作品概要

ASOポップカルチャー専門学校
ゲーム・CG・アニメ専攻　ゲーム専攻

永冨 心

機人

k i z i n

作品概要

３Dアクションゲームで、敵を殲滅したら勝利というゲームです。操作を簡単にしてだれでも楽しめるようにチームで制作していきました。

担当箇所

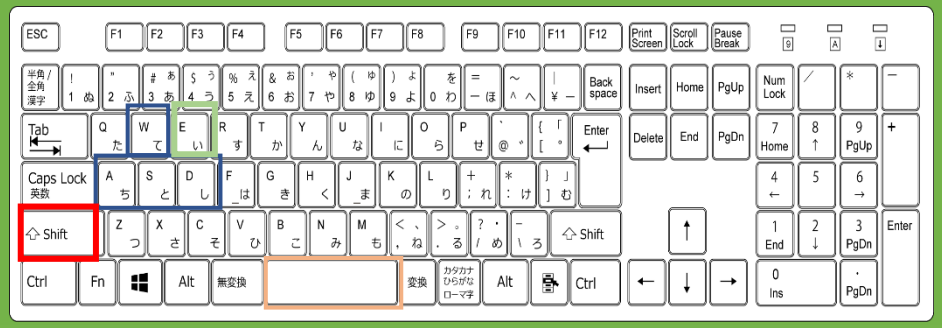
主にグラフィック周りとカメラ制御を担当しました。
グラフィック周りの処理では、ポストエフェクトや影表現をしていきました。
モデルに対して、ディレクションライトを使って、影をつけたりモデルの影を落としたりしました。
ポストエフェクトについては、被写界深度の実装をしています。

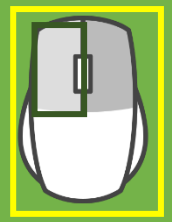
製作時期	３年後期
製作期間	４か月
チーム人数	５人
使用言語	C++/C#/HLSL
使用ライブラリ	DXライブラリ

動画QR

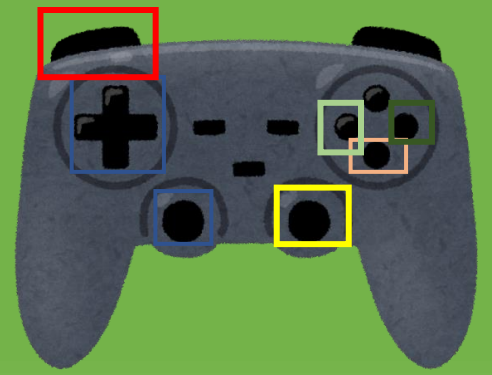
操作説明

キーボード＋マウス





コントローラー



移動	ジャンプ	必殺技	攻撃	ダッシュ	視点操作

担当箇所

・シェーダー周り全般

・ライティングの制御
Shader/Pixel/のすべて
Shader/Vertex/のすべて
・ポストエフェクトの描画処理
Shader/PostEffect/のすべて
Shader/の中のcppとhすべて
・シャドウマップの作成
Shader/ShadowMapのすべて
Scene/のシャドウマップ作成
処理を担当

・その他

・キーコンフィグの実装
Common/Input/のすべて
・カメラ制御
Common/Camera.cppと.h
・レーダーマップの実装
UI/MiniMap.cppと.h
・レンダーの１部処理を担当
Component/Render/の
１部処理を担当

