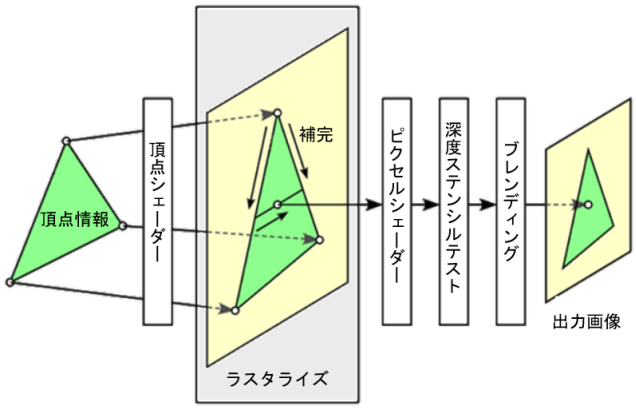
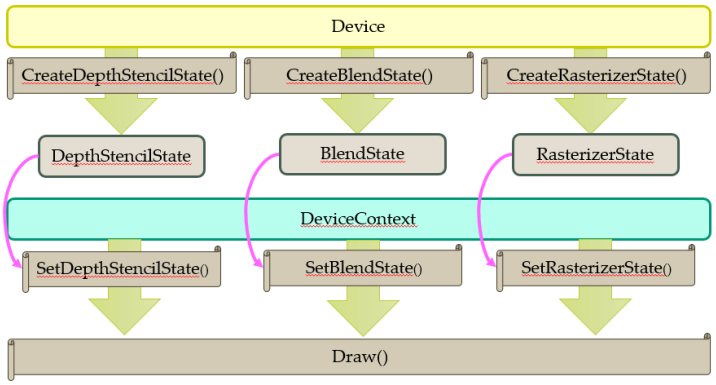
描画エンジン開発 EX

○概要

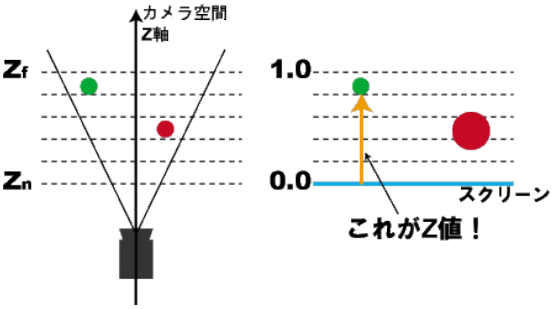
描画命令を実行してから画面に色が出力するまでに様々な工程があります。
様々な描画設定について知り、描画パイプラインを理解しましょう。

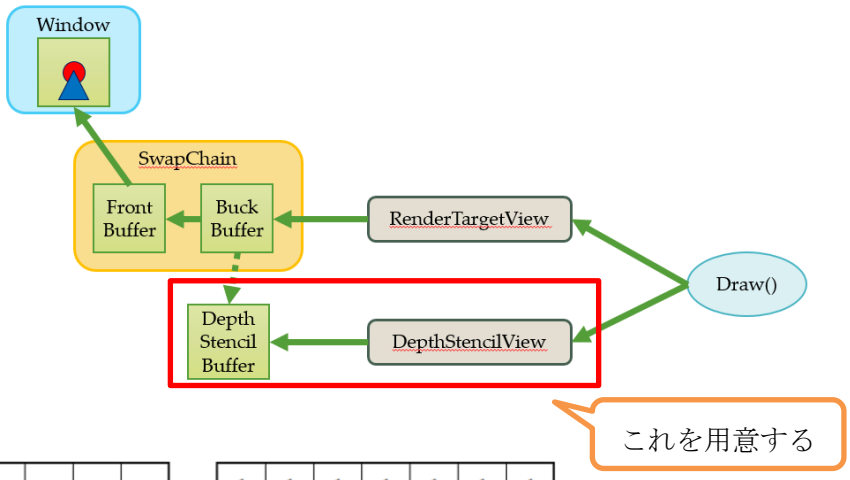




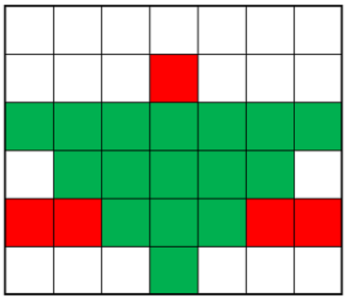
○深度ステンシルバッファ

ポリゴンの表示が重なった場合にどちらが前面に表示するか制御する必要が出てきます。
「色」はスワップチェーンに存在するバックバッファに書き込むように実装しました。
重なったポリゴンの表示順の制御をするためには「深度」を書き込むテクスチャを用意する必要があります。深度値は 0.0～1.0 で表現されることが多く、一般的に 0.0 が一番手前で 1.0 が一番奥となります。





表示結果



カラーバッファ

1	1	1	1	1	1	1
1	1	1	0.5	1	1	1
0.2	0.2	0.2	0.2	0.2	0.2	0.2
1	0.2	0.2	0.2	0.2	0.2	1
0.5	0.5	0.2	0.2	0.2	0.5	0.5
1	1	1	0.2	1	1	1

深度バッファ

FrameBuffer.h

```
---省略---

class FrameBuffer
{
    ---省略---

private:
    ---省略---
    Microsoft::WRL::ComPtr<ID3D11DepthStencilView>        depthStencilView;
};
```

