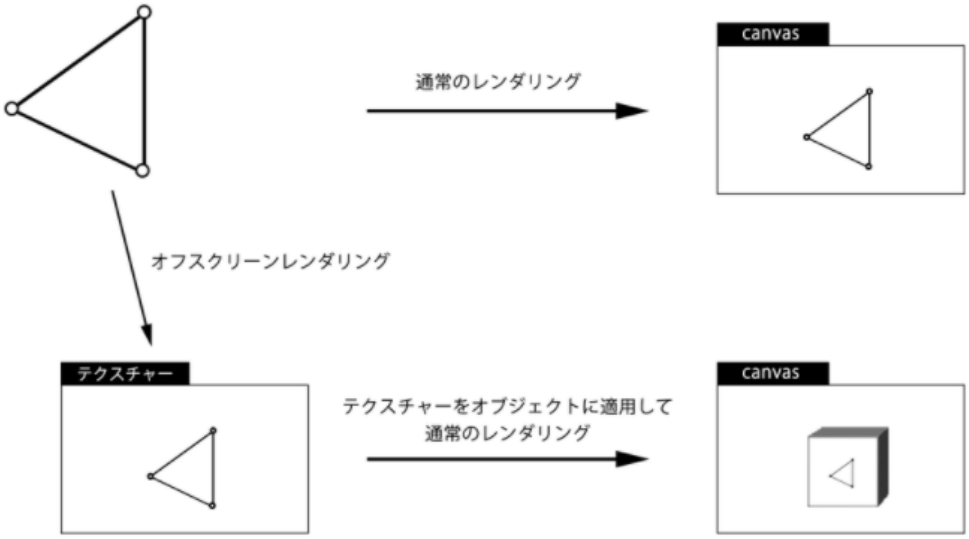
描画エンジン開発 EX

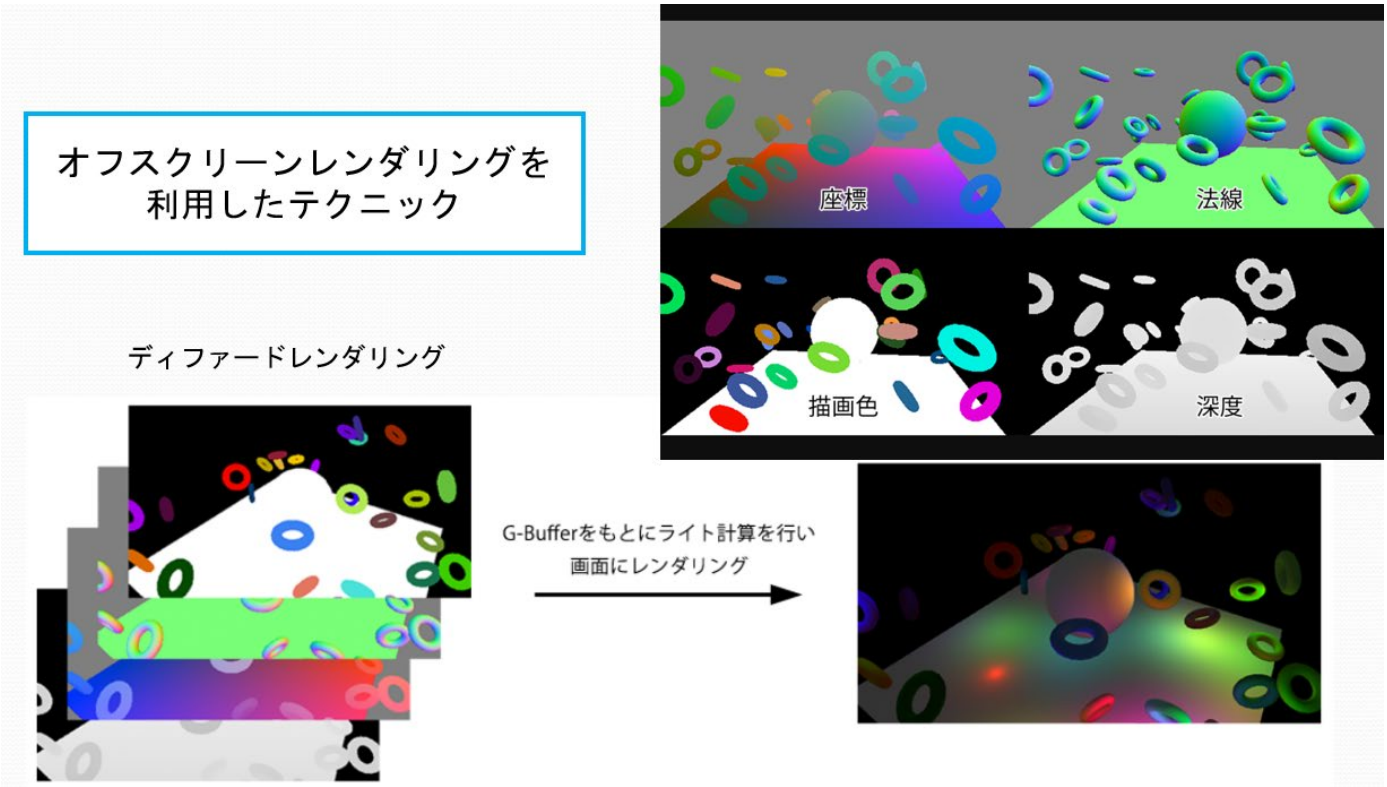
○概要

オフスクリーンレンダリングを行いポストエフェクト処理を実装する。

○オフスクリーンレンダリング

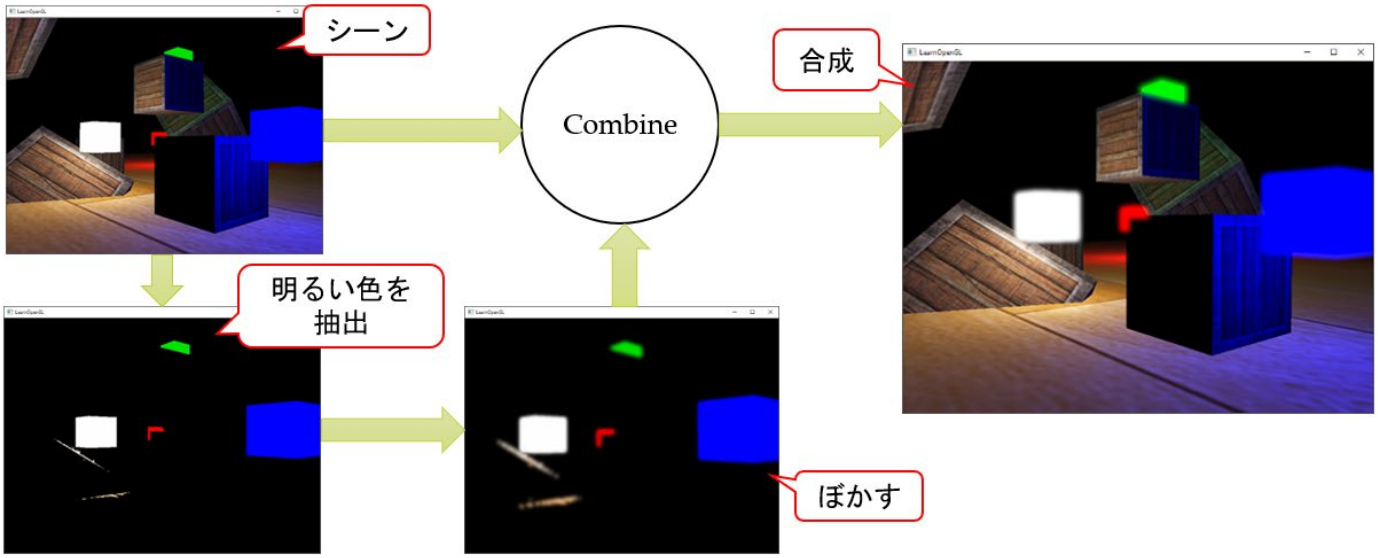
オフスクリーンレンダリングとはバックバッファに直接シーンを描画するのではなく、別のフレームバッファにシーンを描画することです。
最終的にバックバッファに描画する際にオフスクリーンレンダリングしたテクスチャを利用して最終的な画面を描画します。