ライティングの制御

・実装しようと思った理由

ライティングを実装しようと考えたのは、ロボットアクションゲームを作る予定だったので、リアル感を出すために実装しようと考えました。

・参考資料

HLSLの魔導書を参考にライティングの実装をしました。

・ライティングの実装

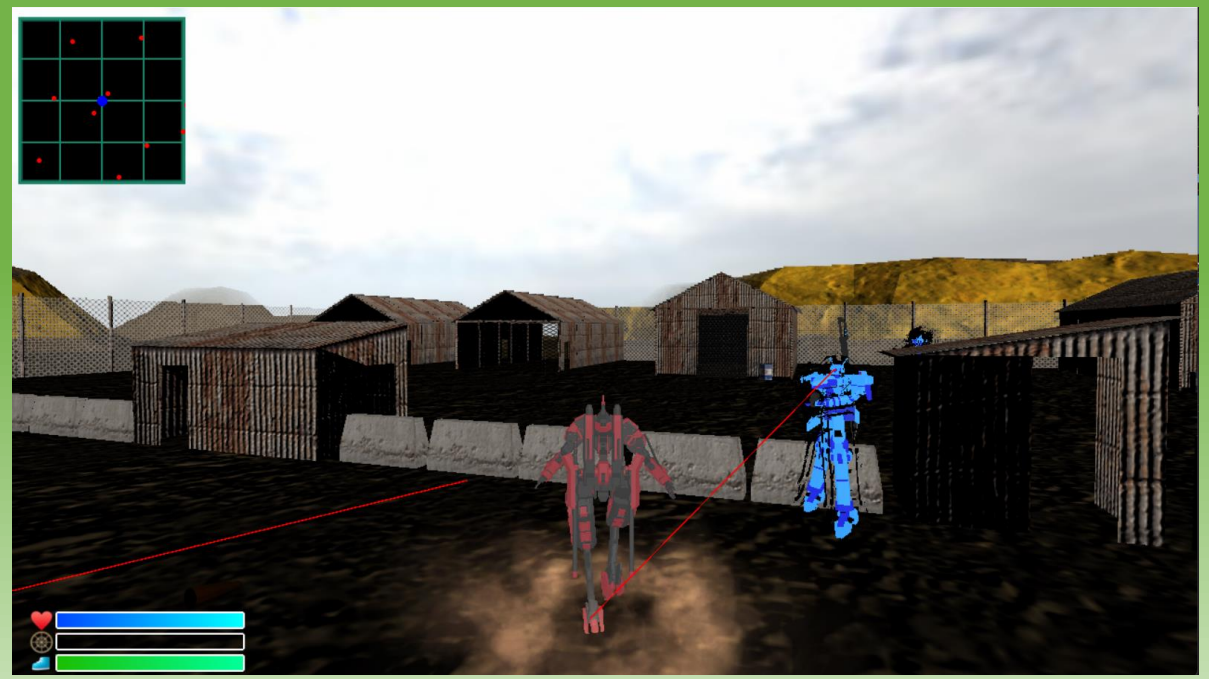
ディレクションライトを使って拡散反射と鏡面反射を実装しました。
実装する中でモデルが法線しかないモデルだったので、従法線と接線を計算しました。(＊1)

(＊1)

```
// 従法線と接線を求める
float3 dp1 = ddx(input.worldpos.xyz);
float3 dp2 = ddy(input.worldpos.xyz);
float2 duv1 = ddx(input.uv.rb);
float2 duv2 = ddy(input.uv.rb);
float3x3 M = float3x3(dp1, dp2, cross(dp1, dp2));
float3x3 UV = float3x3(float3(input.uv.rb, 1.0f), float3(duv1, 0.0f), float3(duv2, 0.0f));
float3x3 inverseM = float3x3(Invert(M));
float3x3 derivative = -inverseM * UV;
float3 tan = normalize(derivative[1]);
float3 bin = normalize(derivative[2]);
float3 normal = normalize(derivative[0]);

// 法線を計算
input.norm = CalcNormal(normal, tan, bin, input.uv.rb);
```

ライティングの制御

実装前



実装後

シャドウマップの作成

・実装しようと思った理由

リアル感を出すときに影がないと微妙という意見を受けて実装しました。

・参考資料

HLSLの魔導書を参考にライティングの実装をしました。

・影の実装

Z値を色として出力します。この時に透明にならないように
α値は必ず1にします。(*1)

そこで出した情報をモデルに対してライティングを
実装するときに渡して(*2)のようにします。
この時に、DXライブラリを使用していてsampler
にしか対応していなかったのでこのようにして
影を実装しました。

(*1)