FrameBuffer.cpp

```
---省略---

// コンストラクタ
FrameBuffer::FrameBuffer(ID3D11Device* device, IDXGISwapChain* swapchain)
{
    ---省略---

    // 深度ステンシルビューの生成
    {
        // 深度ステンシル情報を書き込むためのテクスチャを作成する。
        Microsoft::WRL::ComPtr<ID3D11Texture2D> texture2d;
        D3D11_TEXTURE2D_DESC texture2dDesc;
        texture2dDesc.Width = width;
        texture2dDesc.Height = height;
        texture2dDesc.MipLevels = 1;
        texture2dDesc.ArraySize = 1;
        texture2dDesc.Format = DXGI_FORMAT_D24_UNORM_S8_UINT;
        texture2dDesc.SampleDesc.Count = 1;
        texture2dDesc.SampleDesc.Quality = 0;
        texture2dDesc.Usage = D3D11_USAGE_DEFAULT;
        texture2dDesc.BindFlags = D3D11_BIND_DEPTH_STENCIL;
        texture2dDesc.CPUAccessFlags = 0;
        texture2dDesc.MiscFlags = 0;
        hr = device->CreateTexture2D(&texture2dDesc, nullptr, texture2d.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

        // 深度ステンシルテクスチャへの書き込みに窓口になる深度ステンシルビューを作成する。
        hr = device->CreateDepthStencilView(texture2d.Get(), nullptr, depthStencilView.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
}

// クリア
void FrameBuffer::Clear(ID3D11DeviceContext* dc, float r, float g, float b, float a)
{
    ---省略---
    dc->ClearDepthStencilView(depthStencilView.Get(), D3D11_CLEAR_DEPTH | D3D11_CLEAR_STENCIL, 1.0f, 0);
}

// レンダーターゲット設定
void FrameBuffer::SetRenderTargets(ID3D11DeviceContext* dc)
```

深度値は 1.0 にクリアする

深度ステンシルビューを通して
深度テクスチャに深度を書き込む

```
{
    // ビューポート＆レンダーターゲットを設定
    dc->RSSetViewports(1, &viewport);
    dc->OMSetRenderTargets(1, renderTargetView.GetAddressOf(), depthStencilView.Get());
}
```

○デプスステンシルステート

深度テクスチャに書き込まれた深度値とこれから描くポリゴンの深度値を比較して表示順の制御を行います。この制御はデプスステンシルステートの設定によって行われます。
描画設定関連を管理する RenderState クラスにデプスステンシルステートに追加し、取得できるようにしましょう。

RenderState.h

```
---省略---

// デプスステート
enum class DepthState
{
    TestAndWrite,
    TestOnly,
    WriteOnly,
    NoTestNoWrite,

    EnumCount
};

// レンダーステート
class RenderState
{
public:
    ---省略---

    // デプスステート取得
    ID3D11DepthStencilState* GetDepthStencilState(DepthState state) const
    {
        return depthStencilStates[static_cast<int>(state)].Get();
    }

private:
    ---省略---
  Microsoft::WRL::ComPtr<ID3D11DepthStencilState> depthStencilStates[static_cast<int>(DepthState::EnumCount)];
};
```

RenderState.cpp

```
---省略---

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

```
    ---省略---

    // 深度テストあり＆深度書き込みあり
    {
        D3D11_DEPTH_STENCIL_DESC desc{};
        desc.DepthEnable = true;
        desc.DepthWriteMask = D3D11_DEPTH_WRITE_MASK_ALL;
        desc.DepthFunc = D3D11_COMPARISON_LESS_EQUAL;
        HRESULT hr = device->CreateDepthStencilState(&desc,
                            depthStencilStates[static_cast<int>(DepthState::TestAndWrite)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // 深度テストあり＆深度書き込みなし
    {
        D3D11_DEPTH_STENCIL_DESC desc{};
        desc.DepthEnable = true;
        desc.DepthWriteMask = D3D11_DEPTH_WRITE_MASK_ZERO;
        desc.DepthFunc = D3D11_COMPARISON_LESS_EQUAL;
        HRESULT hr = device->CreateDepthStencilState(&desc,
                            depthStencilStates[static_cast<int>(DepthState::TestOnly)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // 深度テストなし＆深度書き込みあり
    {
        D3D11_DEPTH_STENCIL_DESC desc{};
        desc.DepthEnable = true;
        desc.DepthWriteMask = D3D11_DEPTH_WRITE_MASK_ALL;
        desc.DepthFunc = D3D11_COMPARISON_ALWAYS;
        HRESULT hr = device->CreateDepthStencilState(&desc,
                            depthStencilStates[static_cast<int>(DepthState::WriteOnly)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // 深度テストなし＆深度書き込みなし
    {
        D3D11_DEPTH_STENCIL_DESC desc{};
        desc.DepthEnable = false;
        desc.DepthWriteMask = D3D11_DEPTH_WRITE_MASK_ZERO;
        desc.DepthFunc = D3D11_COMPARISON_ALWAYS;
        HRESULT hr = device->CreateDepthStencilState(&desc,
                           depthStencilStates[static_cast<int>(DepthState::NoTestNoWrite)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
}
```

Sprite クラスを改造し、奥行を表現する深度値を書き込めるようにします。

Sprite.h

```
---省略---

// スプライト
class Sprite
{
public:
```

```
    ---省略---

    // 描画実行
    void Render(ID3D11DeviceContext* dc,
        float dx, float dy,                 // 左上位置
        float dz,                           // 奥行
        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 dz,                           // 奥行
        float dw, float dh,                 // 幅、高さ
        float angle,                        // 角度
        float r, float g, float b, float a // 色
    ) const;