○ポストエフェクト

画面のレンダリングが終わった後に、画面全体に演出効果をかける技術です。
今回はブルームという光が溢れるような演出を実装します。



○フレームバッファの作成

オフスクリーンレンダリングをするためのフレームバッファを作成します。
また、レンダリングした画像をテクスチャとして利用できるようにシェーダーリソースビューを作成し、取得できるようにします。

FrameBuffer.h

```
---省略---

class FrameBuffer
{
public:
    ---省略---
    FrameBuffer(ID3D11Device* device, UINT width, UINT height);

    // カラーマップ取得
    ID3D11ShaderResourceView* GetColorMap() const { return colorMap.Get(); }

    ---省略---

private:
    ---省略---
    Microsoft::WRL::ComPtr<ID3D11ShaderResourceView>      colorMap;
};
```

FrameBuffer.cpp

```
---省略---
```

```
// コンストラクタ
FrameBuffer::FrameBuffer(ID3D11Device* device, UINT width, UINT height)
{
    HRESULT hr = S_OK;

    // レンダーターゲット
    {
        // テクスチャ生成
        Microsoft::WRL::ComPtr<ID3D11Texture2D> renderTargetBuffer;
        D3D11_TEXTURE2D_DESC texture2dDesc{};
        texture2dDesc.Width = width;
        texture2dDesc.Height = height;
        texture2dDesc.MipLevels = 1;
        texture2dDesc.ArraySize = 1;
        texture2dDesc.Format = DXGI_FORMAT_R16G16B16A16_FLOAT;
        texture2dDesc.SampleDesc.Count = 1;
        texture2dDesc.SampleDesc.Quality = 0;
        texture2dDesc.Usage = D3D11_USAGE_DEFAULT;
        texture2dDesc.BindFlags = D3D11_BIND_RENDER_TARGET | D3D11_BIND_SHADER_RESOURCE;
        texture2dDesc.CPUAccessFlags = 0;
        texture2dDesc.MiscFlags = 0;
        hr = device->CreateTexture2D(&texture2dDesc, 0, renderTargetBuffer.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

        // レンダーターゲットビュー生成
        D3D11_RENDER_TARGET_VIEW_DESC renderTargetViewDesc{};
        renderTargetViewDesc.Format = texture2dDesc.Format;
        renderTargetViewDesc.ViewDimension = D3D11_RTV_DIMENSION_TEXTURE2D;
        hr = device->CreateRenderTargetView(renderTargetBuffer.Get(), &renderTargetViewDesc,
                                                                    renderTargetView.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

        // シェーダーリソースビュー生成
        D3D11_SHADER_RESOURCE_VIEW_DESC shaderResourceViewDesc{};
        shaderResourceViewDesc.Format = texture2dDesc.Format;
        shaderResourceViewDesc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
        shaderResourceViewDesc.Texture2D.MipLevels = 1;
        hr = device->CreateShaderResourceView(renderTargetBuffer.Get(), &shaderResourceViewDesc,
                                                                        colorMap.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }

    // デプスステンシル
    {
        // テクスチャ生成
        Microsoft::WRL::ComPtr<ID3D11Texture2D> depthStencilBuffer;
        D3D11_TEXTURE2D_DESC texture2dDesc{};
        texture2dDesc.Width = width;
        texture2dDesc.Height = height;
        texture2dDesc.MipLevels = 1;
        texture2dDesc.ArraySize = 1;
        texture2dDesc.Format = DXGI_FORMAT_R24G8_TYPELESS;
        texture2dDesc.SampleDesc.Count = 1;
        texture2dDesc.SampleDesc.Quality = 0;
        texture2dDesc.Usage = D3D11_USAGE_DEFAULT;
        texture2dDesc.BindFlags = D3D11_BIND_DEPTH_STENCIL;
```

ブルームは光が溢れる演出をするため、
色の値が 1.0 を超える可能性があるので、
HDR フォーマットにする

シェーダーリソースビューとして
扱えるようにテクスチャを生成する

```
        texture2dDesc.CPUAccessFlags = 0;
        texture2dDesc.MiscFlags = 0;
        hr = device->CreateTexture2D(&texture2dDesc, 0, depthStencilBuffer.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

        // デプスステンシルビュー生成
        D3D11_DEPTH_STENCIL_VIEW_DESC depthStencilViewDesc{};
        depthStencilViewDesc.Format = DXGI_FORMAT_D24_UNORM_S8_UINT;
        depthStencilViewDesc.ViewDimension = D3D11_DSV_DIMENSION_TEXTURE2D;
        depthStencilViewDesc.Flags = 0;
        hr = device->CreateDepthStencilView(depthStencilBuffer.Get(), &depthStencilViewDesc,
                                                                    depthStencilView.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }

    // ビューポート
    {
        viewport.Width = static_cast<float>(width);
        viewport.Height = static_cast<float>(height);
        viewport.MinDepth = 0.0f;
        viewport.MaxDepth = 1.0f;
        viewport.TopLeftX = 0.0f;
        viewport.TopLeftY = 0.0f;
    }
}
```

Graphics.h

```
---省略---

enum class FrameBufferId
{
    Display,
    Scene,
    Luminance,

    EnumCount
};

// グラフィックス
class Graphics
{
    ---省略---

public:
    ---省略---

    // フレームバッファ取得
    FrameBuffer* GetFrameBuffer() { frameBuffer.get(); }
    FrameBuffer* GetFrameBuffer(FrameBufferId frameBufferId)
    {
        return frameBuffers[static_cast<int>(frameBufferId)].get();
    }

    ---省略---
```

```
private:
    ---省略---

    std::unique_ptr<FrameBuffer>    frameBuffer;
    std::unique_ptr<FrameBuffer>    frameBuffers[static_cast<int>(FrameBufferId::EnumCount)];
};
```

Graphics.cpp

```
---省略---

// 初期化
void Graphics::Initialize(HWND hWnd)
{
    ---省略---

    // フレームバッファ作成
    frameBuffer = std::make_unique<FrameBuffer>(device.Get(), swapchain.Get());
    frameBuffers[static_cast<int>(FrameBufferId::Display)] = std::make_unique<FrameBuffer>(device.Get(),
                                                                                           swapchain.Get());
    frameBuffers[static_cast<int>(FrameBufferId::Scene)] = std::make_unique<FrameBuffer>(device.Get(),
                                                                                         screenWidth, screenHeight);
    frameBuffers[static_cast<int>(FrameBufferId::Luminance)] = std::make_unique<FrameBuffer>(device.Get(),
                                                                                             screenWidth, screenHeight);


    ---省略---
}
```

