

描画エンジン開発 EX

○概要

NuGet を利用して DirectXTex ライブラリを導入する。

画像ファイルを読み込み、スプライトにテクスチャを表示できるようにする。

画像ファイル内の任意の領域をスプライトで表示できるようにする。

サンプルステートの違いを確認する。

○DirectXTex

DirectXTex とは画像ファイルを読み込み、簡単にテクスチャを生成できるライブラリです。

今回は NuGet という Microsoft が提供するライブラリの導入環境を利用して DirectXTex を導入します。

下図の手順に沿って DirectXTex をインストールしましょう。

The screenshot illustrates the process of installing DirectXTex in Visual Studio. It shows the 'Tools' menu with 'Solution NuGet Package Manager' selected. The NuGet interface displays the search results for 'DirectXTex', with 'directxtex_desktop_2019' highlighted. The package details pane shows the 'Game' checkbox checked, and the 'Install' button is visible.

○シェーダー対応

テクスチャを利用するためシェーダープログラムを改造します。

描画エンジン開発 EX

Sprite.hlsl

```
// 頂点シェーダー出力データ
struct VS_OUT
{
    float4 position : SV_POSITION;
    float4 color      : COLOR;
    float2 texcoord  : TEXCOORD;
};
```

SpriteVS.hlsl

```
#include "Sprite.hlsl"

// 頂点シェーダーエントリポイント
VS_OUT main(float4 position : POSITION, float4 color : COLOR, float2 texcoord : TEXCOORD)
{
    VS_OUT vout;
    vout.position = position;
    vout.color = color;
    vout.texcoord = texcoord;

    return vout;
}
```

SpritePS.hlsl

```
#include "sprite.hlsl"

Texture2D spriteTexture : register(t0);
SamplerState spriteSampler : register(s0);

// ピクセルシェーダーエントリポイント
float4 main(VS_OUT pin) : SV_TARGET
{
    return spriteTexture.Sample(spriteSampler, pin.texcoord) * pin.color;
}
```

○テクスチャ読み込み

DirectXTex を利用してテクスチャを読み込みます。

テクスチャの読み込みはよく利用するのでユーティリティ関数を作成しておきましょう。

また、シェーダーファイルの読み込みもよく利用するので同じく作成しておきましょう。

GpuResourceUtils.cpp と GpuResourceUtils.h を作成しましょう。

GpuResourceUtils.h

```
#pragma once

#include <d3d11.h>

// GPUリソースユーティリティ
```

```
class GpuResourceUtils
{
public:
    // 頂点シェーダー読み込み
    static HRESULT LoadVertexShader(
        ID3D11Device* device,
        const char* filename,
        const D3D11_INPUT_ELEMENT_DESC inputElementDescs[],
        UINT inputElementCount,
        ID3D11InputLayout** inputLayout,
        ID3D11VertexShader** vertexShader);

    // ピクセルシェーダー読み込み
    static HRESULT LoadPixelShader(
        ID3D11Device* device,
        const char* filename,
        ID3D11PixelShader** pixelShader);

    // テクスチャ読み込み
    static HRESULT LoadTexture(
        ID3D11Device* device,
        const char* filename,
        ID3D11ShaderResourceView** shaderResourceView,
        D3D11_TEXTURE2D_DESC* texture2dDesc = nullptr);
};
```

GpuResourceUtils.cpp

```
#include <filesystem>
#include <wrl.h>
#include <DirectXTex.h>
#include "Misc.h"
#include "GpuResourceUtils.h"

// 頂点シェーダー読み込み
HRESULT GpuResourceUtils::LoadVertexShader(
    ID3D11Device* device,
    const char* filename,
    const D3D11_INPUT_ELEMENT_DESC inputElementDescs[],
    UINT inputElementCount,
    ID3D11InputLayout** inputLayout,
    ID3D11VertexShader** vertexShader)
{
    // ファイルを開く
    FILE* fp = nullptr;
    fopen_s(&fp, filename, "rb");
    _ASSERT_EXPR_A(fp, "Vertex Shader File not found");

    // ファイルのサイズを求める
    fseek(fp, 0, SEEK_END);
    long size = ftell(fp);
    fseek(fp, 0, SEEK_SET);

    // メモリ上に頂点シェーダーデータを格納する領域を用意する
    std::unique_ptr<u_char[]> data = std::make_unique<u_char[]>(size);
```

描画エンジン開発 EX

```
fread(data.get(), size, 1, fp);
fclose(fp);

// 頂点シェーダー生成
HRESULT hr = device->CreateVertexShader(data.get(), size, nullptr, vertexShader);
_ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

// 入力レイアウト
if (inputLayout != nullptr)
{
    hr = device->CreateInputLayout(inputElementDescs, inputElementCount, data.get(), size, inputLayout);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}

return hr;
}

// ピクセルシェーダー読み込み
HRESULT GpuResourceUtils::LoadPixelShader(
    ID3D11Device* device,
    const char* filename,
    ID3D11PixelShader** pixelShader)
{
    // ファイルを開く
    FILE* fp = nullptr;
    fopen_s(&fp, filename, "rb");
    _ASSERT_EXPR_A(fp, "Pixel Shader File not found");

    // ファイルのサイズを求める
    fseek(fp, 0, SEEK_END);
    long size = ftell(fp);
    fseek(fp, 0, SEEK_SET);

    // メモリ上に頂点シェーダーデータを格納する領域を用意する
    std::unique_ptr<u_char[]> data = std::make_unique<u_char[]>(size);
    fread(data.get(), size, 1, fp);
    fclose(fp);

    // ピクセルシェーダー生成
    HRESULT hr = device->CreatePixelShader(data.get(), size, nullptr, pixelShader);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

    return hr;
}

// テクスチャ読み込み
HRESULT GpuResourceUtils::LoadTexture(
    ID3D11Device* device,
    const char* filename,
    ID3D11ShaderResourceView** shaderResourceView,
    D3D11_TEXTURE2D_DESC* texture2dDesc)
{
    // 拡張子を取得
    std::filesystem::path filepath(filename);
    std::string extension = filepath.extension().string();
    std::transform(extension.begin(), extension.end(), extension.begin(), tolower); // 小文字化
```