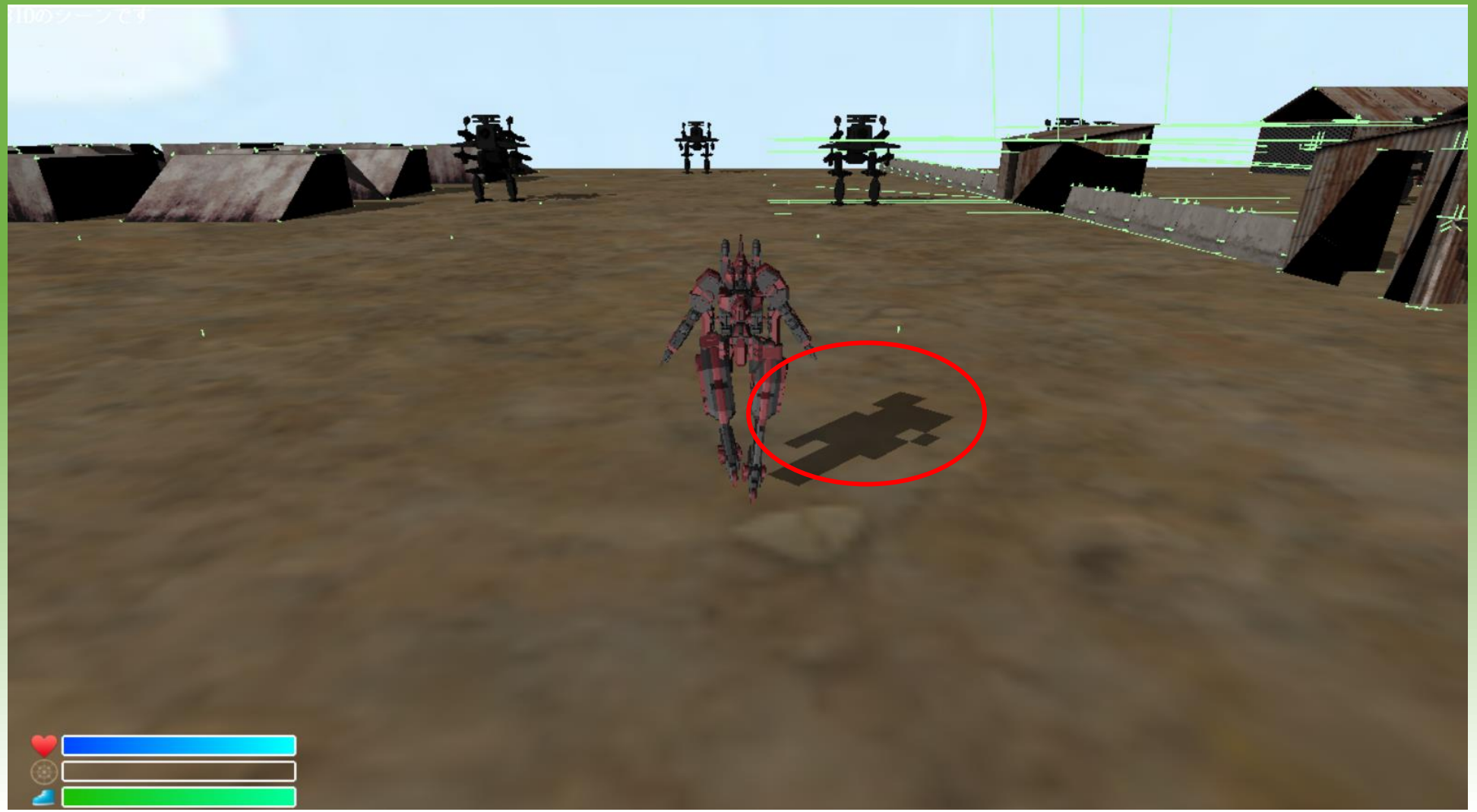
```
{
    PSOutput output;
    // Z値を色として出力
    output.color = input.vpos.z / input.vpos.w;
    // 透明にならないようにアルファは必ず1
    output.color.a = 1.0f;
    return output;
}
```

