_HEAP_TYPE type,
    uint32_t capacity,
    bool shaderVisible) {

    this->type = type;
    this->capacity = capacity;
    this->shaderVisible = shaderVisible;

    // デスクリプタヒープ作成
    D3D12_DESCRIPTOR_HEAP_DESC heapDesc = {};
    heapDesc.Type = type;
    heapDesc.NumDescriptors = capacity;
    heapDesc.Flags = shaderVisible ?
        D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE :
        D3D12_DESCRIPTOR_HEAP_FLAG_NONE;
    heapDesc.NodeMask = 0;

    HRESULT hr = device->CreateDescriptorHeap(&heapDesc, IID_PPV_ARGS(&heap));
    if (FAILED(hr)) {
        throw std::runtime_error("Failed to create descriptor heap");
    }
```

```
    // デスクリプタサイズ取得
    descriptorSize = device->GetDescriptorHandleIncrementSize(type);

    // ハンドル取得
    cpuStart = heap->GetCPUDescriptorHandleForHeapStart();
    if (shaderVisible) {
        gpuStart = heap->GetGPUDescriptorHandleForHeapStart();
    }

    // 割り当て管理初期化
    allocated.resize(capacity, false);

    Logger::Info("DescriptorHeap initialized (capacity: %u)", capacity);
}

void DescriptorHeap::Shutdown() {
    allocated.clear();
    heap.Reset();
}

DescriptorHandle DescriptorHeap::Allocate() {
    // フリーなインデックスを検索
    for (uint32_t i = nextFreeIndex; i < capacity; ++i) {
        if (!allocated[i]) {
            allocated[i] = true;
            nextFreeIndex = i + 1;

            DescriptorHandle handle;
            handle.index = i;
            handle.cpu.ptr = cpuStart.ptr + i * descriptorSize;
            if (shaderVisible) {
                handle.gpu.ptr = gpuStart.ptr + i * descriptorSize;
            }

            return handle;
        }
    }

    throw std::runtime_error("Descriptor heap is full");
}

void DescriptorHeap::Free(const DescriptorHandle& handle) {
    if (handle.index < capacity) {
        allocated[handle.index] = false;
        if (handle.index < nextFreeIndex) {
            nextFreeIndex = handle.index;
        }
```

```
    }
}

} // namespace Athena
```

レンダーグラフシステム

RenderPass (レンダーパス基底クラス)

include/Athena/RenderGraph/RenderPass.h

```
#pragma once
#include <string>
#include <vector>

namespace Athena {

class RenderGraphBuilder;
class RenderContext;

class RenderPass {
public:
    virtual ~RenderPass() = default;

    // パスのセットアップ(リソース宣言)
    virtual void Setup(RenderGraphBuilder& builder) = 0;

    // 実行
    virtual void Execute(RenderContext& context) = 0;

    // 入力リソース宣言
    virtual std::vector<std::string> GetInputs() const { return {}; }

    // 出力リソース宣言
    virtual std::vector<std::string> GetOutputs() const { return {}; }

    // パス名
    virtual const char* GetName() const = 0;

    // 有効/無効
    void SetEnabled(bool enabled) { isEnabled = enabled; }
    bool IsEnabled() const { return isEnabled; }

    // 実行時間取得
    float GetLastExecutionTime() const { return lastExecutionTime; }
```

```
protected:
    bool isEnabled = true;
    float lastExecutionTime = 0.0f;
};

} // namespace Athena
```

RenderGraph (レンダーグラフ本体)

include/Athena/RenderGraph/RenderGraph.h

```
#pragma once
#include "Athena/RenderGraph/RenderPass.h"
#include "Athena/RenderGraph/ResourceHandle.h"
#include <memory>
#include <vector>
#include <unordered_map>
#include <string>

namespace Athena {

class RenderContext;

class RenderGraph {
public:
    RenderGraph();
    ~RenderGraph();

    // パス追加
    template<typename PassType, typename... Args>
    PassType* AddPass(Args&&... args) {
        auto pass = std::make_unique<PassType>(std::forward<Args>(args)...);
        PassType* ptr = pass.get();
        passes.push_back(std::move(pass));
        return ptr;
    }

    // 動的パス追加
    void AddPassDynamic(std::unique_ptr<RenderPass> pass);

    // グラフコンパイル
    void Compile();

    // 実行
    void Execute(RenderContext& context);
```

```
    // リソース取得
    ResourceHandle GetResource(const std::string& name);