描画エンジン開発 EX

```
// ワイド文字に変換
std::wstring wfilename = filepath.wstring();

// フォーマット毎に画像読み込み処理
HRESULT hr;
DirectX::TexMetadata metadata;
DirectX::ScratchImage scratch_image;
if (extension == ".tga")
{
    hr = DirectX::GetMetadataFromTGAFFile(wfilename.c_str(), metadata);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

    hr = DirectX::LoadFromTGAFFile(wfilename.c_str(), &metadata, scratch_image);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}
else if (extension == ".dds")
{
    hr = DirectX::GetMetadataFromDDSFile(wfilename.c_str(), DirectX::DDS_FLAGS_NONE, metadata);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

    hr = DirectX::LoadFromDDSFile(wfilename.c_str(), DirectX::DDS_FLAGS_NONE, &metadata, scratch_image);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}
else if (extension == ".hdr")
{
    hr = DirectX::GetMetadataFromHDRFile(wfilename.c_str(), metadata);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

    hr = DirectX::LoadFromHDRFile(wfilename.c_str(), &metadata, scratch_image);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}
else
{
    hr = DirectX::GetMetadataFromWICFile(wfilename.c_str(), DirectX::WIC_FLAGS_NONE, metadata);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

    hr = DirectX::LoadFromWICFile(wfilename.c_str(), DirectX::WIC_FLAGS_NONE, &metadata, scratch_image);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}

// シェーダーリソースビュー作成
hr = DirectX::CreateShaderResourceView(device, scratch_image.GetImages(), scratch_image.GetImageCount(),
                                         metadata, shaderResourceView);
_ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

// テクスチャ情報取得
if (texture2dDesc != nullptr)
{
    Microsoft::WRL::ComPtr<ID3D11Resource> resource;
    (*shaderResourceView->GetResource(resource.GetAddressOf()));

    Microsoft::WRL::ComPtr<ID3D11Texture2D> texture2d;
    hr = resource->QueryInterface<ID3D11Texture2D>(texture2d.GetAddressOf());
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    texture2d->GetDesc(texture2dDesc);
}
return hr;
```

```
}
```

Sprite クラスでテクスチャの読み込みと描画設定をしましょう。

Sprite.h

```
#pragma once

#include <wrl.h>
#include <d3d11.h>
#include <DirectXMath.h>

// スプライト
class Sprite
{
public:
    Sprite(ID3D11Device* device, const char* filename);

    // 頂点データ
    struct Vertex
    {
        ---省略---
        DirectX::XMFLOAT2 texcoord;
    };

    ---省略---

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

Sprite.cpp

```
---省略---
#include "GpuResourceUtils.h"

// コンストラクタ
Sprite::Sprite(ID3D11Device* device, const char* filename)
{
    HRESULT hr = S_OK;

    ---省略---

    // 頂点シェーダー
    {
        // 入力レイアウト
        D3D11_INPUT_ELEMENT_DESC inputElementDesc[] =
        {
            ---省略---
            { "TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT,
              D3D11_INPUT_PER_VERTEX_DATA, 0 },
        };
    };
};
```

描画エンジン開発 EX

```
hr = GpuResourceUtils::LoadVertexShader (
    device,
    "Data/Shader/SpriteVS.cso",
    inputElementDesc,
    ARRAYSIZE(inputElementDesc),
    inputLayout.GetAddressOf(),
    vertexShader.GetAddressOf());
_ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}

// ピクセルシェーダー
{
    hr = GpuResourceUtils::LoadPixelShader (
        device,
        "Data/Shader/SpritePS.cso",
        pixelShader.GetAddressOf());
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}

// テクスチャの生成
{
    // テクスチャファイル読み込み
    hr = GpuResourceUtils::LoadTexture(device, filename, shaderResourceView.GetAddressOf());
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}

// 描画実行
void Sprite::Render (---省略---) const
{
    // 頂点座標
    ---省略---

    // テクスチャ座標
    DirectX::XMFLOAT2 texcoords[] = {
        DirectX::XMFLOAT2(0.0f, 0.0f), // 左上
        DirectX::XMFLOAT2(1.0f, 0.0f), // 右上
        DirectX::XMFLOAT2(0.0f, 1.0f), // 左下
        DirectX::XMFLOAT2(1.0f, 1.0f), // 右下
    };

    ---省略---

    // 頂点バッファの内容を編集
    Vertex* v = static_cast<Vertex*>(mappedSubresource.pData);
    for (int i = 0; i < 4; ++i)
    {
        ---省略---

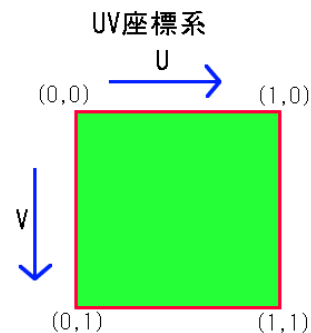
        v[i].texcoord.x = texcoords[i].x;
        v[i].texcoord.y = texcoords[i].y;
    }