(*2)

```
float2 shadowUV;
// 深度テクスチャの座標を算出
shadowUV.x = (input.lpos.x + 1.0f) * 0.5f;
// yは上下反転しないといけない
shadowUV.y = 1.0f - (input.lpos.y + 1.0f) * 0.5f;
// マッハバンドを起こさないようにするため
input.lpos.z -= 0.005f;

// 周囲のデータと深度テクスチャの深度を取得
float comp = 0;
float U = 1.0f / 2048;
float V = 1.0f / 2048;
comp += saturate(max(input.lpos.z - depthtex.Sample(depthsam, shadowUV + float2(0, 0)).r, 0.0f) * 1500 - 0.5f);
comp += saturate(max(input.lpos.z - depthtex.Sample(depthsam, shadowUV + float2(U, 0)).r, 0.0f) * 1500 - 0.5f);
comp += saturate(max(input.lpos.z - depthtex.Sample(depthsam, shadowUV + float2(-U, 0)).r, 0.0f) * 1500 - 0.5f);
comp += saturate(max(input.lpos.z - depthtex.Sample(depthsam, shadowUV + float2(0, V)).r, 0.0f) * 1500 - 0.5f);
comp += saturate(max(input.lpos.z - depthtex.Sample(depthsam, shadowUV + float2(0, -V)).r, 0.0f) * 1500 - 0.5f);
comp += saturate(max(input.lpos.z - depthtex.Sample(depthsam, shadowUV + float2(U, V)).r, 0.0f) * 1500 - 0.5f);
comp += saturate(max(input.lpos.z - depthtex.Sample(depthsam, shadowUV + float2(-U, V)).r, 0.0f) * 1500 - 0.5f);
comp += saturate(max(input.lpos.z - depthtex.Sample(depthsam, shadowUV + float2(U, -V)).r, 0.0f) * 1500 - 0.5f);
comp += saturate(max(input.lpos.z - depthtex.Sample(depthsam, shadowUV + float2(-U, -V)).r, 0.0f) * 1500 - 0.5f);

// 出したものの平均を取得
comp = 1 - saturate(comp / 9);

// そのまま入れると黒が強いので、少しだけ薄める
output.color0.xyz *= comp / 2.0f + 0.2;
```