    ---省略---
};
```

奥行を追加

奥行を追加

Sprite.cpp

```
---省略---

// 描画実行
void Sprite::Render(ID3D11DeviceContext* dc,
    float dx, float dy,                     // 左上位置
    float dz,                               // 奥行
    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---

    // 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].position.z = dz;

        ---省略---
    }

    ---省略---
}
```

深度値を設定

```
// 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 dz,                           // 奥行
    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, dz, dw, dh, 0, 0, textureWidth, textureHeight, angle, r, g, b, a);
}
```

Scene.h

```
---省略---

// 深度テストシーン
class DepthTestScene : public Scene
{
public:
    DepthTestScene();
    ~DepthTestScene() override = default;

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

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

Scene.cpp

```
---省略---

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

    sprites[0]->Render(dc, 100, 50, 0, width, height, 10 * width, 3 * height, width, height, 0, 1, 1, 1, 1);
    sprites[0]->Render(dc, 100, 350, 0, 480, 300, 0, 1, 1, 1, 1);
    sprites[1]->Render(dc, 300, 50, 0, width, height, 0, 0, 1, 0, 1);
}

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

    sprite = std::make_unique<Sprite>(device);
}

// 描画処理
```

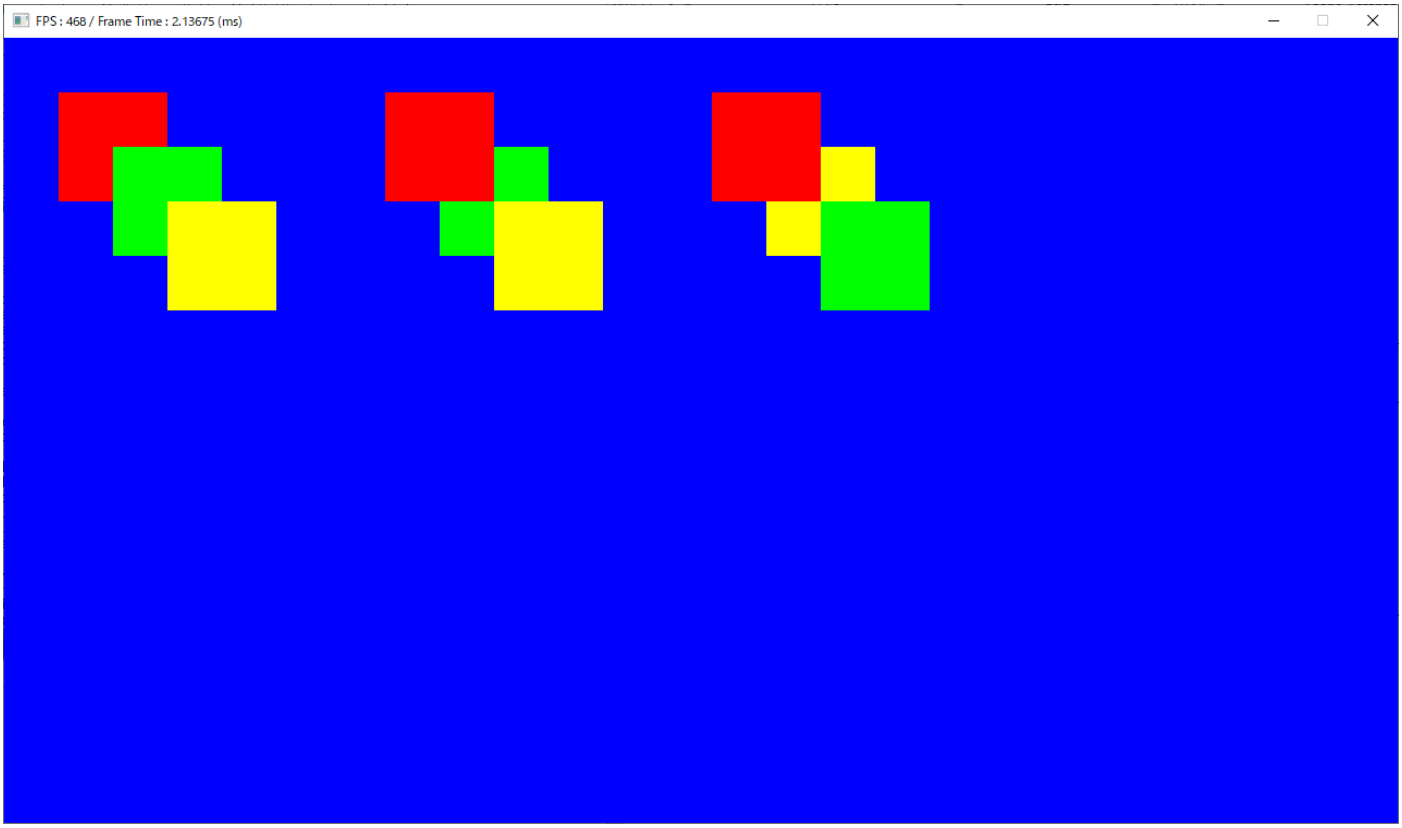
引数が増えた対応

```
void DepthTestScene::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);

    // 深度書き込みなし＆深度テストなし
    dc->OMSetDepthStencilState(renderState->GetDepthStencilState(DepthState::NoTestNoWrite), 0);
    sprite->Render(dc,  50,  50, 0.0f, 100, 100, 0, 1, 0, 0, 0);
    sprite->Render(dc, 100, 100, 1.0f, 100, 100, 0, 0, 1, 0, 0);
    sprite->Render(dc, 150, 150, 0.5f, 100, 100, 0, 1, 1, 0, 0);

    // 深度書き込みあり＆深度テストあり
    dc->OMSetDepthStencilState(renderState->GetDepthStencilState(DepthState::TestAndWrite), 0);
    sprite->Render(dc, 350,  50, 0.0f, 100, 100, 0, 1, 0, 0, 0);
    sprite->Render(dc, 400, 100, 1.0f, 100, 100, 0, 0, 1, 0, 0);
    sprite->Render(dc, 450, 150, 0.5f, 100, 100, 0, 1, 1, 0, 0);

    // 深度書き込みのみ
    dc->OMSetDepthStencilState(renderState->GetDepthStencilState(DepthState::WriteOnly), 0);
    sprite->Render(dc, 650,  50, 0.0f, 100, 100, 0, 1, 0, 0, 0);
    // 深度テストのみ
    dc->OMSetDepthStencilState(renderState->GetDepthStencilState(DepthState::TestOnly), 0);
    sprite->Render(dc, 700, 100, 0.5f, 100, 100, 0, 1, 1, 0, 0);
    // 深度書き込みのみ
    dc->OMSetDepthStencilState(renderState->GetDepthStencilState(DepthState::WriteOnly), 0);
    sprite->Render(dc, 750, 150, 1.0f, 100, 100, 0, 0, 1, 0, 0);
}
```