    ---省略---

    // GPUに描画するためのデータを渡す
    ---省略---
```

ユーティリティ関数に
置き換え

ユーティリティ関数に
置き換え



テクスチャ座標は
0.0～1.0の間で表現する

描画エンジン開発 EX

```
dc->PSSetShaderResources(0, 1, shaderResourceView.GetAddressOf());
```

```
---省略---
```

```
}
```

テクスチャを設定

Scene.cpp

```
#include "Scene.h"
#include "Graphics.h"

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

    sprites[0] = std::make_unique<Sprite>(device, "Data/Sprite/player-sprites.png");
}

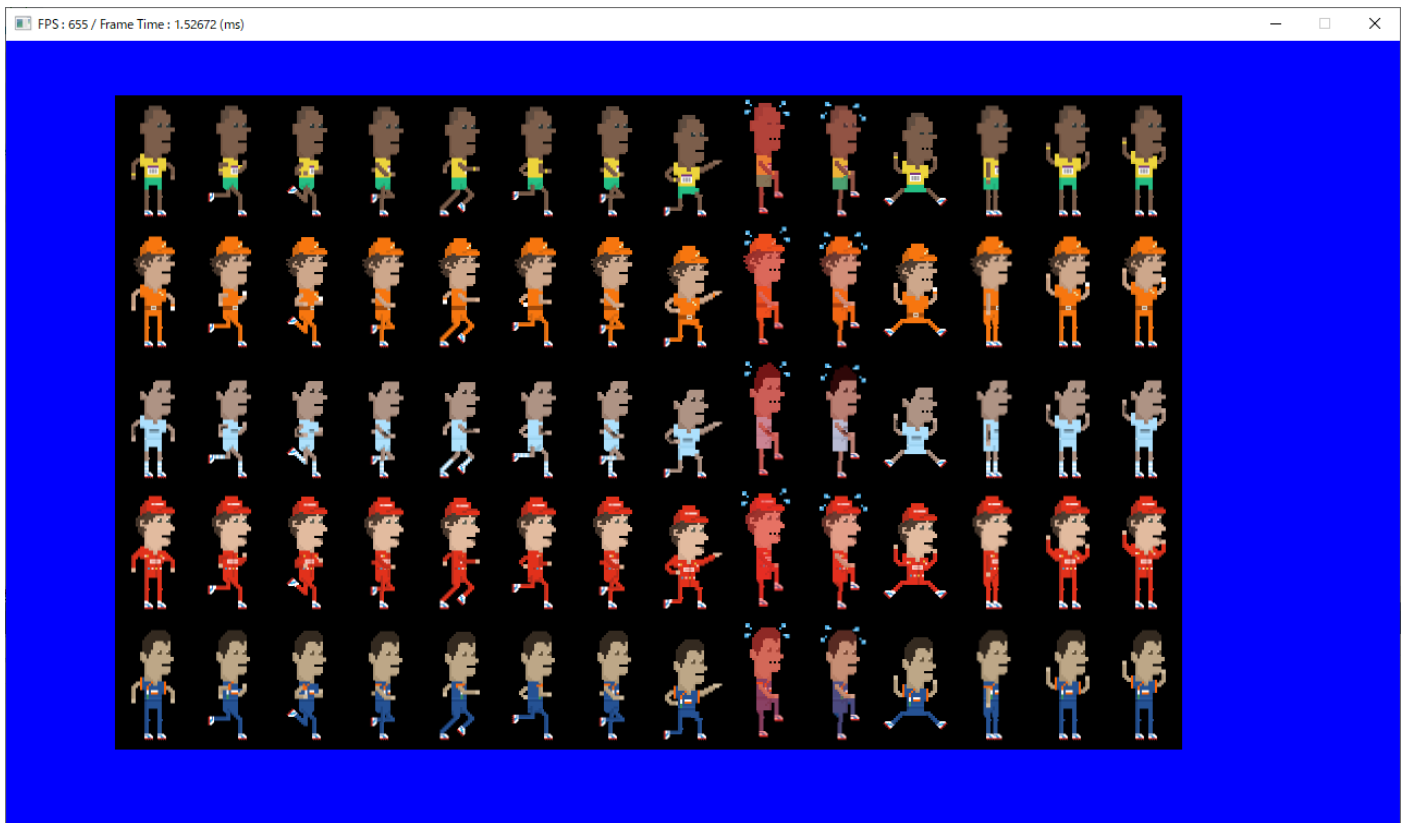
// 描画処理
void SpriteTestScene::Render(float elapsedTime)
{
    ID3D11DeviceContext* dc = Graphics::Instance().GetDeviceContext();

    sprites[0]->Render(dc, 100, 50, 980, 600, 0, 1, 1, 1, 1);
}
```

実行確認してみましょう。

下図のようにスプライトに画像が表示されていれば OK です。

描画エンジン開発 EX



○テクスチャの指定領域の切り抜き

画像内の指定領域をスプライトに表示できるようにします。

テクスチャ空間での座標は 0.0~1.0 の間で表現されますが、描画する際に指定しやすいようにピクセルサイズで領域指定できるようにしましょう。

Sprite.h

---省略---

```
class Sprite
```

```
{
```

```
public:
```

---省略---

```
// 描画実行
```

```
void Render(ID3D11DeviceContext* dc,
```

```
float dx, float dy,
```

```
// 左上位置
```

```
float dw, float dh,
```

```
// 幅、高さ
```

```
float sx, float sy,
```

```
// 画像切り抜き位置
```

```
float sw, float sh,
```

```
// 画像切り抜きサイズ
```

```
float angle,
```

```
// 角度
```

```
float r, float g, float b, float a
```

```
// 色
```

```
) const;
```

```
private:
```

---省略---

描画エンジン開発 EX

```
float textureWidth = 0;
float textureHeight = 0;
};
```

Sprite.cpp

---省略---

// コンストラクタ