シャドウマップの作成

- 影の実装

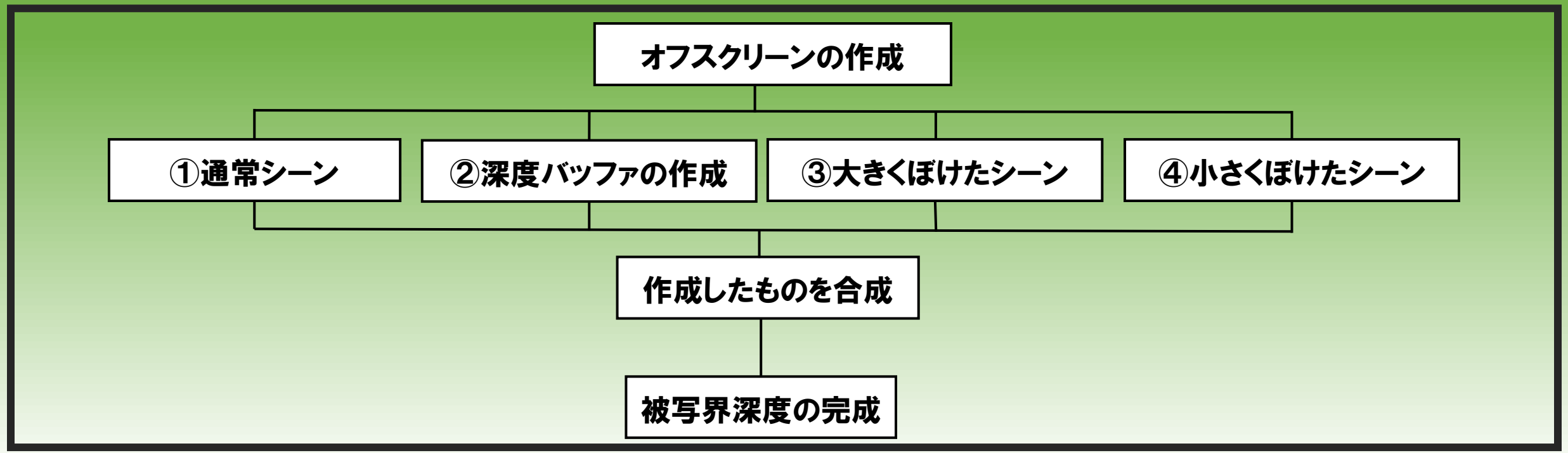


ポストエフェクト

- **被写界深度**

被写界深度を実装するにあたって、以下の工程で実装しました。
大きくぼけたシーンと小さくぼけたシーンを準備する理由は合成したときにより自然に見えるようにするためです。

①通常シーンはぼかしていないスクリーンを作成します。
②深度バッファは、作成したオフスクリーンの深度値を求めます。
③のぼけたシーンはオフスクリーンに強めのブラーをかけます。
④のぼけたシーンは③と同じくブラーをかけますが、弱めのブラーにします。



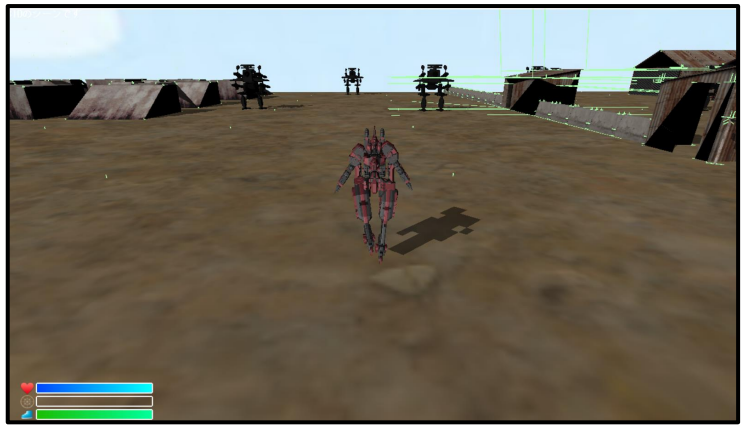
ポストエフェクト

- 深度バッファの作成

被写界深度を実装する際にカメラから見た深度バッファが必要となるのでカメラ空間でのZ値を描画した画像を用意します。このプログラムは範囲内か範囲外かをもとめて作成しました。

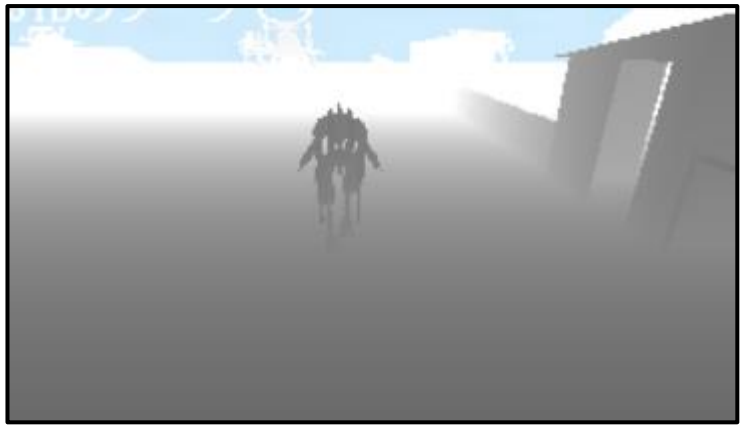
```
PSOutput output;
// 被写界深度の範囲内を0.0f～1.0fに変換
if (input.pos.z < DepthStart)
{
    output.color.x = 0.0f;
    output.color.y = 0.0f;
    output.color.z = 0.0f;
}
else if (input.pos.z > DepthEnd)
{
    output.color.x = 1.0f;
    output.color.y = 1.0f;
    output.color.z = 1.0f;
}
else
{
    output.color.x = (input.pos.z - DepthStart) * DepthScope;
    output.color.y = (input.pos.z - DepthStart) * DepthScope;
    output.color.z = (input.pos.z - DepthStart) * DepthScope;
}
output.color.w = 1.0f;
return output;
```