Framework.cpp

```
// 描画処理
void Framework::Render(float elapsedTime)
{
    ---省略---

    // 画面クリア
    Graphics::Instance().GetFrameBuffer()->Clear(dc, 0, 0, 1, 1);
    Graphics::Instance().GetFrameBuffer(FrameBufferId::Display)->Clear(dc, 0, 0, 1, 1);

    // レンダーターゲット設定
    Graphics::Instance().GetFrameBuffer()->SetRenderTargets(dc);
    Graphics::Instance().GetFrameBuffer(FrameBufferId::Display)->SetRenderTargets(dc);

    ---省略---
}
```

ポストエフェクトテストシーンを作成し、画面全体にスプライトを描画します。

Scene.h

```
---省略---
```

```
// ポストエフェクトテストシーン
class PostEffectTestScene : public Scene
{
public:
    PostEffectTestScene();
    ~PostEffectTestScene() override = default;

    // 描画処理
    void Render(float elapsedTime) override;

private:
    std::unique_ptr<Sprite>        sprite;
};
```

Scene.cpp

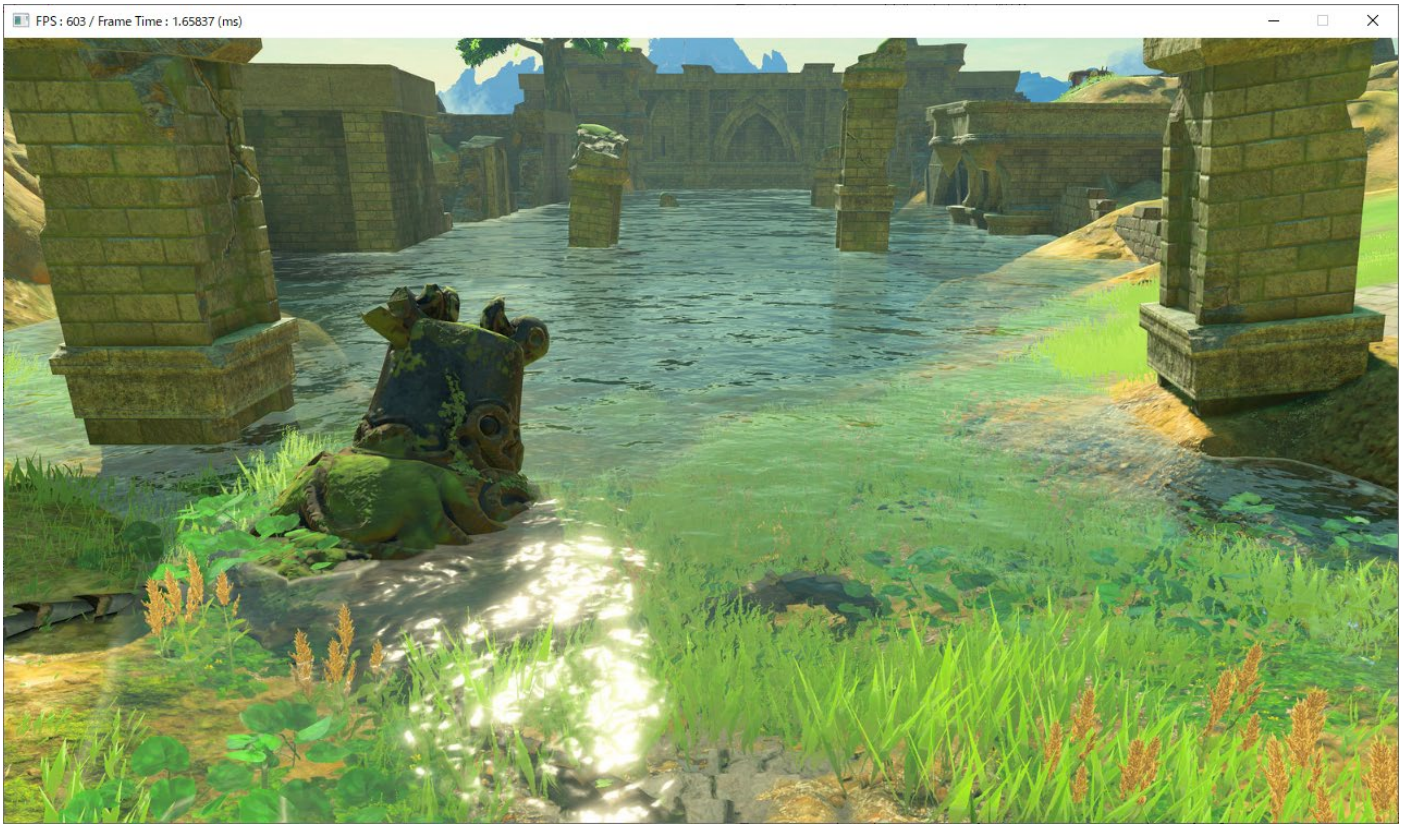
```
---省略---

// コンストラクタ
PostEffectTestScene::PostEffectTestScene()
{
    ID3D11Device* device = Graphics::Instance().GetDevice();

    sprite = std::make_unique<Sprite>(device, "Data/Sprite/screenshot.jpg");
}

// 描画処理
void PostEffectTestScene::Render(float elapsedTime)
{
    ID3D11DeviceContext* dc = Graphics::Instance().GetDeviceContext();

    RenderState* renderState = Graphics::Instance().GetRenderState();
    float screenWidth = Graphics::Instance().GetScreenWidth();
    float screenHeight = Graphics::Instance().GetScreenHeight();

    // ブレンドステート設定
    FLOAT blendFactor[4] = { 1.0f, 1.0f, 1.0f, 1.0f };
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Opaque), blendFactor, 0xFFFFFFFF);

    // 深度ステンシルステート設定
    dc->OMSetDepthStencilState(renderState->GetDepthStencilState(DepthState::NoTestNoWrite), 0);

    // ラスタライザーステート設定
    dc->RSSetState(renderState->GetRasterizerState(RasterizerState::SolidCullNone));

    // サンプラステート設定
    ID3D11SamplerState* samplers[] =
    {
        renderState->GetSamplerState(SamplerState::LinearWrap)
    };
    dc->PSSetSamplers(0, _countof(samplers), samplers);

    // スプライト描画
    sprite->Render(dc, 0, 0, 0, screenWidth, screenHeight, 0, 1, 1, 1, 1);
```

```
}
```

Framework.cpp

```
---省略---

// コンストラクタ
Framework::Framework(HWND hWnd)
	: hWnd(hWnd)
{
	---省略---

	// シーン初期化
	scene = std::make_unique<ModelTestScene>();
	scene = std::make_unique<PostEffectTestScene>();
}
```

実行確認してみましょう。
下図のような画像が表示されていれば OK です。



○画面全体を描画するシェーダーの作成

ポストエフェクトは画面全体に演出効果を与える技術です。
そのための準備として今回は頂点バッファを使用せずに画面全体にポリゴンを描画するシェーダ

ーを実装します。

FullScreenQuad.hlsli と FullScreenQuadVS.hlsl と LuminanceExtractionPS.hlsl を作成しましょう。