```
Sprite::Sprite(ID3D11Device* device, const char* filename)
```

```
{
```

---省略---

// テクスチャの生成

```
{
```

// テクスチャファイル読み込み

```
D3D11_TEXTURE2D_DESC desc;
```

```
hr = GpuResourceUtils::LoadTexture(device, filename, shaderResourceView.GetAddressOf());
```

```
hr = GpuResourceUtils::LoadTexture(device, filename, shaderResourceView.GetAddressOf(), &desc);
```

```
_ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
```

```
textureWidth = static_cast<float>(desc.Width);
```

```
textureHeight = static_cast<float>(desc.Height);
```

```
}
```

```
}
```

// 描画実行

```
void Sprite::Render(ID3D11DeviceContext* dc,
```

```
float dx, float dy, // 左上位置
```

```
float dw, float dh, // 幅、高さ
```

```
float sx, float sy, // 画像切り抜き位置
```

```
float sw, float sh, // 画像切り抜きサイズ
```

```
float angle, // 角度
```

```
float r, float g, float b, float a // 色
```

```
) const
```

```
{
```

---省略---

// テクスチャ座標

```
DirectX::XMFLOAT2 texcoords[] = {
```

```
DirectX::XMFLOAT2(0.0f, 0.0f), // 左上
```

```
DirectX::XMFLOAT2(1.0f, 0.0f), // 右上
```

```
DirectX::XMFLOAT2(0.0f, 1.0f), // 左下
```

```
DirectX::XMFLOAT2(1.0f, 1.0f), // 右下
```

```
DirectX::XMFLOAT2(sx, sy), // 左上
```

```
DirectX::XMFLOAT2(sx + sw, sy), // 右上
```

```
DirectX::XMFLOAT2(sx, sy + sh), // 左下
```

```
DirectX::XMFLOAT2(sx + sw, sy + sh), // 右下
```

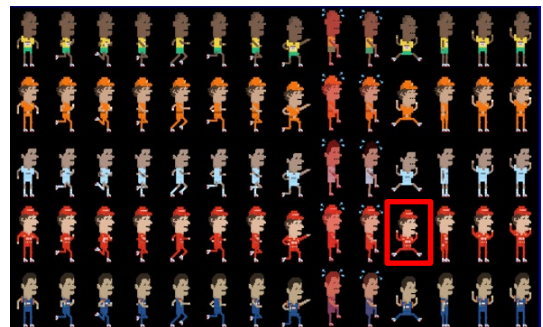
```
};
```

---省略---

// 頂点バッファの内容を編集

```
Vertex* v = static_cast<Vertex*>(mappedSubresource.pData);
```

テクスチャ読み込み時に
テクスチャサイズを取得



ピクセル単位で切り抜く
画像の領域を指定

描画エンジン開発 EX

```
for (int i = 0; i < 4; ++i)
{
    ---省略---

    v[i].texcoord.x = texcoords[i].x;
    v[i].texcoord.y = texcoords[i].y;

    v[i].texcoord.x = texcoords[i].x / textureWidth;
    v[i].texcoord.y = texcoords[i].y / textureHeight;
}

---省略---
```

ピクセル単位の座標を
テクスチャ空間の座標
(0.0~1.0)に変換する

Scene.cpp

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

    sprites[0]->Render(dc, 100, 50, 980, 600, 0, 1, 1, 1, 1);

    float width = 140;
    float height = 240;
    sprites[0]->Render(dc, 100, 50, width, height, 10 * width, 3 * height, width, height, 0, 1, 1, 1, 1);
}
```

実行確認してみましょう。

指定領域の画像を切り抜いて表示されていれば OK です。



○テクスチャなしスプライトの描画

ここまでの実装でスプライトにテクスチャを表示できるようになりました。
しかし、逆に画像ファイルがないとスプライトが表示できなくなってしまったので、画像ファイルなしでもスプライトが表示できるように対応します。

GpuResourceUtils.h

```
---省略---

// GPUリソースユーティリティ
class GpuResourceUtils
{
public:
    ---省略---

    // ダミーテクスチャ作成
    static HRESULT CreateDummyTexture(
        ID3D11Device* device,
        UINT color,
        ID3D11ShaderResourceView** shaderResourceView,
        D3D11_TEXTURE2D_DESC* texture2dDesc = nullptr);
};
```

GpuResourceUtils.cpp

描画エンジン開発 EX

---省略---

// ダミーテクスチャ作成

```
HRESULT GpuResourceUtils::CreateDummyTexture(
    ID3D11Device* device,
    UINT color,
    ID3D11ShaderResourceView** shaderResourceView,
    D3D11_TEXTURE2D_DESC* texture2dDesc)
{
    D3D11_TEXTURE2D_DESC desc = { 0 };
    desc.Width = 1;
    desc.Height = 1;
    desc.MipLevels = 1;
    desc.ArraySize = 1;
    desc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
    desc.SampleDesc.Count = 1;
    desc.SampleDesc.Quality = 0;
    desc.Usage = D3D11_USAGE_IMMUTABLE;
    desc.BindFlags = D3D11_BIND_SHADER_RESOURCE;
    desc.CPUAccessFlags = 0;
    desc.MiscFlags = 0;
    D3D11_SUBRESOURCE_DATA data{};
    data.pSysMem = &color;
    data.SysMemPitch = desc.Width;

    Microsoft::WRL::ComPtr<ID3D11Texture2D> texture;
    HRESULT hr = device->CreateTexture2D(&desc, &data, texture.GetAddressOf());
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

    hr = device->CreateShaderResourceView(texture.Get(), nullptr, shaderResourceView);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

    // テクスチャ情報取得
    if (texture2dDesc != nullptr)
    {
        Microsoft::WRL::ComPtr<ID3D11Resource> resource;
        (*shaderResourceView)->GetResource(resource.GetAddressOf());

        Microsoft::WRL::ComPtr<ID3D11Texture2D> texture2d;
        hr = resource->QueryInterface<ID3D11Texture2D>(texture2d.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
        texture2d->GetDesc(texture2dDesc);
    }

    return hr;
}
```

