頂点シェーダを使ってみよう

ピクセルシェーダと同じ要領で、頂点シェーダ用のクラスを作成しましょう。

例①

ピクセルシェーダクラス --- 設定、描画	頂点シェーダクラス --- 設定、描画

イメージしやすいが、描画が２回必要なわけではない。
（ 1回のみで良い ）

例②

シェーダクラス --- ピクセルシェーダ設定 頂点シェーダ設定 描画

描画自体は1回で良いのであれば、
この形の方がよりシンプルかもしれません。

例③

シェーダ設定 --- ピクセル 頂点	シェーダ描画 --- ピクセル設定反映 頂点設定反映 描画

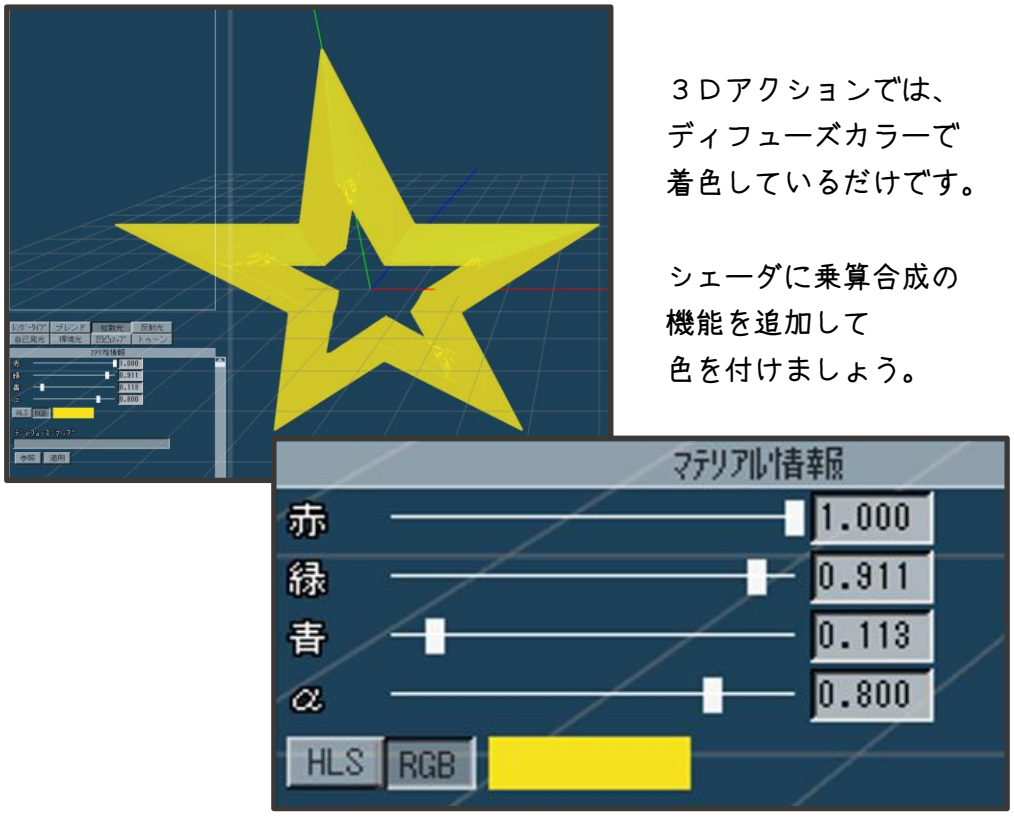
例②でクラスが肥大化するのが嫌な方は、役割でクラスを分けると
良いでしょう。

頂点シェーダの方は、教材で紹介している必要最低限のコードで作り、
ピクセルシェーダの方も、UV座標からテクスチャの色を取り出すだけの
最低限の機能で作成してみましょう。