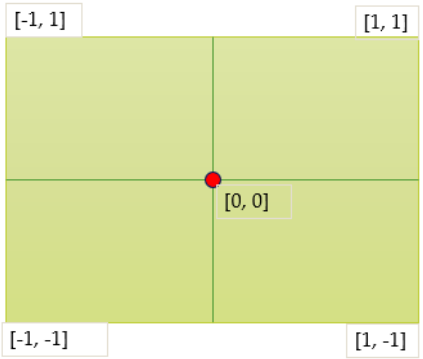
FullScreenQuad.hlsli

```
struct VS_OUT
{
    float4 position : SV_POSITION;
    float2 texcoord : TEXCOORD;
};
```

FullScreenQuadVS.hlsl

```
#include "FullScreenQuad.hlsli"

VS_OUT main(uint vertexId : SV_VERTEXID)
{
    const float2 position[4] =
    {
        { -1, +1 },   // 左上
        { +1, +1 },   // 右上
        { -1, -1 },   // 左下
        { +1, -1 }    // 右下
    };
    const float2 texcoords[4] =
    {
        { 0, 0 },     // 左上
        { 1, 0 },     // 右上
        { 0, 1 },     // 左下
        { 1, 1 } // 右下
    };
    VS_OUT vout;
    vout.position = float4(position[vertexId], 0, 1);
    vout.texcoord = texcoords[vertexId];
    return vout;
}
```

SV_VERTEXID には頂点の番号が送られる。
4 頂点で描画した場合は(0～3)の番号が渡される



NDC 空間の座標

LuminanceExtractionPS は画面の明るい色を抽出するためのシェーダーです。
この実装は後で行うので今はテスト用に画面全体が赤色になるようにしておきます。

LuminanceExtractionPS.hlsl

```
#include "FullScreenQuad.hlsli"

float4 main(VS_OUT pin) : SV_TARGET
{
    return float4(1.0f, 0.0f, 0.0f, 1.0f);
}
```

とりあえず赤色を表示

ポストエフェクトを管理する PostEffect クラスを実装します。
PostEffect.cpp と PostEffect.h を作成しましょう。

PostEffect.h

```
#pragma once

#include <wrl.h>
#include <d3d11.h>
#include "RenderContext.h"

class PostEffect
{
public:
    PostEffect(ID3D11Device* device);

    // 開始処理
    void Begin(const RenderContext& rc);

    // 輝度抽出処理
    void LuminanceExtraction(const RenderContext& rc);

    // 終了処理
    void End(const RenderContext& rc);

private:
    Microsoft::WRL::ComPtr<ID3D11VertexShader>      fullscreenQuadVS;
    Microsoft::WRL::ComPtr<ID3D11PixelShader>       luminanceExtractionPS;
};
```

PostEffect.cpp

```
#include "PostEffect.h"
#include "GpuResourceUtils.h"

PostEffect::PostEffect(ID3D11Device* device)
{
    // フルクリーンクアッド頂点シェーダー読み込み
    GpuResourceUtils::LoadVertexShader(
        device,
        "Data/Shader/FullScreenQuadVS.cso",
        nullptr, 0,
        nullptr,
        fullscreenQuadVS.GetAddressOf());

    // 輝度抽出ピクセルシェーダー読み込み
    GpuResourceUtils::LoadPixelShader(
        device,
        "Data/Shader/LuminanceExtractionPS.cso",
        luminanceExtractionPS.GetAddressOf());
}
```

```
// 開始処理
void PostEffect::Begin(const RenderContext& rc)
{
    ID3D11DeviceContext* dc = rc.deviceContext;
    const RenderState* renderState = rc.renderState;

    // ブレンドステート設定
    FLOAT blendFactor[4] = { 1.0f, 1.0f, 1.0f, 1.0f };
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Opaque), blendFactor, 0xFFFFFFFF);

    // 深度ステンシルステート設定
    dc->OMSetDepthStencilState(renderState->GetDepthStencilState(DepthState::NoTestNoWrite), 0);

    // ラスタライザーステート設定
    dc->RSSetState(renderState->GetRasterizerState(RasterizerState::SolidCullNone));

    // 頂点バッファ設定（使用しない）
    dc->IASetVertexBuffers(0, 0, nullptr, nullptr, nullptr);
    dc->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP);
    dc->IASetInputLayout(nullptr);
}

// 輝度抽出処理
void PostEffect::LuminanceExtraction(const RenderContext& rc)
{
    ID3D11DeviceContext* dc = rc.deviceContext;

    // シェーダー設定
    dc->VSSetShader(fullscreenQuadVS.Get(), 0, 0);
    dc->PSSetShader(luminanceExtractionPS.Get(), 0, 0);

    // 描画
    dc->Draw(4, 0);
}

// 終了処理
void PostEffect::End(const RenderContext& rc)
{
    // 今のところ何もしない
}
```

頂点バッファを使わない

4 頂点で描画をする

Scene.h

```
---省略---
#include "PostEffect.h"

---省略---

// ポストエフェクトテストシーン
class PostEffectTestScene : public Scene
{
    ---省略---

private:
    ---省略---
```

```
    std::unique_ptr<PostEffect>    postEffect;
};
```

PostEffect.cpp

```
---省略---

// コンストラクタ
PostEffectTestScene::PostEffectTestScene()
{
    ---省略---
    postEffect = std::make_unique<PostEffect>(device);
}

// 描画処理
void PostEffectTestScene::Render(float elapsedTime)
{
    ---省略---

    // 描画コンテキスト設定
    RenderContext rc;
    rc.deviceContext = Graphics::Instance().GetDeviceContext();
    rc.renderState = Graphics::Instance().GetRenderState();

    // ポストエフェクト処理開始
    postEffect->Begin(rc);

    // 輝度抽出処理
    postEffect->LuminanceExtraction(rc);

    // 終了処理
    postEffect->End(rc);
}
```

実行確認してみましょう。
下図のように画面全体にポリゴンを描き、赤色に塗られていることが確認できれば OK です。

○輝度を抽出するシェーダーを作成

ブルーム処理は画面の明るい部分を抽出し、明るい部分をぼかすことによって光が溢れる演出効果を与えます。
まずは画面の明るい部分を抽出するシェーダーを実装します。
PostEffect.hlsli を作成しましょう。

PostEffect.hlsli

```
cbuffer CbPostEffect : register(b0)
{
    float    luminanceExtractionLowerEdge;
    float    luminanceExtractionHigherEdge;
};
```

輝度の範囲を指定

LuminanceExtractionPS.hlsl

```
#include "FullScreenQuad.hlsli"
#include "PostEffect.hlsli"

Texture2D    colorMap     : register(t0);
SamplerState linearSampler: register(s0);

float4 main(VS_OUT pin) : SV_TARGET
{
```

```
    float4 color = colorMap.Sample(linearSampler, pin.texcoord);

    color.rgb *= smoothstep(luminanceExtractionLowerEdge, luminanceExtractionHigherEdge, dot(color.rgb,
                                                                    float3(0.299f, 0.587f, 0.114f)));

    return color;
}
```

元の画像から暗い部分をより暗くする計算

PostEffect.h

```
---省略---

class PostEffect
{
public:
    ---省略---

    // 輝度抽出処理
    void LuminanceExtraction(const RenderContext& rc);
    void LuminanceExtraction(const RenderContext& rc, ID3D11ShaderResourceView* colorMap);

    ---省略---

private:
    ---省略---

    struct CbPostEffect
    {
        float    luminanceExtractionLowerEdge = 0.6f;
        float    luminanceExtractionHigherEdge = 0.8f;
        float    padding[2];
    };
    CbPostEffect                            cbPostEffect;
    Microsoft::WRL::ComPtr<ID3D11Buffer>    constantBuffer;
};
```