色は 16 進数で指定する

0xFFFFFFFF
(A)(R)(G)(B)

1x1 の指定した色の
テクスチャを作成する

0xFF を 10 進数で
表現すると 255
つまり各成分の色を
0~255 で指定する

Sprite.h

---省略---

// スプライト

```
class Sprite
{
public:
```

描画エンジン開発 EX

```
Sprite(ID3D11Device* device);
Sprite(ID3D11Device* device, const char* filename);

---省略---

// 描画実行
void Render(ID3D11DeviceContext* dc,
    float dx, float dy,           // 左上位置
    float dw, float dh,           // 幅、高さ
    float sx, float sy,           // 画像切り抜き位置
    float sw, float sh,           // 画像切り抜きサイズ
    float angle,                  // 角度
    float r, float g, float b, float a // 色
) const;

// 描画実行（テクスチャ切り抜き指定なし）
void Render(ID3D11DeviceContext* dc,
    float dx, float dy,           // 左上位置
    float dw, float dh,           // 幅、高さ
    float angle,                  // 角度
    float r, float g, float b, float a // 色
) const;

---省略---
};
```

Sprite.cpp

```
---省略---

// コンストラクタ
Sprite::Sprite(ID3D11Device* device)
    : Sprite(device, nullptr)
{
}

// コンストラクタ
Sprite::Sprite(ID3D11Device* device, const char* filename)
{
    ---省略---

    // テクスチャの生成
    if (filename != nullptr)
    {
        // テクスチャファイル読み込み
        D3D11_TEXTURE2D_DESC desc;
        hr = GpuResourceUtils::LoadTexture(device, filename, shaderResourceView.GetAddressOf(), &desc);
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

        textureWidth = static_cast<float>(desc.Width);
        textureHeight = static_cast<float>(desc.Height);
    }
    else
    {
        // ダミーテクスチャ生成
```

画像ファイルを使わない
場合は nullptr 指定

描画エンジン開発 EX

```
D3D11_TEXTURE2D_DESC desc;
hr = GpuResourceUtils::CreateDummyTexture(device, 0xFFFFFFFF, shaderResourceView.GetAddressOf(),
                                            &desc);

_ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

textureWidth = static_cast<float>(desc.Width);
textureHeight = static_cast<float>(desc.Height);
}

---省略---
```

画像ファイルを使わない場合は
白色のテクスチャを作成する

```
// 描画実行（テクスチャ切り抜き指定なし）
void Sprite::Render(ID3D11DeviceContext* dc,
    float dx, float dy,           // 左上位置
    float dw, float dh,           // 幅、高さ
    float angle,                  // 角度
    float r, float g, float b, float a // 色
) const
{
    Render(dc, dx, dy, dw, dh, 0, 0, textureWidth, textureHeight, angle, r, g, b, a);
}
```

テクスチャ領域全体を指定

Scene.cpp

```
---省略---
```

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

    sprites[0] = std::make_unique<Sprite>(device, "Data/Sprite/player-sprites.png");
    sprites[1] = std::make_unique<Sprite>(device);
}