Framework.cpp

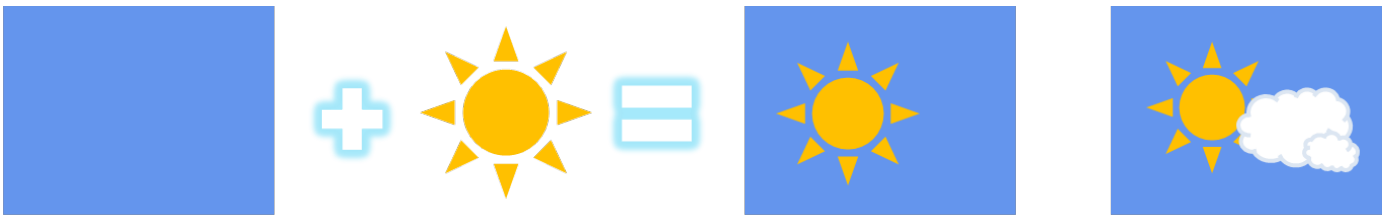
```
---省略---

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

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

実行確認してみましょう。
デプスステンシルステートの設定と深度値の組み合わせから意図通りの描画順序になっていれば OK です。



○ブレンドステート

ポリゴンを描く際にブレンドステートを設定することにより、重なった部分の色の合成制御ができます。

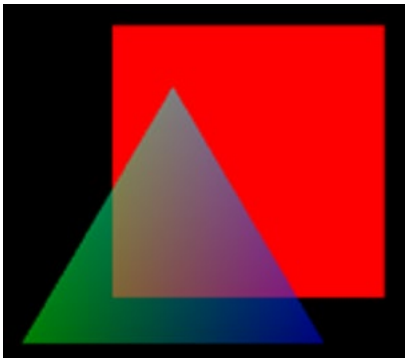


D3D11_BLEND	
D3D11_BLEND_SRC_COLOR	これから描画する画の色(R, G, B)
D3D11_BLEND_INV_SRC_COLOR	これから描画する画の色の反転(1.0 – R, 1.0 – G, 1.0 – B)
D3D11_BLEND_SRC_ALPHA	これから描画する画の透明値(A)
D3D11_BLEND_INV_SRC_ALPHA	これから描画する画の透明値の反転(1.0 – A)
D3D11_BLEND_DEST_ALPHA	既に描画されている画の透明値(A)
D3D11_BLEND_INV_DEST_ALPHA	既に描画されている画の透明値の反転(1.0 – A)
D3D11_BLEND_DEST_COLOR	既に描画されている画の色(R, G, B)

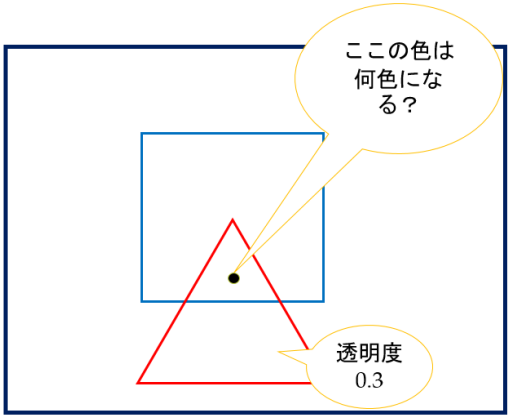
D3D11_BLEND_INV_DEST_COLOR	既に描画されている画の色の反転(1.0 – R, 1.0 – G, 1.0 – B)

D3D11_BLEND_OP	
D3D11_BLEND_OP_ADD	Destination + Source
D3D11_BLEND_OP_SUBTRACT	Source - Destination
D3D11_BLEND_OP_REV_SUBTRACT	Destination - Source

描画済みの色（Destination）と新たに描画する色（Source）の設定の組み合わせで様々な色のブレンディングができます。
一番よく使われるアルファブレンディングについて説明します。

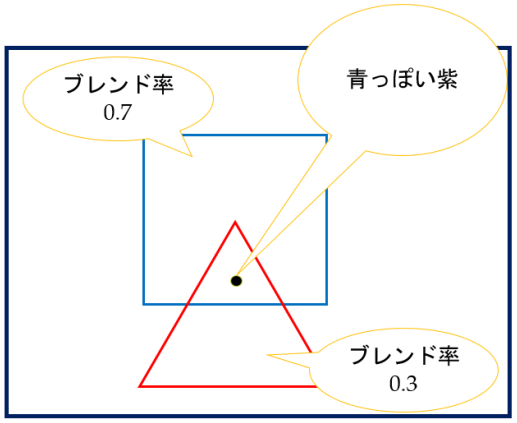


半透明のポリゴンを描いた場合、
下に描かれている色が透けるように
色を合成する表現



青色の四角形（不透明）の上に
赤色の三角形（半透明）を描くと
三角形の透明度の強さに合わせて
色を合成したい

三角形の色の強さは三角形の透明値
四角形の色の強さは 1.0－三角形の透明値



- SrcBlend = BLEND_SRC_ALPHA;
- DestBlend = BLEND_INV_SRC_ALPHA;
- BlendOp = BLEND_OP_ADD;