定数バッファは 16 バイトアライメントでしか作成できないのでパディングする。

PostEffect.cpp

```
---省略---

PostEffect::PostEffect(ID3D11Device* device)
{
    ---省略---

    // 定数バッファ作成
    GpuResourceUtils::CreateConstantBuffer(
        device,
        sizeof(CbPostEffect),
        constantBuffer.GetAddressOf());
}

// 開始処理
void PostEffect::Begin(const RenderContext& rc)
```

```
{
    ---省略---

    // サンプラステート設定
    ID3D11SamplerState* samplers[] =
    {
        renderState->GetSamplerState(SamplerState::LinearWrap)
    };
    dc->PSSetSamplers(0, _countof(samplers), samplers);

    // 定数バッファ設定
    dc->PSSetConstantBuffers(0, 1, constantBuffer.GetAddressOf());

    // 定数バッファ更新
    dc->UpdateSubresource(constantBuffer.Get(), 0, 0, &cbPostEffect, 0, 0);
}

// 輝度抽出処理
void PostEffect::LuminanceExtraction(const RenderContext& rc)
void PostEffect::LuminanceExtraction(const RenderContext& rc, ID3D11ShaderResourceView* colorMap)
{
    ---省略---

    // シェーダーリソース設定
    ID3D11ShaderResourceView* srvs[] = { colorMap };
    dc->PSSetShaderResources(0, _countof(srvs), srvs);

    // 描画
    ---省略---
}

// 終了処理
void PostEffect::End(const RenderContext& rc)
{
    ID3D11DeviceContext* dc = rc.deviceContext;

    // 設定されているシェーダーリソースを解除
    ID3D11ShaderResourceView* srvs[] = { nullptr };
    dc->PSSetShaderResources(0, _countof(srvs), srvs);
}
```

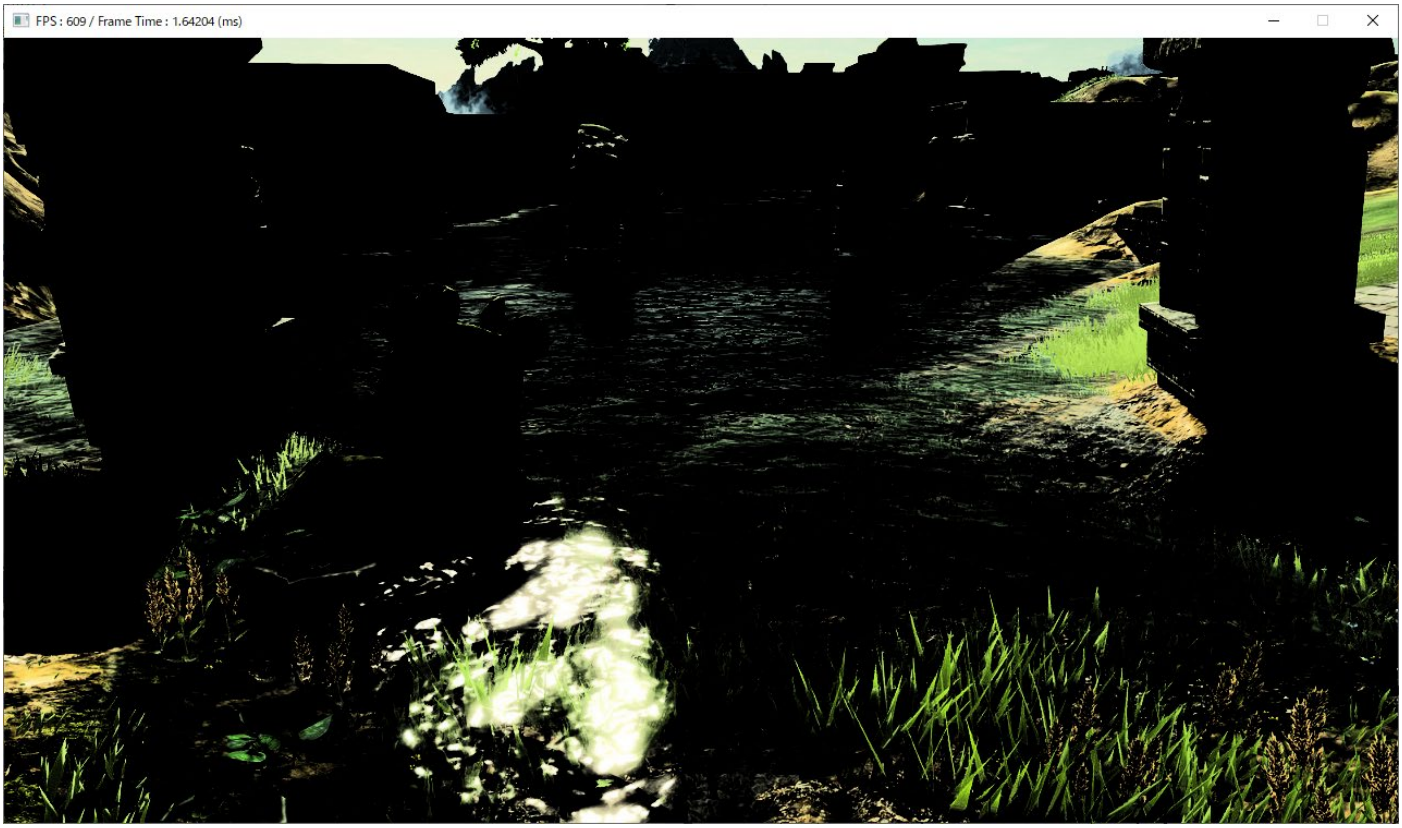
シェーダーリソースの解除をせずに
レンダーターゲットに書き込みをしてしまうと
エラーが起きるので使用した後は解除する

Scene.cpp

```
---省略---

// 描画処理
void PostEffectTestScene::Render(float elapsedTime)
{
    ---省略---

    // フレームバッファを取得
    FrameBuffer* displayFB = Graphics::Instance().GetFrameBuffer(FrameBufferId::Display);
    FrameBuffer* sceneFB = Graphics::Instance().GetFrameBuffer(FrameBufferId::Scene);

    // シーン用のフレームバッファにスプライトを描画
```

```
    sceneFB->SetRenderTargets(dc);

    // スプライト描画
    sprite->Render(dc, 0, 0, 0, screenWidth, screenHeight, 0, 1, 1, 1, 1);

    ---省略---

    // ポストエフェクト処理開始
    postEffect->Begin(rc);

    // 輝度抽出処理
    displayFB->SetRenderTargets(dc); // バックバッファに輝度を抽出した結果を描画
    postEffect->LuminanceExtraction(rc, sceneFB->GetColorMap());

    // 終了処理
    postEffect->End(rc);
}
```

実行確認してみましょう。
明るい部分だけが強調された画面が表示されていれば OK です。



○抽出した輝度をぼかす

輝度が抽出できたので、この画像をぼかす処理を実装し、元の画面に重ねて表示することで光が溢れる演出が実現できます。

PostEffect.hlsli

```
cbuffer CbPostEffect : register(b0)
{
    ---省略---
    float    gaussianSigma;
    float    bloomIntensity;
};
```