演習①　WarpStarクラスのモデルを
オリジナルシェーダで描画しましょう

最初はこんな感じでＯＫ。

これができたら、モデルに色を付けてみましょう。



３Ｄアクションでは、
ディフューズカラーで
着色しているだけです。

シェーダに乗算合成の
機能を追加して
色を付けましょう。

演習②　半透明対策

ディフューズカラーのアルファが 0.8 となっていましたので、
おそらくスターモデルが半透明になっていると思います。

星の中に描かれている模様をよく見て欲しいのですが、
丘の背景ではなく、空の背景が半透明として、写りこんでいます。

半透明は処理負荷も高く、本格的に対応すると、処理の制御も難しいです。
ここでは、ひとまず描画順を調整して、正しい描画となるようにしましょう。

※難しく考えなくて大丈夫です

演習③　メインステージのモデルを
オリジナルシェーダで描画しましょう

初めは色がついていなくて大丈夫です。
ひとまず、オリジナルシェーダを使用するようにしましょう。

色を付けるためには

色がついていない理由ですが、
ピクセルシェーダにテクスチャを渡していないのが原因だと思われる
かもしれませんが、MV1LoadModelで予めロードしているモデルの場合、
おそらくモデルのハンドルIDを通して、自動的にテクスチャが渡るように
なっているか、独自で渡さなくとも大丈夫なようです。

原因は、頂点シェーダ内で、
UV座標をアウトプットに渡していないことにあります。

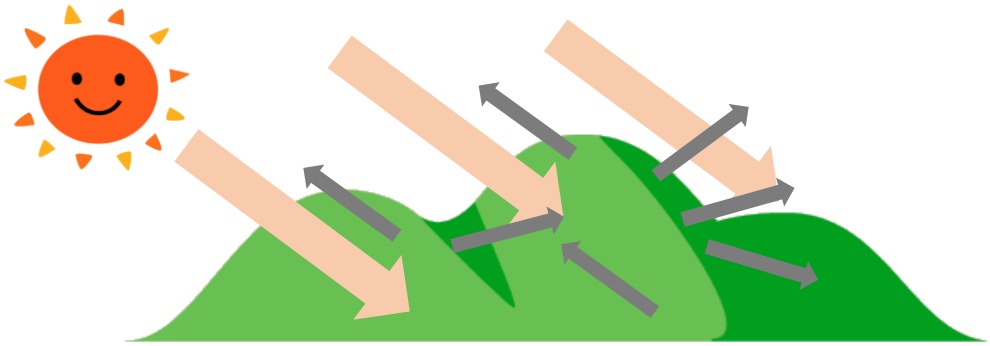
頂点シェーダに以下を追加。
UV座標の他にも、念のため値を設定しておく。

```
// その他、ピクセルシェーダへ引継&初期化 ++++++++++++( 開始 )
// UV座標
ret.uv.x = VSInput.uv0.x;
ret.uv.y = VSInput.uv0.y;
// 法線
ret.normal = VSInput.norm;
// ディフューズカラー
ret.diffuse = VSInput.diffuse;
// ライト方向(ローカル)
ret.lightDir = float3(0.0f, 0.0f, 0.0f);
// ライトから見た座標
ret.lightAtPos = float3(0.0f, 0.0f, 0.0f);
// その他、ピクセルシェーダへ引継&初期化 ++++++++++++( 終了 )
```

やっと色がつきました。
しかし、デフォルトシェーダと比較して凹凸が無くなっているような
気がします。これは、ライト処理によって、凹凸を明るくしたり、
暗くしたりする効果が無くなったためです。

超、簡単なライト処理

理論はこうです。



ライトの方向と、モデルの法線方向が同じだったら暗い、
ライトの方向と、モデルの法線方向が逆だったら明るい、です。

※　本当は、距離による減衰だったり、影であったり、
　　反射であったり、様々な計算をしないといけません。
　　写実的なリアルテイストの場合は、もっと物理を勉強して、
　　PBRを実装できるようになりましょう

今回は、とにかくシンプルに。イージーに。
シェーダに慣れていきましょう。

まずは、ライト方向をしっかり把握する必要があります。
シェーダのデバッグは難しいですが、ピクセルシェーダの方で、
色を使ってライト方向を確認していきたいと思います。

SceneManagerクラスのInit3D関数で、
わかりやすいように、ライトの方向を単純化します。

```
// ライトの設定
//ChangeLightTypeDir({ 0.3f, -0.7f, 0.8f });
ChangeLightTypeDir({ 0.0f, 0.0f, 0.5f });
```