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

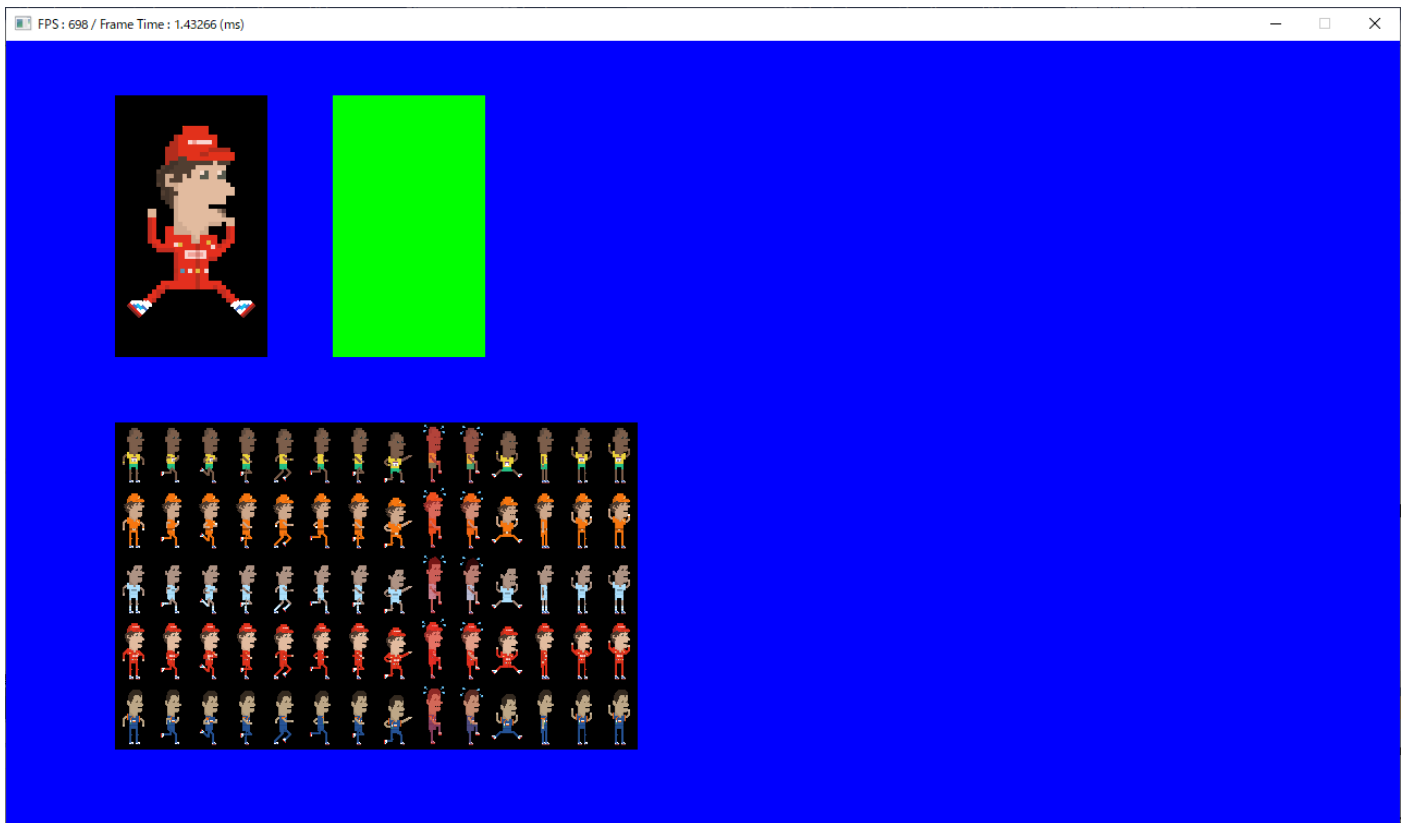
    float width = 140;
    float height = 240;
    sprites[0]→Render(dc, 100, 50, width, height, 10 * width, 3 * height, width, height, 0, 1, 1, 1, 1);
    sprites[0]→Render(dc, 100, 350, 480, 300, 0, 1, 1, 1, 1);
    sprites[1]→Render(dc, 300, 50, width, height, 0, 0, 1, 0, 1);
}
```

画像ファイルを指定しない

実行確認してみましょう。

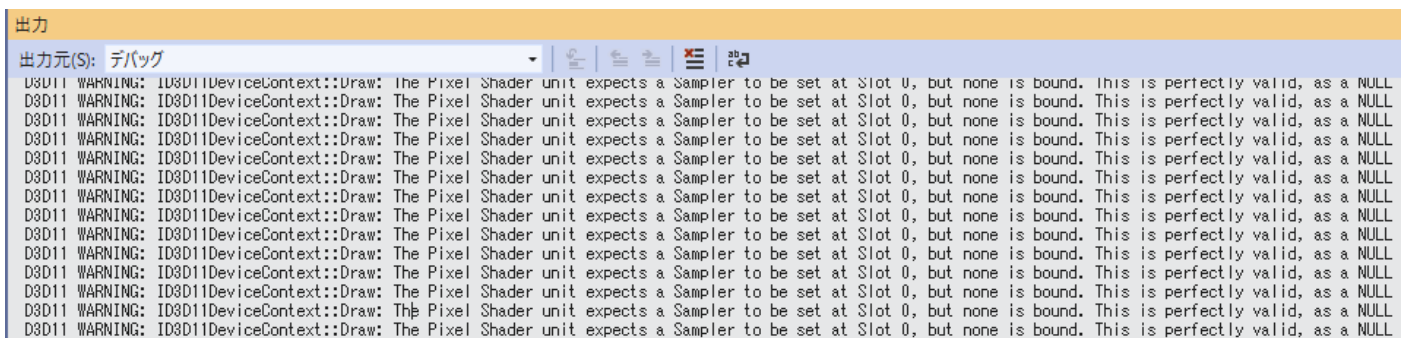
テクスチャなしのSpriteの表示とテクスチャ座標の指定なしで画像全体が見えるSpriteが表示されていればOKです。

描画エンジン開発 EX



○サンプリステート

プログラム実行中に Visual Studio の出力ウィンドウを確認してください。
下図のようにエラーが出力されているはずです。



これはピクセルシェーダーにサンプリステートというリソースが渡されていないからです。

```
#include "sprite.hlsl"

Texture2D spriteTexture : register(t0);
SamplerState spriteSampler : register(s0);

// ピクセルシェーダーエントリポイント
float4 main(VS_OUT pin) : SV_TARGET
{
    return spriteTexture.Sample(spriteSampler, pin.texcoord) * pin.color;
}
```

これが設定されていない

テクスチャから指定位置の色を取り出す

描画エンジン開発 EX

サンプリステートとはテクスチャから色を取り出し方を指定するリソースのことです。
サンプリステートはテクスチャ座標の指定が 0.0~1.0 以外だった場合の色の取り出し方や色を取り出す際の補完方法を指定します。

様々な描画ステートを管理する `RenderState` クラスを作成します。
`RenderState.cpp` と `RenderState.h` を作成しましょう。

RenderState.h

```
#pragma once

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

// サンプリステート
enum class SamplerState
{
    PointWrap,
    PointClamp,
    LinearWrap,
    LinearClamp,

    EnumCount
};

// レンダーステート
class RenderState
{
public:
    RenderState(ID3D11Device* device);
    ~RenderState() = default;