ぼかし具合

光の溢れ具合

BloomPS.hlsl を作成しましょう。

BloomPS.hlsl

```
#include "FullScreenQuad.hlsli"
#include "PostEffect.hlsli"

Texture2D         colorMap      : register(t0);
Texture2D         luminanceMap  : register(t1);
SamplerState      linearSampler : register(s0);

float4 main(VS_OUT pin) : SV_TARGET
{
    uint width, height;
    luminanceMap.GetDimensions(width, height);

    float4 color = colorMap.Sample(linearSampler, pin.texcoord);
    float alpha = color.a;

    float3 blurColor = 0;
    float gaussianKernelTotal = 0;

    const float PI = 3.14159265358979f;
    const int gaussianHalfKernelSize = 3;
    [unroll]
    for (int x = -gaussianHalfKernelSize; x <= +gaussianHalfKernelSize; x += 1)
    {
        [unroll]
        for (int y = -gaussianHalfKernelSize; y <= +gaussianHalfKernelSize; y += 1)
        {
            float gaussianKernel = exp(-(x * x + y * y) / (2.0 * gaussianSigma * gaussianSigma)) /
                                                                (2 * PI * gaussianSigma * gaussianSigma);
            blurColor += luminanceMap.Sample(linearSampler, pin.texcoord + float2(x * 1.0 / width, y * 1.0 /
                                                                                    height)).rgb * gaussianKernel;
            gaussianKernelTotal += gaussianKernel;
        }
    }
    blurColor /= gaussianKernelTotal;

    color.rgb += blurColor * bloomIntensity;

    return color;
}
```

ぼかし処理

ぼかした色を元の画像に重ねる

PostEffect.h

```
---省略---

class PostEffect
{
public:
    ---省略---

    // ブルーム処理
    void Bloom(const RenderContext& rc, ID3D11ShaderResourceView* colorMap,
                                                                ID3D11ShaderResourceView* luminanceMap);

    ---省略---

private:
    struct CbPostEffect
    {
        ---省略---
        float    padding[2];
        float    gaussianSigma = 1.0f;
        float    bloomIntensity = 1.0f;
    };
    ---省略---

    Microsoft::WRL::ComPtr<ID3D11PixelShader>       bloomPS;
};
```

PostEffect.cpp

```
---省略---

// コンストラクタ
PostEffect::PostEffect(ID3D11Device* device)
{
    ---省略---

    // ブルームピクセルシェーダー読み込み
    GpuResourceUtils::LoadPixelShader(
        device,
        "Data/Shader/BloomPS.cso",
        bloomPS.GetAddressOf());
}


// ブルーム処理
void PostEffect::Bloom(const RenderContext& rc, ID3D11ShaderResourceView* colorMap,
                                                                ID3D11ShaderResourceView* luminanceMap)
{
    ID3D11DeviceContext* dc = rc.deviceContext;

    // シェーダー設定
    dc->VSSetShader(fullscreenQuadVS.Get(), 0, 0);
    dc->PSSetShader(bloomPS.Get(), 0, 0);
```

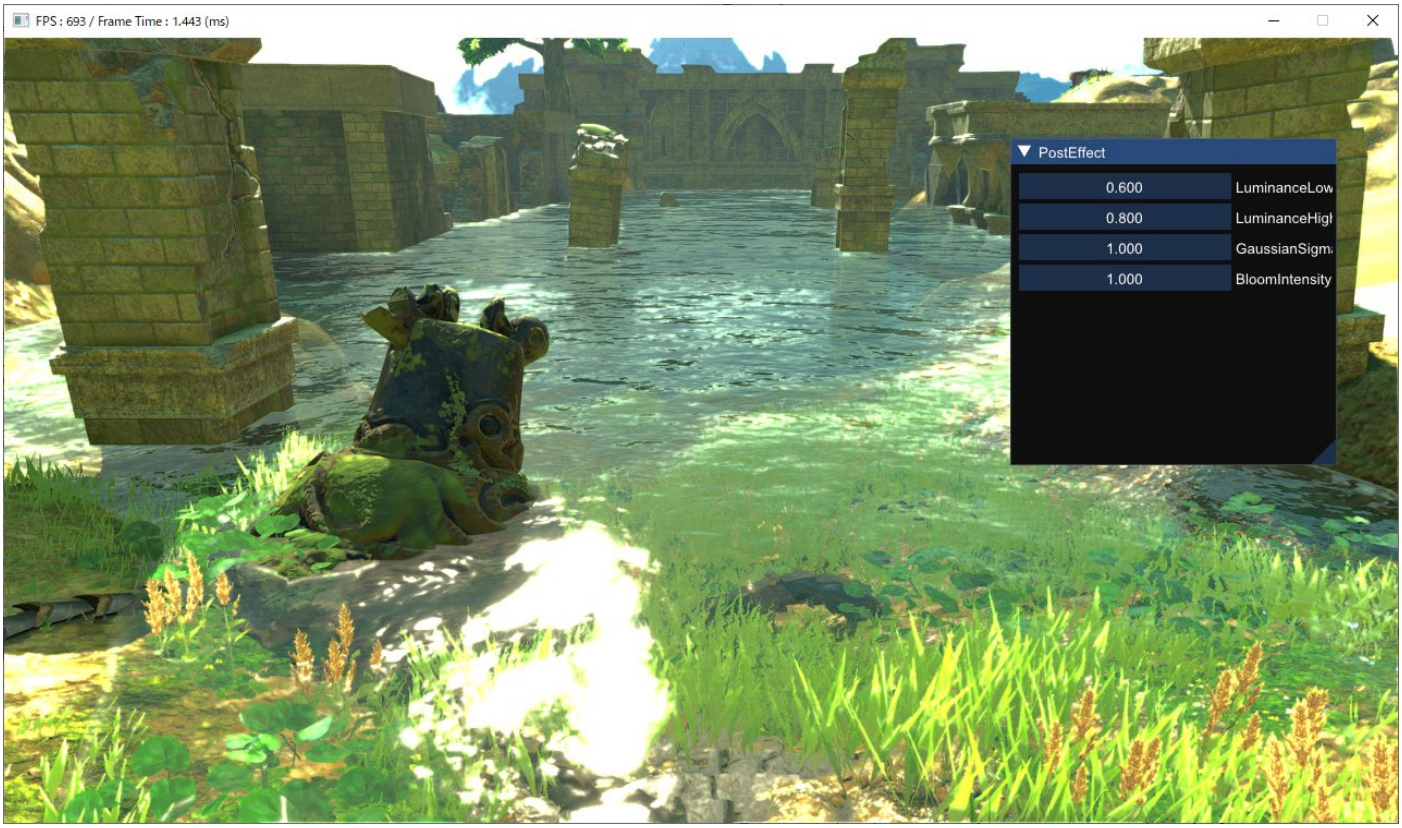
```
    // シェーダーリソース設定
    ID3D11ShaderResourceView* srvs[] = { colorMap, luminanceMap };
    dc->PSSetShaderResources(0, _countof(srvs), srvs);

    // 描画
    dc->Draw(4, 0);
}

// 終了処理
void PostEffect::End(const RenderContext& rc)
{
    ---省略---

    // 設定されているシェーダーリソースを解除
    ~~ID3D11ShaderResourceView* srvs[] = { nullptr };~~
    ID3D11ShaderResourceView* srvs[] = { nullptr, nullptr };
    ---省略---
}
```