Zの正方向にしていますので、真正面にライトが向いている、
もし、この方向を色で表すとしたら、RGBのBにあたりますので、
青になるはずです。

CommonShader3DHeaderの中に、
DxLibで定義されているLight構造体にdirection属性があります。
ビュー空間とコメントに書いてありますので、これは今回使えないのですが、
何となくビュー空間がイメージできる良い機会ですので、使ってみましょう。

ビュー空間は、カメラを基準とした時の空間です。
カメラを向きを変えると、基準自体が変わりますので、
その空間上のライト方向も変わるはずです。

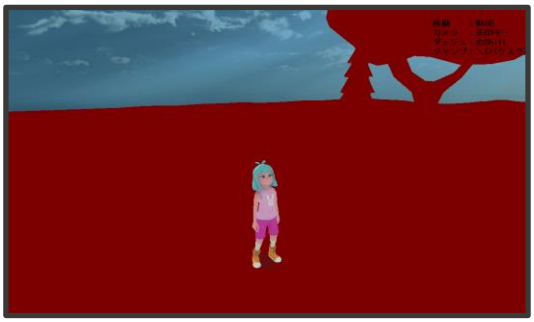
ピクセルシェーダのアウトプットを下記に書き換えましょう。

```
float3 lightDir = g_common.light[0].direction;
return float4(lightDir.x, lightDir.y, lightDir.z, 1.0f);
```

カメラ 0 度

カメラ 90 度

カメラ 180 度



カメラ 270 度

カメラを回すと色が変わることから、
ビュー空間の方向だということがわかります。

ローカルなら、ローカル、ワールドなら、ワールド、
ビューなら、ビュー。座標系は合わさないと、計算自体ができません。
イメージしやすいのは、やはりワールド座標系だと思いますので、
ワールド座標系で、超簡易ライト処理を実装していきます。

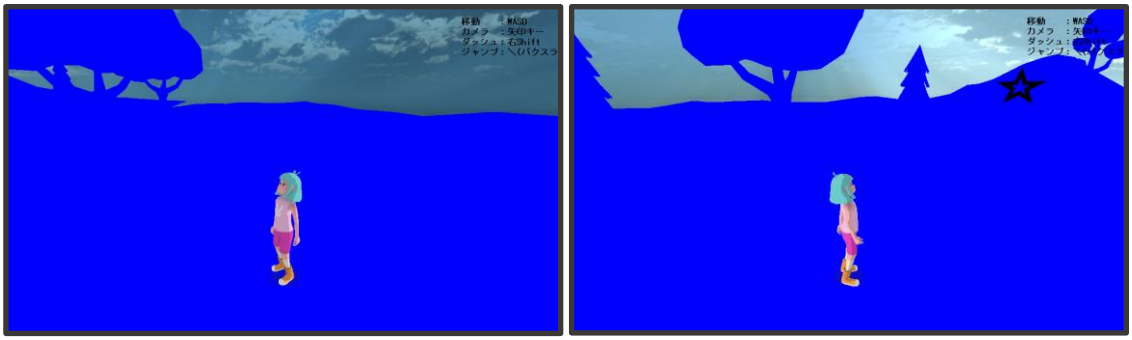
DxLibの関数で、GetLightDirectionを使えば、
ワールド空間上のライト方向を取得することができますので、
その値をピクセルシェーダに渡してください。

渡された定数バッファを g_light_dir とした場合、

return float4(g_light_dir.x, g_light_dir.y, g_light_dir.z, 1.0f)