```
    R     G     B
 (0.0,  0.0,  1.0)×0.7  Dest
+(1.0,  0.0,  0.0)×0.3  Src
-----------------------
 (0.3,  0.0,  0.7)
```

RenderState クラスにブレンドステートを追加し、取得できるようにしましょう。

RenderState.h

```
---省略---

// ブレンドステート
enum class BlendState
{
    Opaque,
    Transparency,
    Additive,
    Subtraction,
    Multiply,

    EnumCount
};

// レンダーステート
class RenderState
{
public:
    ---省略---

    // ブレンドステート取得
    ID3D11BlendState* GetBlendState(BlendState state) const
    {
        return blendStates[static_cast<int>(state)].Get();
    }

private:
    ---省略---
    Microsoft::WRL::ComPtr<ID3D11BlendState>    blendStates[static_cast<int>(BlendState::EnumCount)];
};
```

RenderState.cpp

```
---省略---

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

    // 合成なし
    {
        D3D11_BLEND_DESC desc{};
        desc.AlphaToCoverageEnable = false;
        desc.IndependentBlendEnable = false;
        desc.RenderTarget[0].BlendEnable = false;
        desc.RenderTarget[0].SrcBlend = D3D11_BLEND_SRC_ALPHA;
        desc.RenderTarget[0].DestBlend = D3D11_BLEND_INV_SRC_ALPHA;
        desc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_ADD;
```

```
        desc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
        desc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
        desc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
        desc.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;
        HRESULT hr = device->CreateBlendState(&desc,
                                blendStates[static_cast<int>(BlendState::Opaque)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // 通常合成
    {
        D3D11_BLEND_DESC desc{};
        desc.AlphaToCoverageEnable = false;
        desc.IndependentBlendEnable = false;
        desc.RenderTarget[0].BlendEnable = true;
        desc.RenderTarget[0].SrcBlend = D3D11_BLEND_SRC_ALPHA;
        desc.RenderTarget[0].DestBlend = D3D11_BLEND_INV_SRC_ALPHA;
        desc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_ADD;
        desc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
        desc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
        desc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
        desc.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;
        HRESULT hr = device->CreateBlendState(&desc,
                                blendStates[static_cast<int>(BlendState::Transparency)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // 加算合成
    {
        D3D11_BLEND_DESC desc{};
        desc.AlphaToCoverageEnable = false;
        desc.IndependentBlendEnable = false;
        desc.RenderTarget[0].BlendEnable = true;
        desc.RenderTarget[0].SrcBlend = D3D11_BLEND_SRC_ALPHA;
        desc.RenderTarget[0].DestBlend = D3D11_BLEND_ONE;
        desc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_ADD;
        desc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
        desc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
        desc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
        desc.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;
        HRESULT hr = device->CreateBlendState(&desc,
                                blendStates[static_cast<int>(BlendState::Additive)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // 減算合成
    {
        D3D11_BLEND_DESC desc{};
        desc.AlphaToCoverageEnable = false;
        desc.IndependentBlendEnable = false;
        desc.RenderTarget[0].BlendEnable = true;
        desc.RenderTarget[0].SrcBlend = D3D11_BLEND_SRC_ALPHA;
        desc.RenderTarget[0].DestBlend = D3D11_BLEND_ONE;
        desc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_REV_SUBTRACT;
        desc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
        desc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
        desc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
        desc.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;
        HRESULT hr = device->CreateBlendState(&desc,
                                blendStates[static_cast<int>(BlendState::Subtraction)].GetAddressOf());
```

```
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // 乗算合成
    {
        D3D11_BLEND_DESC desc{};
        desc.AlphaToCoverageEnable = false;
        desc.IndependentBlendEnable = false;
        desc.RenderTarget[0].BlendEnable = true;
        desc.RenderTarget[0].SrcBlend = D3D11_BLEND_ZERO;
        desc.RenderTarget[0].DestBlend = D3D11_BLEND_SRC_COLOR;
        desc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_ADD;
        desc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
        desc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
        desc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
        desc.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;
        HRESULT hr = device->CreateBlendState(&desc,
                                blendStates[static_cast<int>(BlendState::Multiply)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
}
```

Scene.h