    // パス取得
    const std::vector<std::unique_ptr<RenderPass>>& GetPasses() const { return passes; }
    size_t GetPassCount() const { return passes.size(); }

    // グラフクリア
    void Clear();

private:
    std::vector<std::unique_ptr<RenderPass>> passes;
    std::unordered_map<std::string, ResourceHandle> resources;
    bool compiled = false;

    // 依存関係解決
    void ResolveDependencies();

    // リソースライフタイム解析
    void AnalyzeResourceLifetime();

    // メモリ最適化
    void OptimizeResourceAllocation();

    // トポロジカルソート
    void TopologicalSort();
};

} // namespace Athena
```

src/Athena/RenderGraph/RenderGraph.cpp

```
#include "Athena/RenderGraph/RenderGraph.h"
#include "Athena/Utils/Logger.h"
#include <algorithm>
#include <unordered_set>

namespace Athena {

RenderGraph::RenderGraph() = default;
RenderGraph::~RenderGraph() = default;

void RenderGraph::AddPassDynamic(std::unique_ptr<RenderPass> pass) {
    passes.push_back(std::move(pass));
}

void RenderGraph::Compile() {
```

```
    Logger::Info("Compiling RenderGraph...");

    // 1. 依存関係解決
    ResolveDependencies();

    // 2. トポロジカルソート
    TopologicalSort();

    // 3. リソースライフタイム解析
    AnalyzeResourceLifetime();

    // 4. メモリ最適化
    OptimizeResourceAllocation();

    compiled = true;
    Logger::Info("RenderGraph compiled successfully (%zu passes)", passes.size());
}

void RenderGraph::Execute(RenderContext& context) {
    if (!compiled) {
        Logger::Warning("RenderGraph not compiled, compiling now...");
        Compile();
    }

    for (auto& pass : passes) {
        if (pass->IsEnabled()) {
            // TODO: タイマーで実行時間計測
            pass->Execute(context);
        }
    }
}

ResourceHandle RenderGraph::GetResource(const std::string& name) {
    auto it = resources.find(name);
    if (it != resources.end()) {
        return it->second;
    }
    return ResourceHandle();
}

void RenderGraph::Clear() {
    passes.clear();
    resources.clear();
    compiled = false;
}

void RenderGraph::ResolveDependencies() {
    // 各パスの入出力を解析
```

```
    std::unordered_map<std::string, std::vector<RenderPass*>> producers;
    std::unordered_map<std::string, std::vector<RenderPass*>> consumers;

    for (auto& pass : passes) {
        // 出力リソースの登録
        for (const auto& output : pass->GetOutputs()) {
            producers[output].push_back(pass.get());
        }

        // 入力リソースの登録
        for (const auto& input : pass->GetInputs()) {
            consumers[input].push_back(pass.get());
        }
    }

    // 依存関係の検証
    for (const auto& [resource, consumerList] : consumers) {
        if (producers.find(resource) == producers.end()) {
            Logger::Warning("Resource '%s' has no producer", resource.c_str());
        }
    }
}

void RenderGraph::AnalyzeResourceLifetime() {
    // リソースの最初の使用と最後の使用を特定
    // これにより、リソースのエイリアシングが可能に
    // TODO: 実装
}

void RenderGraph::OptimizeResourceAllocation() {
    // メモリエイリアシング
    // 使用期間が重ならないリソースは同じメモリを共有可能
    // TODO: 実装
}

void RenderGraph::TopologicalSort() {
    // 依存関係に基づいてパスをソート
    // 現在は単純な順序を維持(後で改善)
    // TODO: 本格的なトポロジカルソート実装
}

} // namespace Athena
```

PassRegistry (パス登録)

include/Athena/RenderGraph/PassRegistry.h

```
#pragma once
#include "Athena/RenderGraph/RenderPass.h"
#include <memory>
#include <string>
#include <unordered_map>
#include <functional>

namespace Athena {

class PassRegistry {
public:
    using CreateFunc = std::function<std::unique_ptr<RenderPass>()>;

    static void Register(const std::string& name, CreateFunc func);
    static std::unique_ptr<RenderPass> Create(const std::string& name);
    static PassRegistry& GetInstance();

private:
    std::unordered_map<std::string, CreateFunc> registry;
};

// パス登録マクロ
#define ATHENA_REGISTER_PASS(PassClass) \
    namespace { \
        struct PassClass##Registrar { \
            PassClass##Registrar() { \
                PassRegistry::Register(#PassClass, []() { \
                    return std::make_unique<PassClass>(); \
                }); \
            } \
        }; \
        static PassClass##Registrar g_##PassClass##Registrar; \
    }