ピクセルシェーダのアウトプットを上記のようにします。

ライトのワールド空間上の方向は、{ 0.0f, 0.0f, 0.5f } に設定して
いますので、カメラがどの方向を向いても、青色になるはずです。



次にポリゴンの法線を用意する必要があります。

頂点シェーダ内で、

```
    // 法線
    ret.normal = VSInput.norm;
```

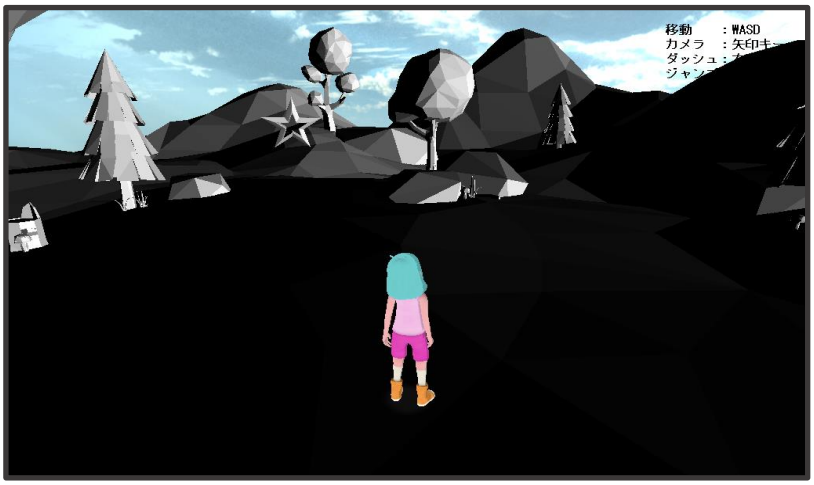
ポリゴンの法線をモデル情報からアウトプットにコピーしていますが、
これでは不十分です。

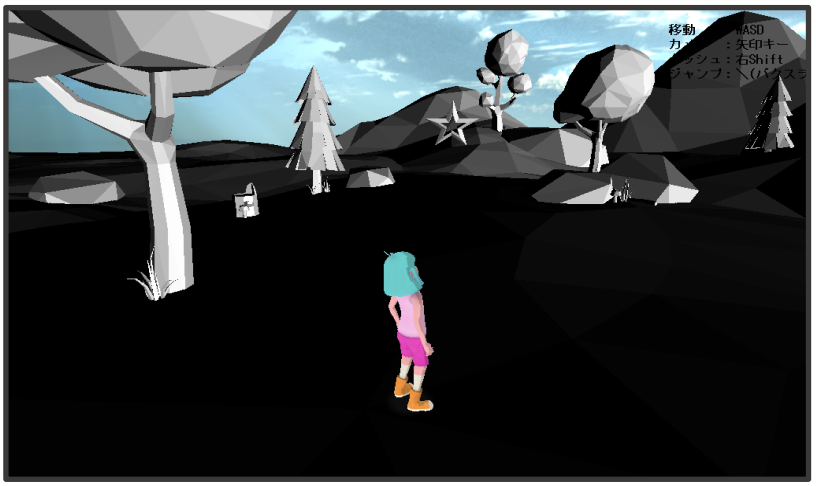
繰り返しになりますが、頂点シェーダのお仕事は座標変換です。
ここでは、モデルのローカル座標系から、ワールド座標系に変換したいと
思いますので、ワールド変換行列を使用して、
ワールド空間上の方向(法線)に変換します。

```
    // 法線をローカル空間からワールド空間へ変換
    ret.normal = normalize(
                    mul(VSInput.norm, (float3x3)g_base.localWorldMatrix));
```

確認のために、ピクセルシェーダで法線情報を色に充ててみて確認します。

```
// ③法線がワールド空間になっているか確認
return float4(
        PSInput.normal.x, PSInput.normal.x, PSInput.normal.x, 1.0f);
```





RGBの色に、法線のX方向のみを充てているため、
右方向に法線が向いているポリゴンは、白(1.0)に近づき、
右方向を全く向いていないポリゴンは、黒(0.0)になっているのが、
確認できました。

法線の座標変換もバッチリできているようです。

最後に、ライトの方向と、法線の方向を比較して、
逆方法だったら明るく(白 = 1.0)に近づけるようにしましょう。

方向の比較を行うためには、定番の内積を使います。
内積は、同じ方向に近づくほど 0.0 → 1.0 に近づきます。

演習③　ライト方向、法線、内積を使用して、色を明るくしたり、暗くしたりしてみてください

かなり標準シェーダに近づいたのではないでしょうか。

今作ったシェーダセットを、オリジナルの標準シェーダにしておいて、
このシェーダを元に色々とカスタマイズしていきましょう。