    // サンプリステート取得
    ID3D11SamplerState* GetSamplerState(SamplerState state) const
    {
        return samplerStates[static_cast<int>(state)].Get();
    }

private:
    Microsoft::WRL::ComPtr<ID3D11SamplerState>
        samplerStates[static_cast<int>(SamplerState::EnumCount)];
};
```

RenderState.cpp

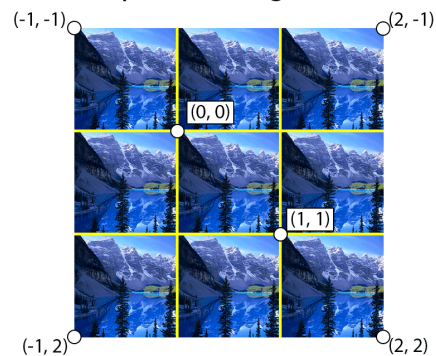
```
#include "Misc.h"
#include "RenderState.h"

// コンストラクタ
RenderState::RenderState(ID3D11Device* device)
```

```
// ポイントサンプリング&テクスチャ繰り返しあり
```

```
{
    D3D11_SAMPLER_DESC desc;
    desc.MipLODBias = 0.0f;
    desc.MaxAnisotropy = 1;
    desc.ComparisonFunc = D3D11_COMPARISON_NEVER;
    desc.MinLOD = -D3D11_FLOAT32_MAX;
    desc.MaxLOD = D3D11_FLOAT32_MAX;
    desc.BorderColor[0] = 1.0f;
    desc.BorderColor[1] = 1.0f;
    desc.BorderColor[2] = 1.0f;
    desc.BorderColor[3] = 1.0f;
    desc.AddressU = D3D11_TEXTURE_ADDRESS_WRAP;
    desc.AddressV = D3D11_TEXTURE_ADDRESS_WRAP;
    desc.AddressW = D3D11_TEXTURE_ADDRESS_WRAP;
    desc.Filter = D3D11_FILTER_MIN_MAG_MIP_POINT;
    HRESULT hr = device->CreateSamplerState(&desc,
        samplerStates[static_cast<int>(SamplerState::PointWrap)].GetAddressOf());
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}
```

Wrap Addressing Mode

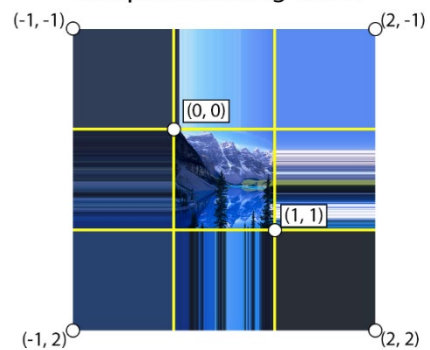


0.1 と 1.1 が同じ

```
// ポイントサンプリング&テクスチャ繰り返しなし
```

```
{
    D3D11_SAMPLER_DESC desc;
    desc.MipLODBias = 0.0f;
    desc.MaxAnisotropy = 1;
    desc.ComparisonFunc = D3D11_COMPARISON_NEVER;
    desc.MinLOD = -D3D11_FLOAT32_MAX;
    desc.MaxLOD = D3D11_FLOAT32_MAX;
    desc.BorderColor[0] = 1.0f;
    desc.BorderColor[1] = 1.0f;
    desc.BorderColor[2] = 1.0f;
    desc.BorderColor[3] = 1.0f;
    desc.AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;
    desc.AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;
    desc.AddressW = D3D11_TEXTURE_ADDRESS_CLAMP;
    desc.Filter = D3D11_FILTER_MIN_MAG_MIP_POINT;
    HRESULT hr = device->CreateSamplerState(&desc,
        samplerStates[static_cast<int>(SamplerState::PointClamp)].GetAddressOf());
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}
```

Clamp Addressing Mode

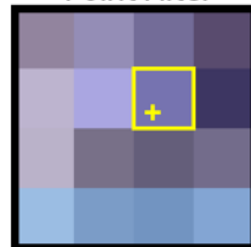


1.0 と 1.1 が同じ

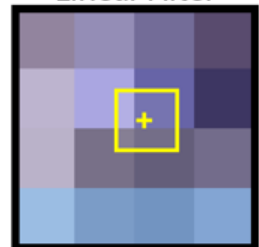
```
// リニアサンプリング&テクスチャ繰り返しあり
```

```
{
    D3D11_SAMPLER_DESC desc;
    desc.MipLODBias = 0.0f;
    desc.MaxAnisotropy = 1;
    desc.ComparisonFunc = D3D11_COMPARISON_NEVER;
    desc.MinLOD = -D3D11_FLOAT32_MAX;
    desc.MaxLOD = D3D11_FLOAT32_MAX;
    desc.BorderColor[0] = 1.0f;
    desc.BorderColor[1] = 1.0f;
    desc.BorderColor[2] = 1.0f;
    desc.BorderColor[3] = 1.0f;
    desc.AddressU = D3D11_TEXTURE_ADDRESS_WRAP;
    desc.AddressV = D3D11_TEXTURE_ADDRESS_WRAP;
    desc.AddressW = D3D11_TEXTURE_ADDRESS_WRAP;
    desc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
}
```

Point Filter



Linear Filter



描画エンジン開発 EX