} // namespace Athena
```

src/Athena/RenderGraph/PassRegistry.cpp

```
#include "Athena/RenderGraph/PassRegistry.h"
#include "Athena/Utils/Logger.h"

namespace Athena {

PassRegistry& PassRegistry::GetInstance() {
    static PassRegistry instance;
    return instance;
}
```

```
void PassRegistry::Register(const std::string& name, CreateFunc func) {
    auto& instance = GetInstance();
    instance.registry[name] = func;
    Logger::Info("Registered pass: %s", name.c_str());
}

std::unique_ptr<RenderPass> PassRegistry::Create(const std::string& name) {
    auto& instance = GetInstance();
    auto it = instance.registry.find(name);
    if (it != instance.registry.end()) {
        return it->second();
    }
    Logger::Error("Pass '%s' not found in registry", name.c_str());
    return nullptr;
}

} // namespace Athena
```

標準レンダーパス

GeometryPass

AthenaViewer/Passes/GeometryPass.h

```
#pragma once
#include <Athena/RenderGraph/RenderPass.h>

namespace Athena {

class Scene;

class GeometryPass : public RenderPass {
public:
    GeometryPass(Scene* scene);

    void Setup(RenderGraphBuilder& builder) override;
    void Execute(RenderContext& context) override;

    std::vector<std::string> GetOutputs() const override;
    const char* GetName() const override { return "GeometryPass"; }

private:
    Scene* scene;
};
```

```
} // namespace Athena
```

AthenaViewer/Passes/GeometryPass.cpp

```
#include "GeometryPass.h"
#include <Athena/Scene/Scene.h>
#include <Athena/Rendering/RenderContext.h>

namespace Athena {

GeometryPass::GeometryPass(Scene* scene) : scene(scene) {}

void GeometryPass::Setup(RenderGraphBuilder& builder) {
    // TODO: リソース宣言
}

void GeometryPass::Execute(RenderContext& context) {
    // TODO: ジオメトリレンダリング
}

std::vector<std::string> GeometryPass::GetOutputs() const {
    return {"GBuffer_Albedo", "GBuffer_Normal", "GBuffer_Depth"};
}

ATHENA_REGISTER_PASS(GeometryPass);

} // namespace Athena
```

LightingPass

AthenaViewer/Passes/LightingPass.h

```
#pragma once
#include <Athena/RenderGraph/RenderPass.h>

namespace Athena {

class LightingPass : public RenderPass {
public:
    void Setup(RenderGraphBuilder& builder) override;
    void Execute(RenderContext& context) override;

    std::vector<std::string> GetInputs() const override;
    std::vector<std::string> GetOutputs() const override;
    const char* GetName() const override { return "LightingPass"; }
```

```
};

} // namespace Athena
```

AthenaViewer/Passes/LightingPass.cpp

```
#include "LightingPass.h"
#include <Athena/Rendering/RenderContext.h>

namespace Athena {

void LightingPass::Setup(RenderGraphBuilder& builder) {
    // TODO: リソース宣言
}

void LightingPass::Execute(RenderContext& context) {
    // TODO: ライティング
}

std::vector<std::string> LightingPass::GetInputs() const {
    return {"GBuffer_Albedo", "GBuffer_Normal", "GBuffer_Depth"};
}

std::vector<std::string> LightingPass::GetOutputs() const {
    return {"LitScene"};
}

ATHENA_REGISTER_PASS(LightingPass);

} // namespace Athena
```

ビューアーアプリケーション

ViewerApp

AthenaViewer/ViewerApp.h

```
#pragma once
#include <Athena/Core/Device.h>
#include <Athena/Core/CommandQueue.h>
#include <Athena/Core/SwapChain.h>
#include <Athena/RenderGraph/RenderGraph.h>
#include <Athena/Scene/Scene.h>
#include <Athena/Scene/Camera.h>
```

```
#include "UI/ImGuiManager.h"

namespace Athena {

class ViewerApp {
public:
    ViewerApp();
    ~ViewerApp();

    void Initialize();
    void Run();
    void Shutdown();

private:
    void ProcessInput();
    void Update(float deltaTime);
    void Render();
    void RenderUI();

    // コア
    Device device;
    CommandQueue commandQueue;
    SwapChain swapChain;

    // レンダリング
    RenderGraph renderGraph;
    Scene scene;
    Camera camera;