元の画像



深度バッファに変換

深度バッファ



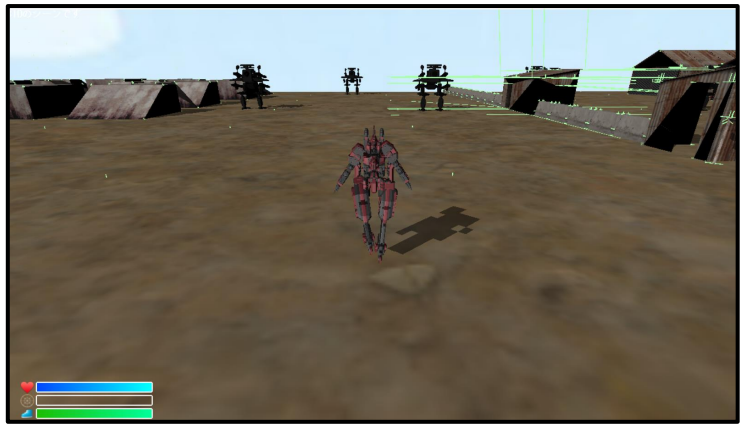
ポストエフェクト

- 被写界深度の実装

先ほど取得した深度バッファと元のスクリーンバッファ、強めのブラーと弱めのブラーを合成して被写界深度を実装しました。
DepthStartとDepthEndの間はブラーをかけないようにし、ブラーをかけるところでもfadeを使い0～1の中で0.5までは弱いブラーをかけて、それ以上は強いブラーをかけるようにして実装していきました。

```
    // 深度値を取得する
    float d = depth.Sample(sam,input.u

    // フォーカス情報からぼやけ率を算出
    if ( d < DepthStart)
    {
        fade = 1.0f - d / DepthStart;
    }
    else if (d < DepthEnd)
    {
        fade = 0.0f;
    }
    else
    {
        fade = (d - DepthEnd) / (1.0f
    }
    // ぼやけ率から色を算出
    if (fade < 0.5f)
    {
        color1 = tex.Sample(sam, input
        color2 = boke1.Sample(sam, inp
        blend = fade / 0.5f;
    }
    else
    {
        color1 = boke1.Sample(sam, inp
        color2 = boke2.Sample(sam, inp
        blend = (fade - 0.5f) / 0.5f;
    }
    output.color = lerp(color1,color2,
    return output;
}
```

元の画像



被写界深度の実装

被写界深度の実装

ポストエフェクト

・FOGの実装

被写界深度のみだと空気感が伝わりにくかったので、フォグを一緒に出すことにより空気感が出るようにしました。

深度値を参考に徐々に空と同じになるようにしていき、被写界深度のようにぼやけるようにもしていきました。

ぼやけ率から表示する色を決定

```
float4 depthOfColor(float fade, float4 color1, float4 color2)
{
    float4 color1_, color2_;
    float blend;
    // 取得した範囲から色をとる
    if (fade < 0.5f)
    {
        color1_ = color1;
        color2_ = float4(color2.xyz, fade);
        blend = fade / 0.5f;
    }
    else
    {
        blend = (fade - 0.5f) / 0.5f;
        // この2つの処理が空の色にする
        color1_ = lerp(color1, float4(color2.xyz, 0.5f), 1);
        color2_ = float4(color2.xyz, fade);
    }

    return lerp(color1_, color2_, blend);
}
```

徐々に空の色に近づくようにした

```
// メインのスクリーンと縮小バッファとのラープ
float4 lerpColor(float fade, float4 color1, float4 color2)
{
    float4 color1_, color2_;
    float blend;
    if (fade < 0.5f)
    {
        color1_ = color1;
        color2_ = color1;
        blend = fade / 0.5;
    }
    else
    {
        color1_ = color1;
        color2_ = color2;
        blend = (fade - 0.5f) / 0.5f;
    }
    return lerp(color1_, color2, blend);
}
```

被写界深度のみ

被写界深度の実装

FOG＋被写界深度