```
---省略---

// ブレンドテストシーン
class BlendTestScene : public Scene
{
public:
    BlendTestScene();
    ~BlendTestScene() 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
BlendTestScene::BlendTestScene()
{
    ID3D11Device* device = Graphics::Instance().GetDevice();

    sprite = std::make_unique<Sprite>(device);
}

// 描画処理
void BlendTestScene::Render(float elapsedTime)
```

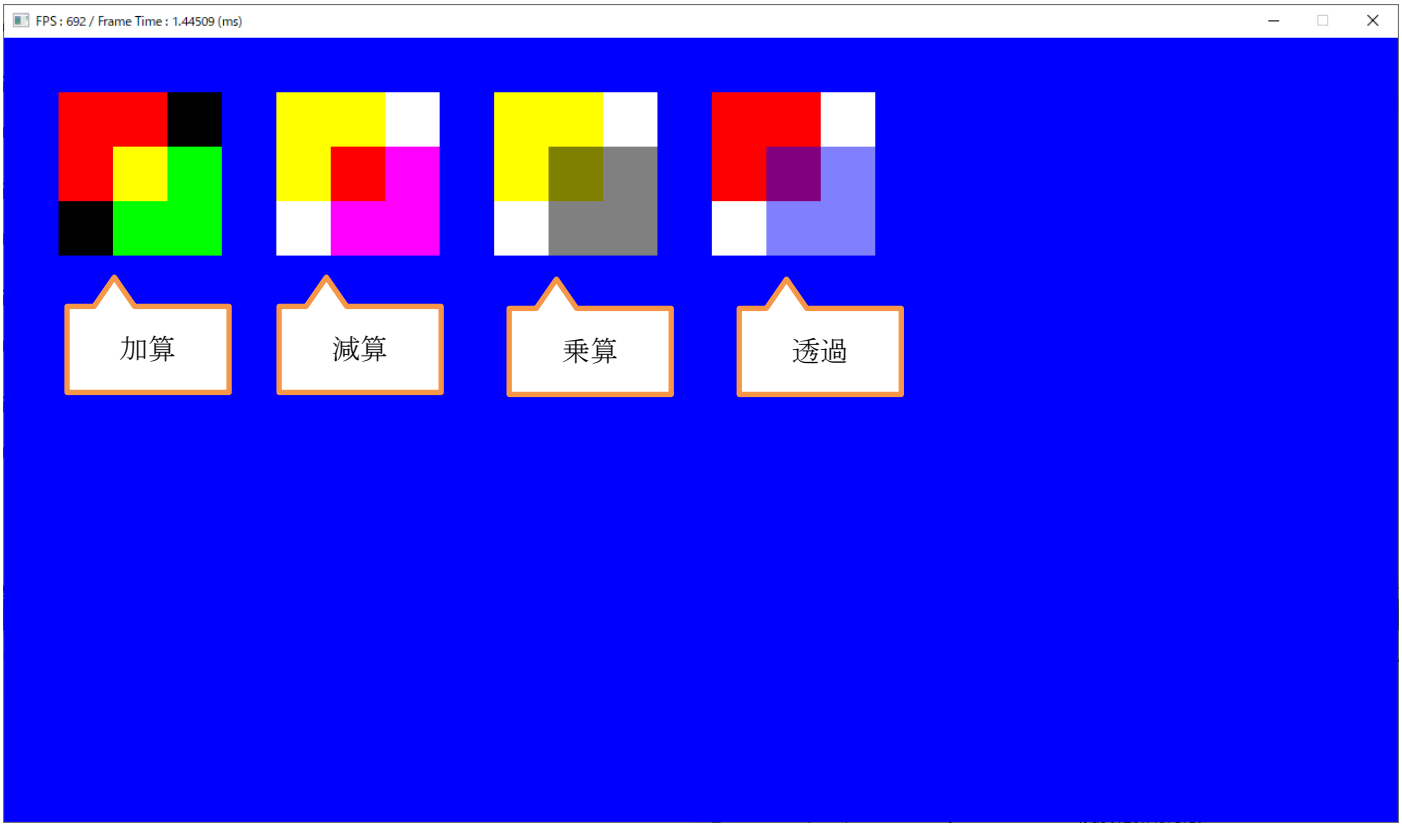
```
{
    ID3D11DeviceContext* dc = Graphics::Instance().GetDeviceContext();
    RenderState* renderState = Graphics::Instance().GetRenderState();

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

    // 深度テストなし＆深度書き込みなし（後に描いたものが手前になる）
    dc->OMSetDepthStencilState(renderState->GetDepthStencilState(DepthState::NoTestNoWrite), 0);

    FLOAT blendFactor[4] = { 1.0f, 1.0f, 1.0f, 1.0f };
    UINT sampleMask = 0xFFFFFFFF;

    // 加算合成テスト
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Opaque), blendFactor, sampleMask);
    sprite->Render(dc, 50, 50, 0.0f, 150, 150, 0, 0, 0, 0, 1.0f);
    sprite->Render(dc, 50, 50, 0.0f, 100, 100, 0, 1, 0, 0, 1.0f);
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Additive), blendFactor, sampleMask);
    sprite->Render(dc, 100, 100, 0.0f, 100, 100, 0, 0, 1, 0, 1.0f);

    // 減算合成テスト
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Opaque), blendFactor, sampleMask);
    sprite->Render(dc, 250, 50, 0.0f, 150, 150, 0, 1, 1, 1, 1.0f);
    sprite->Render(dc, 250, 50, 0.0f, 100, 100, 0, 1, 1, 0, 1.0f);
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Subtraction), blendFactor, sampleMask);
    sprite->Render(dc, 300, 100, 0.0f, 100, 100, 0, 0, 1, 0, 1.0f);