    // UI
    ImGuiManager imguiManager;

    // ウィンドウ
    HWND hwnd = nullptr;
    bool running = true;

    // 状態
    bool showDebugPanel = true;
    bool showPerformance = true;

    void CreateWindow();
    void DestroyWindow();
    void SetupRenderGraph();

    static LRESULT CALLBACK WindowProc(HWND hwnd, UINT msg, WPARAM wParam,
LPARAM lParam);
};
```

```
} // namespace Athena
```

AthenaViewer/ViewerApp.cpp

```
#include "ViewerApp.h"
#include "UI/DebugPanel.h"
#include "UI/PerformancePanel.h"
#include <Athena/Utils/Logger.h>

namespace Athena {

ViewerApp::ViewerApp() = default;
ViewerApp::~ViewerApp() = default;

void ViewerApp::Initialize() {
    Logger::Info("Initializing Athena Viewer...");

    // ウィンドウ作成
    CreateWindow();

    // デバイス初期化
    device.Initialize(true);

    // コマンドキュー初期化
    commandQueue.Initialize(device.GetD3D12Device(),
D3D12_COMMAND_LIST_TYPE_DIRECT);

    // スワップチェーン初期化
    swapChain.Initialize(
        device.GetDXGIFactory(),
        commandQueue.GetD3D12CommandQueue(),
        hwnd,
        1920, 1080
    );

    // ImGui初期化
    imguiManager.Initialize(&device, hwnd);

    // レンダーグラフセットアップ
    SetupRenderGraph();

    Logger::Info("Athena Viewer initialized successfully");
}

void ViewerApp::Run() {
    MSG msg = {};
```

```
    while (running) {
        // メッセージ処理
        while (PeekMessage(&msg, nullptr, 0, 0, PM_REMOVE)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);

            if (msg.message == WM_QUIT) {
                running = false;
            }
        }

        if (!running) break;

        // 更新
        ProcessInput();
        Update(0.016f); // 仮の deltaTime

        // レンダリング
        Render();
    }
}

void ViewerApp::Shutdown() {
    Logger::Info("Shutting down Athena Viewer...");

    // GPU待機
    commandQueue.WaitForGPU();

    // 終了処理
    renderGraph.Clear();
    imguiManager.Shutdown();
    swapChain.Shutdown();
    commandQueue.Shutdown();
    device.Shutdown();

    DestroyWindow();
}

void ViewerApp::ProcessInput() {
    // TODO: 入力処理
}

void ViewerApp::Update(float deltaTime) {
    // TODO: カメラ更新など
}

void ViewerApp::Render() {
    // TODO: レンダリング実装
```

```
    // UI描画
    RenderUI();

    // プレゼント
    swapChain.Present(true);
}

void ViewerApp::RenderUI() {
    imguiManager.NewFrame();

    // デバッグパネル
    if (showDebugPanel) {
        DebugPanel::Render(renderGraph, scene, showDebugPanel);
    }

    // パフォーマンスパネル
    if (showPerformance) {
        PerformancePanel::Render(showPerformance);
    }

    imguiManager.Render(nullptr); // TODO: CommandList渡す
}

void ViewerApp::CreateWindow() {
    // TODO: Win32ウィンドウ作成
}

void ViewerApp::DestroyWindow() {
    if (hwnd) {
        DestroyWindow(hwnd);
        hwnd = nullptr;
    }
}

void ViewerApp::SetupRenderGraph() {
    // TODO: レンダーグラフ構築
    renderGraph.Compile();
}

LRESULT CALLBACK ViewerApp::WindowProc(HWND hwnd, UINT msg, WPARAM
wParam, LPARAM lParam) {
    // TODO: ウィンドウプロシージャ
    return DefWindowProc(hwnd, msg, wParam, lParam);
}

} // namespace Athena
```

main.cpp

AthenaViewer/main.cpp

```
#include "ViewerApp.h"
#include <Athena/Utils/Logger.h>
#include <exception>

int WINAPI WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow) {

    try {
        Athena::Logger::Initialize();

        Athena::ViewerApp app;
        app.Initialize();
        app.Run();
        app.Shutdown();

        Athena::Logger::Shutdown();
        return 0;
    }
    catch (const std::exception& e) {
        Athena::Logger::Error("Fatal error: %s", e.what());
        MessageBoxA(nullptr, e.what(), "Error", MB_OK | MB_ICONERROR);
        return 1;
    }
}
```

研究機能の追加方法

新しいレンダーパスの作成