```
HRESULT hr = device->CreateSamplerState(&desc,
    samplerStates[static_cast<int>(SamplerState::LinearWrap)].GetAddressOf());
_ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}
// リニアサンプリング&テクスチャ繰り返しなし
{
    D3D11_SAMPLER_DESC desc;
    desc.MipLODBias = 0.0f;
    desc.MaxAnisotropy = 1;
    desc.ComparisonFunc = D3D11_COMPARISON_NEVER;
    desc.MinLOD = -D3D11_FLOAT32_MAX;
    desc.MaxLOD = D3D11_FLOAT32_MAX;
    desc.BorderColor[0] = 1.0f;
    desc.BorderColor[1] = 1.0f;
    desc.BorderColor[2] = 1.0f;
    desc.BorderColor[3] = 1.0f;
    desc.AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;
    desc.AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;
    desc.AddressW = D3D11_TEXTURE_ADDRESS_CLAMP;
    desc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
    HRESULT hr = device->CreateSamplerState(&desc,
    samplerStates[static_cast<int>(SamplerState::LinearClamp)].GetAddressOf());
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
}
}
```

拡大表示したとき

Point Filter

Linear Filter

ガタガタだけど
くっきり
ドット絵などはこっち

滑らかになるけど
ぼやける。
基本はこっち。

Graphics.h

```
---省略---
#include "RenderState.h"
// グラフィックス
class Graphics
{
    ---省略---
public:
    ---省略---

    // レンダーステート取得
    RenderState* GetRenderState() { return renderState.get(); }

    ---省略---
private:
    ---省略---
    std::unique_ptr<RenderState> renderState;
};
```

Graphics.cpp

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

描画エンジン開発 EX

```
{
    ---省略---

    // レンダーステート生成
    renderState = std::make_unique<RenderState>(device.Get());
}
```

Scene.cpp

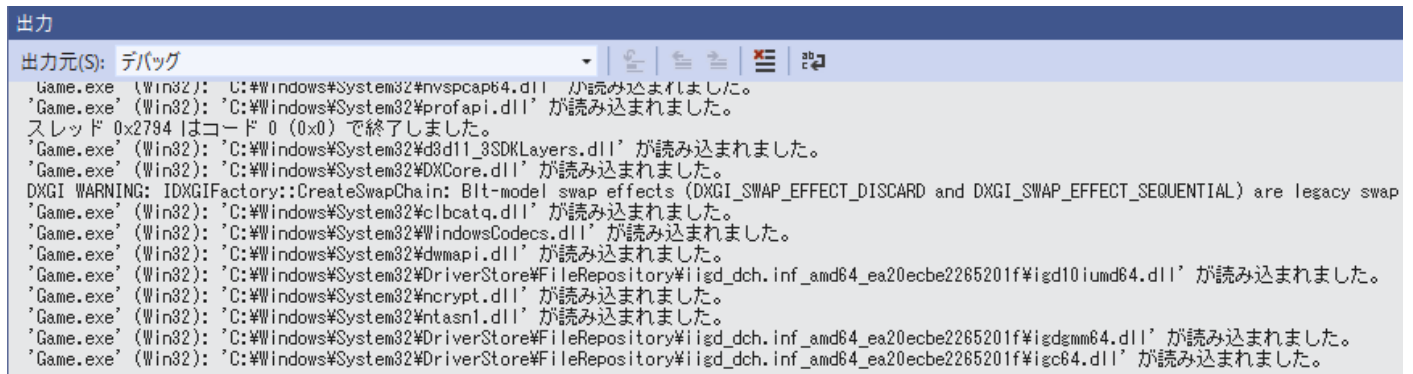
```
---省略---

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

    ID3D11SamplerState* samplers[] =
    {
        renderState->GetSamplerState(SamplerState::PointClamp)
    };
    dc->PSSetSamplers(0, _countof(samplers), samplers);

    ---省略---
}
```

プログラムを実行し、Visual Studio の出力ウィンドウを確認してみましょう。
出力ウィンドウにエラーが表示されていなければ OK です。



```
出力
出力元(S): デバッグ
Game.exe (Win32): C:\Windows\System32\nvscap64.dll が読み込まれました。
'Game.exe' (Win32): 'C:\Windows\System32\profapi.dll' が読み込まれました。
スレッド 0x2794 はコード 0 (0x0) で終了しました。
'Game.exe' (Win32): 'C:\Windows\System32\d3d11_SDKLayers.dll' が読み込まれました。
'Game.exe' (Win32): 'C:\Windows\System32\DXCore.dll' が読み込まれました。
DXGI WARNING: IDXGIFactory::CreateSwapChain: Bit-model swap effects (DXGI_SWAP_EFFECT_DISCARD and DXGI_SWAP_EFFECT_SEQUENTIAL) are legacy swap
'Game.exe' (Win32): 'C:\Windows\System32\clbcatq.dll' が読み込まれました。
'Game.exe' (Win32): 'C:\Windows\System32\WindowsCodecs.dll' が読み込まれました。
'Game.exe' (Win32): 'C:\Windows\System32\dwmapi.dll' が読み込まれました。
'Game.exe' (Win32): 'C:\Windows\System32\DriverStore\FileRepository\iigd_dch.inf_amd64_ea20ecbe2265201f\igdl0iumd64.dll' が読み込まれました。
'Game.exe' (Win32): 'C:\Windows\System32\ncrypt.dll' が読み込まれました。
'Game.exe' (Win32): 'C:\Windows\System32\ntasn1.dll' が読み込まれました。
'Game.exe' (Win32): 'C:\Windows\System32\DriverStore\FileRepository\iigd_dch.inf_amd64_ea20ecbe2265201f\igdgmm64.dll' が読み込まれました。
'Game.exe' (Win32): 'C:\Windows\System32\DriverStore\FileRepository\iigd_dch.inf_amd64_ea20ecbe2265201f\igc64.dll' が読み込まれました。
```

今回の課題ではアドレスモードやフィルタの違いをテストしませんが、重要な要素なので各自で実装し、確認してみてください。