    // 乗算合成テスト
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Opaque), blendFactor, sampleMask);
    sprite->Render(dc, 450, 50, 0.0f, 150, 150, 0, 1, 1, 1, 1.0f);
    sprite->Render(dc, 450, 50, 0.0f, 100, 100, 0, 1, 1, 0, 1.0f);
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Subtraction), blendFactor, sampleMask);
    sprite->Render(dc, 500, 100, 0.0f, 100, 100, 0, 0.5f, 0.5f, 0.5f, 1.0f);

    // 透明合成テスト
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Opaque), blendFactor, sampleMask);
    sprite->Render(dc, 650, 50, 0.0f, 150, 150, 0, 1, 1, 1, 0.5f);
    sprite->Render(dc, 650, 50, 0.0f, 100, 100, 0, 1, 0, 0, 0.5f);
    dc->OMSetBlendState(renderState->GetBlendState(BlendState::Transparency), blendFactor, sampleMask);
    sprite->Render(dc, 700, 100, 0.0f, 100, 100, 0, 0, 0, 1, 0.5f);
}
```

Framework.cpp

```
---省略---

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

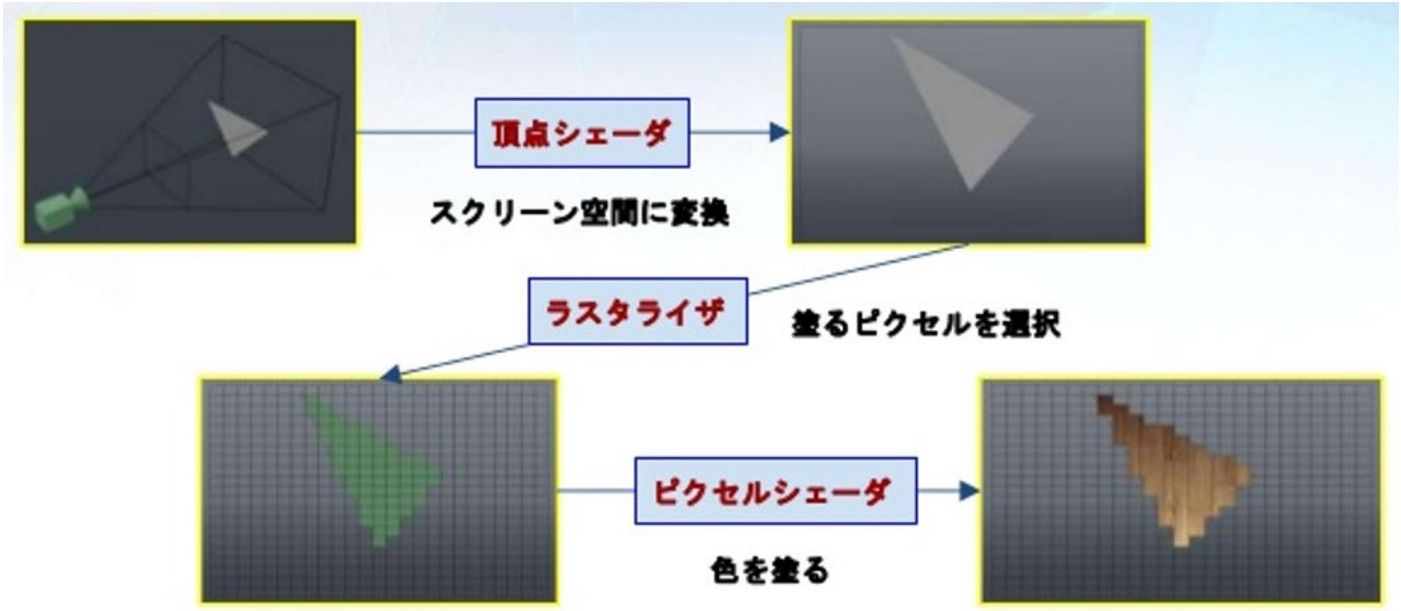
```
    // シーン初期化
    scene = std::make_unique<DepthTestScene>();
    scene = std::make_unique<BlendTestScene>();
}
```

実行確認しましょう。
意図通りに色がブレンディングできていれば OK です。



○ラスタライザステート

ポリゴンが画面内に表示される場合に書き込むテクスチャのどのピクセルに色を塗るかを決定する必要があります。
この色を塗るピクセルを決定する工程をラスタライズと呼び、ラスタライザステートを利用して描画設定を行います。



ラスタライザーステートでは以下のような設定ができます。

D3D11_FILL_MODE	説明
D3D11_FILL_WIREFRAME	ポリゴンの枠だけ色を塗る
D3D11_FILL_SOLID	ポリゴン全体の色を塗る

D3D11_CULL_MODE	説明
D3D11_CULL_NONE	ポリゴンの両面とも描く
D3D11_CULL_FRONT	ポリゴンの表面は描かない
D3D11_CULL_BACK	ポリゴンの裏面は描かない

RenderState クラスにラスタライザステートを追加し、取得できるようにしましょう。

RenderState.h

```
---省略---

// ラスタライザステート
enum class RasterizerState
{
    SolidCullNone,
    SolidCullBack,
    WireCullNone,
    WireCullBack,

    EnumCount
};

// レンダーステート
class RenderState
{
```

```
public:
    ---省略---

    // ラスタライザーステート取得
    ID3D11RasterizerState* GetRasterizerState(RasterizerState state) const
    {
        return rasterizerStates[static_cast<int>(state)].Get();
    }

private:
    ---省略---
 Microsoft::WRL::ComPtr<ID3D11RasterizerState> rasterizerStates[static_cast<int>(RasterizerState::EnumCount)];
};
```