設定するシェーダーリソースが
２つになったので全て解除する

Scene.cpp

```
// 描画処理
void PostEffectTestScene::Render(float elapsedTime)
{
    ---省略---

    // フレームバッファを取得
    ---省略---
    FrameBuffer* luminanceFB = Graphics::Instance().GetFrameBuffer(FrameBufferId::Luminance);

    ---省略---

    // 輝度抽出処理
    ~~swapchainFB->SetRenderTargets(dc); // バックバッファに輝度を抽出した結果を描画~~
    luminanceFB->SetRenderTargets(dc); //輝度抽出用のフレームバッファに描画
    postEffect->LuminanceExtraction(rc, sceneFB->GetColorMap());

    // ブルーム処理
    displayFB->SetRenderTargets(dc); // バックバッファにブルーム処理した結果を描画
    postEffect->Bloom(rc, sceneFB->GetColorMap(), luminanceFB->GetColorMap());

    ---省略---
}
```

実行確認してみましょう。
下図のように光が溢れるような画面になっていれば OK です。



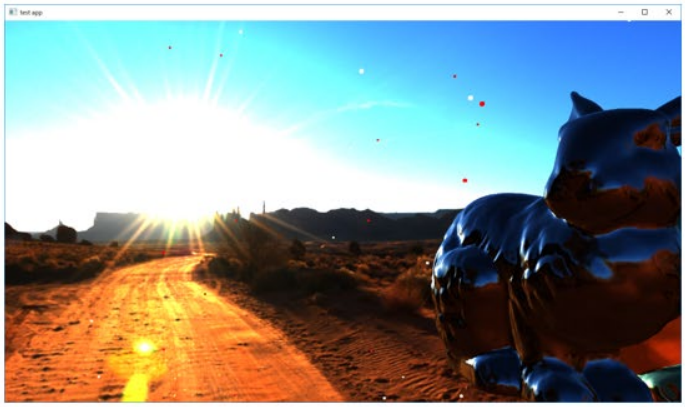
○トーンマッピング

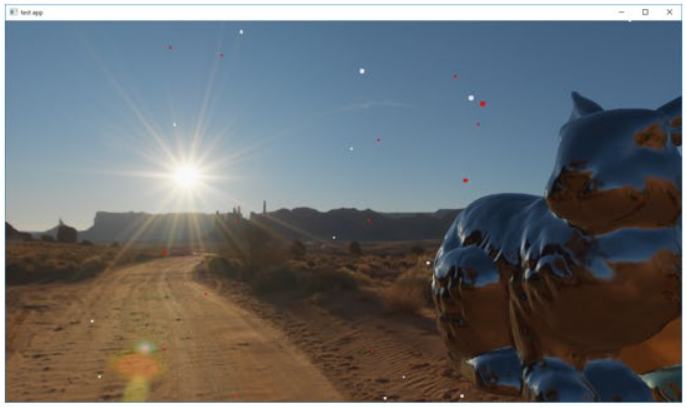
現状の実装では明るい部分が白飛びしてしまっています。
通常、RGB の値は 0.0～1.0 で表現されますが、今回、ブルームの処理によって明るい色がより明るくなってしまって 1.0 を超えてしまっているためです。

今回、シーンと輝度を描くためのフレームバッファは 1.0 以上の色を書き込める HDR フォーマットになっていましたが、画面表示用のフレームバッファは 1.0 以上の色を書き込めないフォーマットになっています。

画面表示用のフレームバッファを HDR フォーマットにする解決方法もありますが、そうすると PC モニターも HDR 対応のものを用意しなければいけません。

今回はトーンマッピングという技術を使って画面全体の色の値を全体的に下げることで白飛びをなくします。

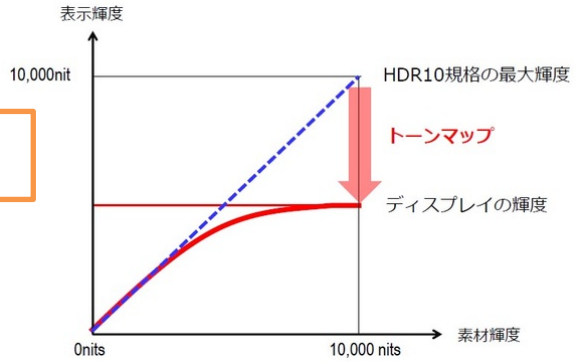




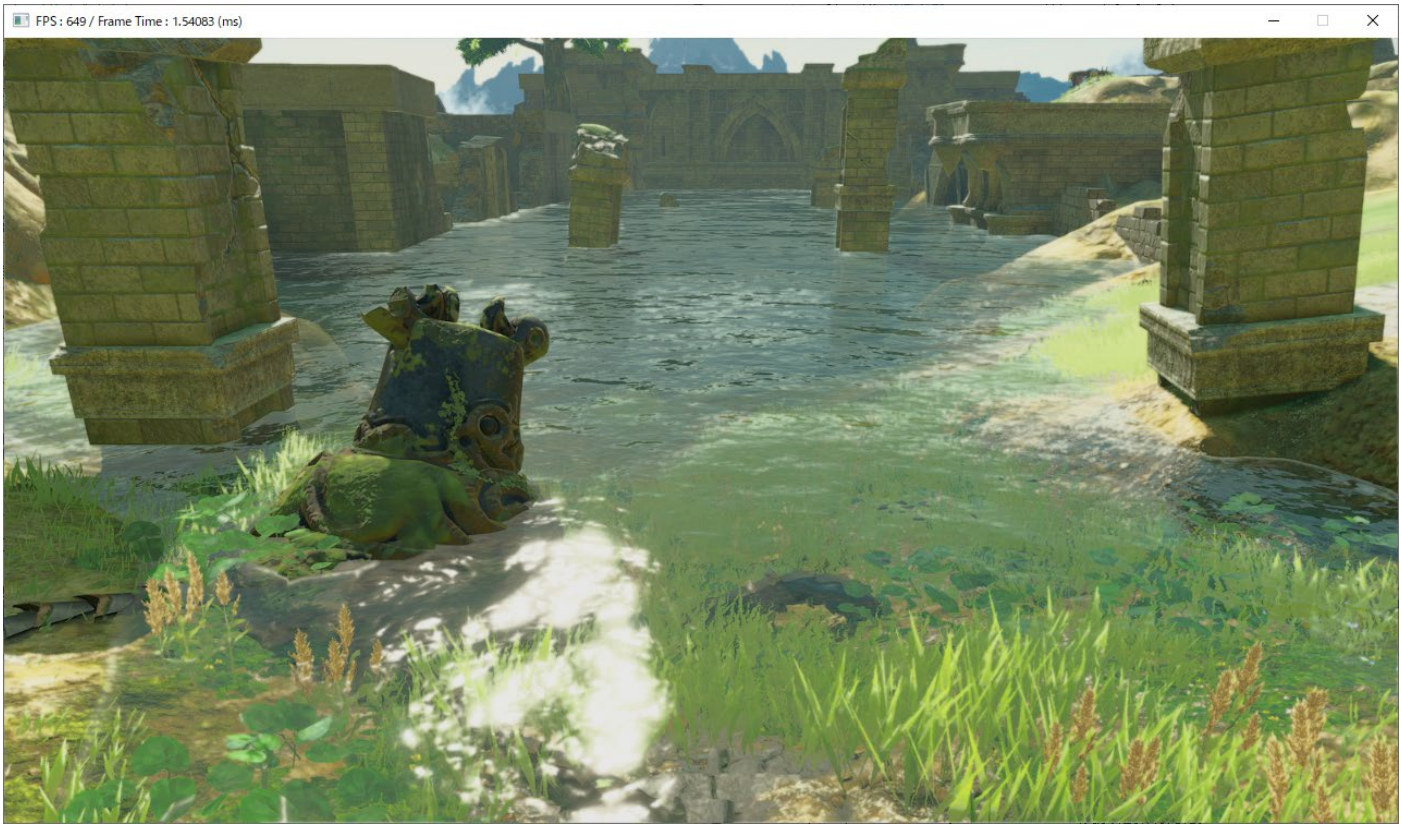
Bloom.hlsl

```
---省略---

float4 main(VS_OUT pin) : SV_TARGET
{
    ---省略---

    const float exposure = 1.2;
    color.rgb = 1 - exp(-color.rgb * exposure);

    return color;
}
```