```
// AthenaResearch/NeuralRendering/NeuralGIPass.h
#pragma once
#include <Athena/RenderGraph/RenderPass.h>

namespace Athena::Research {

class NeuralGIPass : public RenderPass {
public:
    NeuralGIPass();
```

```
    void Setup(RenderGraphBuilder& builder) override;
    void Execute(RenderContext& context) override;

    std::vector<std::string> GetInputs() const override;
    std::vector<std::string> GetOutputs() const override;
    const char* GetName() const override { return "NeuralGIPass"; }

private:
    void LoadNetwork(const std::string& path);
    void RunInference(RenderContext& context);
};

ATHENA_REGISTER_PASS(NeuralGIPass);

} // namespace Athena::Research
```

設定ファイルでの使用

```
// configs/neural_gi.json
{
    "name": "Neural GI Experiment",
    "description": "Real-time GI using neural networks",
    "passes": [
        {
            "type": "GeometryPass",
            "enabled": true
        },
        {
            "type": "NeuralGIPass",
            "enabled": true,
            "config": {
                "modelPath": "models/gi_network.onnx"
            }
        },
        {
            "type": "LightingPass",
            "enabled": true
        }
    ]
}
```

実験・ベンチマーク

ベンチマークの実行

```
// experiments/benchmark_runner.cpp
#include <Athena/Athena.h>
#include <vector>
#include <string>

int main(int argc, char** argv) {
    using namespace Athena;

    // 初期化
    Device device;
    device.Initialize(false); // デバッグレイヤーOFF

    // ベンチマーク対象の設定
    std::vector<std::string> configs = {
        "configs/standard.json",
        "configs/neural_gi.json",
        "configs/gpu_driven.json"
    };

    // 各設定でベンチマーク
    for (const auto& config : configs) {
        Logger::Info("Benchmarking: %s", config.c_str());