RenderState.cpp

```
---省略---

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

    // ベタ塗り＆カリングなし
    {
        D3D11_RASTERIZER_DESC desc{};
        desc.FrontCounterClockwise = false;
        desc.DepthBias = 0;
        desc.DepthBiasClamp = 0;
        desc.SlopeScaledDepthBias = 0;
        desc.DepthClipEnable = true;
        desc.ScissorEnable = false;
        desc.MultisampleEnable = true;
        desc.FillMode = D3D11_FILL_SOLID;
        desc.CullMode = D3D11_CULL_NONE;
        desc.AntialiasedLineEnable = false;
        HRESULT hr = device->CreateRasterizerState(&desc,
                        rasterizerStates[static_cast<int>(RasterizerState::SolidCullNone)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // ベタ塗り＆裏面カリング
    {
        D3D11_RASTERIZER_DESC desc{};
        desc.FrontCounterClockwise = false;
        desc.DepthBias = 0;
        desc.DepthBiasClamp = 0;
        desc.SlopeScaledDepthBias = 0;
        desc.DepthClipEnable = true;
        desc.ScissorEnable = false;
        desc.MultisampleEnable = true;
        desc.FillMode = D3D11_FILL_SOLID;
        desc.CullMode = D3D11_CULL_BACK;
        desc.AntialiasedLineEnable = false;
        HRESULT hr = device->CreateRasterizerState(&desc,
                        rasterizerStates[static_cast<int>(RasterizerState::SolidCullBack)].GetAddressOf());
```

```
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // ワイヤーフレーム＆カリングなし
    {
        D3D11_RASTERIZER_DESC desc{};
        desc.FrontCounterClockwise = false;
        desc.DepthBias = 0;
        desc.DepthBiasClamp = 0;
        desc.SlopeScaledDepthBias = 0;
        desc.DepthClipEnable = true;
        desc.ScissorEnable = false;
        desc.MultisampleEnable = true;
        desc.FillMode = D3D11_FILL_WIREFRAME;
        desc.CullMode = D3D11_CULL_NONE;
        desc.AntialiasedLineEnable = true;
        HRESULT hr = device->CreateRasterizerState(&desc,
                        rasterizerStates[static_cast<int>(RasterizerState::WireCullNone)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
    // ワイヤーフレーム＆裏面カリング
    {
        D3D11_RASTERIZER_DESC desc{};
        desc.FrontCounterClockwise = false;
        desc.DepthBias = 0;
        desc.DepthBiasClamp = 0;
        desc.SlopeScaledDepthBias = 0;
        desc.DepthClipEnable = true;
        desc.ScissorEnable = false;
        desc.MultisampleEnable = true;
        desc.FillMode = D3D11_FILL_WIREFRAME;
        desc.CullMode = D3D11_CULL_BACK;
        desc.AntialiasedLineEnable = true;
        HRESULT hr = device->CreateRasterizerState(&desc,
                        rasterizerStates[static_cast<int>(RasterizerState::WireCullBack)].GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
}
```

Scene.h

```
---省略---

// ラスタライズテストシーン
class RasterizeTestScene : public Scene
{
public:
    RasterizeTestScene();
    ~RasterizeTestScene() 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
RasterizeTestScene::RasterizeTestScene()
{
    ID3D11Device* device = Graphics::Instance().GetDevice();

    sprite = std::make_unique<Sprite>(device);
}

// 描画処理
void RasterizeTestScene::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);

    // 深度テストなし＆深度書き込みなし（後に描いたものが手前になる）
    dc->OMSetDepthStencilState(renderState->GetDepthStencilState(DepthState::NoTestNoWrite), 0);

    // ベタ塗り＆カリングなし
    dc->RSSetState(renderState->GetRasterizerState(RasterizerState::SolidCullNone));
    sprite->Render(dc,  50,  50, 0.0f,  100, 100, 0, 1, 0, 0, 1);
    sprite->Render(dc, 150, 200, 0.0f, -100, 100, 0, 1, 0, 0, 1);

    // ベタ塗り＆裏面カリング
    dc->RSSetState(renderState->GetRasterizerState(RasterizerState::SolidCullBack));
    sprite->Render(dc, 200,  50, 0.0f,  100, 100, 0, 1, 0, 0, 1);
    sprite->Render(dc, 300, 200, 0.0f, -100, 100, 0, 1, 0, 0, 1);

    // ワイヤーフレーム＆カリングなし
    dc->RSSetState(renderState->GetRasterizerState(RasterizerState::WireCullNone));
    sprite->Render(dc, 350,  50, 0.0f,  100, 100, 0, 1, 0, 0, 1);
    sprite->Render(dc, 450, 200, 0.0f, -100, 100, 0, 1, 0, 0, 1);

    // ワイヤーフレーム＆裏面カリング
    dc->RSSetState(renderState->GetRasterizerState(RasterizerState::WireCullBack));
    sprite->Render(dc, 500,  50, 0.0f,  100, 100, 0, 1, 0, 0, 1);
    sprite->Render(dc, 600, 200, 0.0f, -100, 100, 0, 1, 0, 0, 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<BlendTestScene>();
    scene = std::make_unique<RasterizeTestScene>();
}
```

実行確認してみましょう。
色の塗り方やカリングの設定について理解できれば OK です。