トーンマッピング



実行確認してみましょう。
白飛びが軽減され、落ち着いた画面になっていれば OK です。



○デバッグメニュー表示

デバッグメニューを表示し、ポストエフェクトのパラメーターを調整しましょう。

PostEffect.h

```
---省略---

class PostEffect
{
public:
	---省略---

	// デバッグGUI描画
	void DrawDebugGUI();

	---省略---
};
```

PostEffect.cpp

```
#include <imgui.h>
---省略---

// デバッグGUI描画
void PostEffect::DrawDebugGUI()
{
```

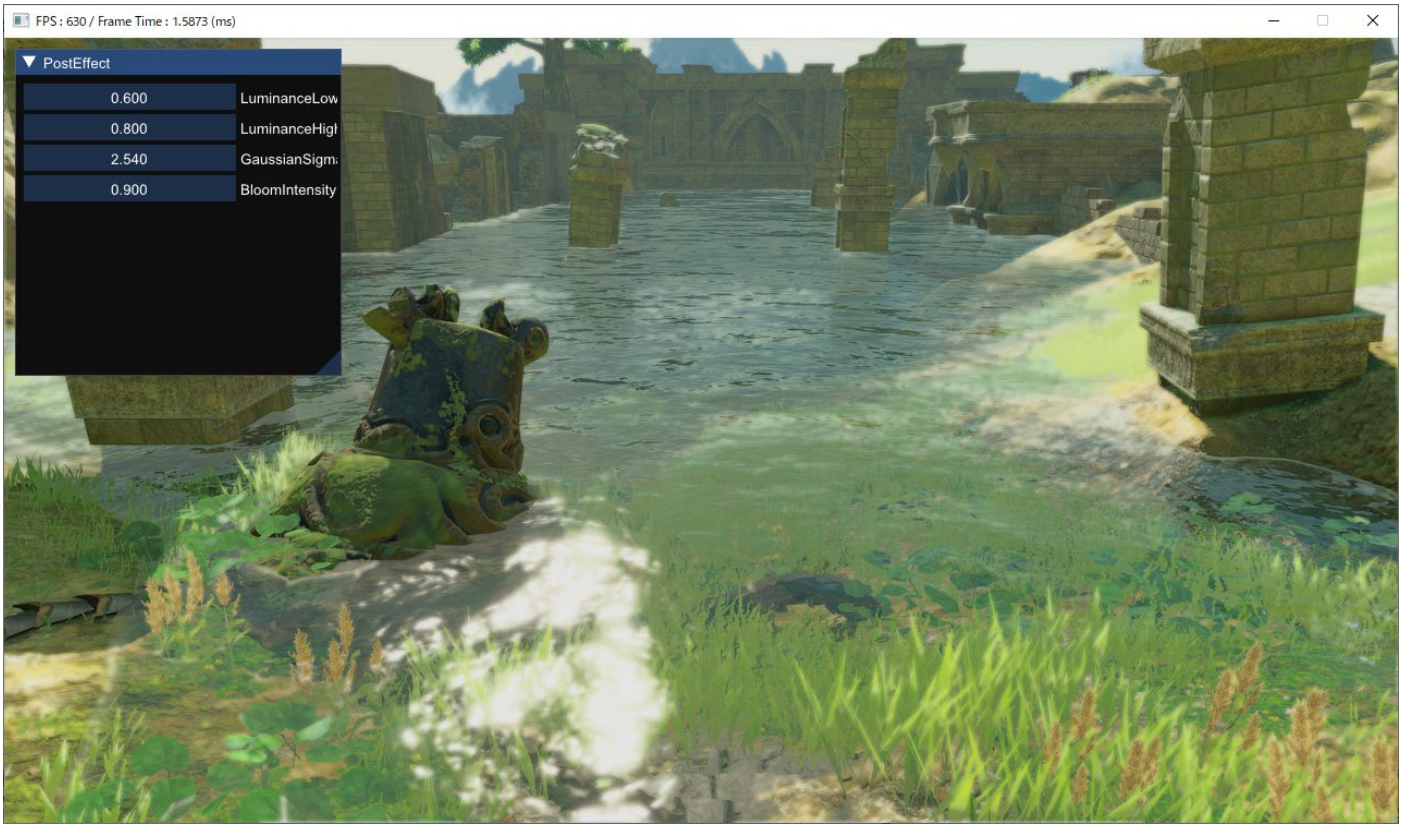
```
    ImGui::DragFloat("LuminanceLowerEdge", &cbPostEffect.luminanceExtractionLowerEdge, 0.01f, 0, 1.0f);
    ImGui::DragFloat("LuminanceHigherEdge", &cbPostEffect.luminanceExtractionHigherEdge, 0.01f, 0, 1.0f);
    ImGui::DragFloat("GaussianSigma", &cbPostEffect.gaussianSigma, 0.01f, 0, 10.0f);
    ImGui::DragFloat("BloomIntensity", &cbPostEffect.bloomIntensity, 0.1f, 0, 10.0f);
}
```

Scene.h

```
---省略---

// ポストエフェクトテストシーン
class PostEffectTestScene : public Scene
{
    ---省略---

private:
    // ポストエフェクトGUI描画
    void DrawPostEffectGUI();

    ---省略---
};
```

Scene.cpp

```
---省略---

// 描画処理
void PostEffectTestScene::Render(float elapsedTime)
{
    ---省略---

    // デバッグGUI描画
    DrawPostEffectGUI();
}

// ポストエフェクトGUI描画
void PostEffectTestScene::DrawPostEffectGUI()
{
    ImVec2 pos = ImGui::GetMainViewport()->GetWorkPos();
    ImGui::SetNextWindowPos(ImVec2(pos.x + 10, pos.y + 10), ImGuiCond_FirstUseEver);
    ImGui::SetNextWindowSize(ImVec2(300, 300), ImGuiCond_FirstUseEver);

    ImGui::Begin("PostEffect", nullptr, ImGuiWindowFlags_None);

    postEffect->DrawDebugGUI();

    ImGui::End();
}
```


FPS : 630 / Frame Time : 1.5873 (ms)
PostEffect
0.600
LuminanceLow
0.800
LuminanceHigh
2.540
GaussianSigma
0.900
BloomIntensity