        // TODO: ベンチマーク実行
    }

    return 0;
}
```

開発フロー

フェーズ**1:** 基盤開発

目標: 三角形を表示

実装内容:
1. Device初期化
2. CommandQueue作成
3. SwapChain作成
4. 基本的なコマンドリスト実行
5. 単純な頂点・ピクセルシェーダー
6. 三角形描画

確認事項:
✓ ウィンドウに三角形が表示される

✓ PIXでキャプチャできる
✓ メモリリークがない

フェーズ2: リソース管理

目標: テクスチャ付きキューブ

実装内容:
1. Buffer抽象化
2. Texture抽象化
3. DescriptorHeap管理
4. ResourceManager
5. 頂点・インデックスバッファ
6. 定数バッファ(MVP行列)
7. テクスチャサンプリング

確認事項:
✓ キューブが回転する
✓ テクスチャが正しく表示される
✓ 深度テストが動作

フェーズ3: レンダーグラフ

目標: 柔軟なパイプライン構成

実装内容:
1. RenderPass基底クラス
2. RenderGraph本体
3. 依存関係解決
4. リソースライフタイム管理
5. PassRegistry
6. 標準パス実装

確認事項:
✓ パスの動的追加・削除
✓ 依存関係の自動解決
✓ 設定ファイルから構築

フェーズ4: シーン管理

目標: 3Dモデル表示

実装内容:
1. Mesh / Material
2. Camera
3. Light

4. Scene管理
5. Assimp統合(モデルローダー)

確認事項:
✓ OBJ/FBXモデルの読み込み
✓ カメラ操作(FPS風)
✓ 複数メッシュの描画

フェーズ5: ビューアー

目標: 使いやすい実験環境

実装内容:
1. ImGui統合
2. DebugPanel
3. PerformancePanel
4. 入力処理
5. スクリーンショット機能

確認事項:
✓ UIで設定変更
✓ リアルタイムプロファイリング
✓ 直感的な操作

フェーズ6: 研究機能実装

目標: 新規レンダリング技術の実装

実装内容:
1. 研究用パス作成
2. シェーダー開発
3. パラメータ調整
4. ベンチマーク実施

確認事項:
✓ 基盤を変更せずに実装
✓ 既存手法との比較
✓ 論文用データ収集

論文執筆への応用

実装章の記述例

\section{Implementation}

We implemented our method in \textit{Athena Renderer}, a DirectX 12-based rendering framework designed for rapid prototyping of rendering techniques. Athena's modular render graph architecture enables researchers to compose and evaluate different rendering strategies without modifying core systems.

\subsection{System Architecture}

Athena consists of four layers (Figure~\ref{fig:architecture}):
\begin{enumerate}
    \item \textbf{DirectX 12 Abstraction}: Device management, resource allocation, and command execution
    \item \textbf{Render Graph}: Automatic dependency resolution and resource lifetime management
    \item \textbf{Standard Passes}: Geometry, lighting, and post-processing
    \item \textbf{Research Modules}: Novel techniques as pluggable passes
\end{enumerate}

Our implementation (\texttt{NeuralGIPass}) integrates seamlessly into the pipeline, requiring no changes to the core framework or other passes.

\subsection{Performance Profiling}

Athena includes integrated GPU profiling using DirectX 12 timestamp queries, enabling precise per-pass timing measurements. All experiments were conducted on an NVIDIA RTX 4090 GPU with 24GB VRAM running Windows 11.

実験結果の提示

\begin{table}[h]
\centering
\caption{Performance Comparison on Sponza Scene (1920×1080)}
\begin{tabular}{lccc}
\toprule
Method & Frame Time (ms) & GPU Time (ms) & PSNR (dB) \\
\midrule
Standard Raster & 8.2 ± 0.3 & 7.1 ± 0.2 & 28.3 \\
SSGI & 15.7 ± 0.5 & 14.2 ± 0.4 & 31.8 \\
Neural GI (Ours) & 11.3 ± 0.4 & 9.8 ± 0.3 & 33.2 \\
\bottomrule
\end{tabular}
\label{tab:performance}
\end{table}

Our method achieves 38\% faster rendering than SSGI while maintaining 4.4\% higher image quality (PSNR). Figure~\ref{fig:visual_comparison} shows qualitative comparisons.

トラブルシューティング

よくある問題

1. ビルドエラー**: "d3d12.h** が見つかりません**"**
原因: Windows SDK がインストールされていない

解決策:
1. Visual Studio Installer起動
2. 変更 → 個別のコンポーネント
3. "Windows 10 SDK" または "Windows 11 SDK" にチェック
4. 変更を適用

2. 実行時エラー**: "D3D12CreateDevice failed"**
原因: DirectX 12に対応していないGPU

解決策:
1. dxdiag で DirectX バージョン確認
2. GPU ドライバーを最新に更新
3. 対応GPUへの交換

3. リンクエラー**: "unresolved external symbol"**
原因: ライブラリのリンク忘れ

解決策:
プロジェクトプロパティ → リンカー → 入力
→ 追加の依存ファイルに以下を追加:
  d3d12.lib
  dxgi.lib
  d3dcompiler.lib

4. ImGuiが表示されない
原因: バックエンドの初期化不足

解決策:
1. imgui_impl_win32.cpp をプロジェクトに追加
2. imgui_impl_dx12.cpp をプロジェクトに追加
3. 初期化順序を確認:
   ImGui_ImplWin32_Init()
   ImGui_ImplDX12_Init()

5. シェーダーコンパイルエラー
原因: シェーダーモデルの不一致

解決策:
シェーダーファイルのプロパティ
→ シェーダーモデル: 6.0 以上に設定

デバッグツール

1. PIX for Windows
   - GPU デバッグ
   - フレームキャプチャ
   - パフォーマンス分析

2. RenderDoc
   - クロスプラットフォーム
   - 詳細なフレーム解析

3. Visual Studio Graphics Debugger
   - 統合環境
   - ブレークポイント設定可能

参考資料

公式ドキュメント

- Microsoft DirectX 12 Programming Guide
- DirectX Graphics Samples

書籍

- "Introduction to 3D Game Programming with DirectX 12" - Frank Luna
- "Real-Time Rendering, 4th Edition" - Tomas Akenine-Möller et al.
- "Physically Based Rendering" - Matt Pharr et al.

参考実装

- Falcor - NVIDIA
- MiniEngine - Microsoft
- AMD Cauldron

ライセンス

まとめ

Athena Rendererは、グラフィックス研究のための柔軟で拡張性の高いフレームワークです。このドキュメントに従って開発を進めることで:

✅ 堅牢な基盤: DirectX 12の深い理解 ✅ 研究の自由: 基盤を変えずに新機能追加 ✅ 論文執筆: 実験データの自動収集 ✅ 技術力証明: ポートフォリオ作品として

知恵の女神アテナのように、賢明に技術を探求し、素晴らしい研究成果を生み出してください。

Athena Renderer Development Team Version 1.0 - 2025