Unityはじめるよ

～シェーダーをいじってみる～

統合開発環境を内蔵したゲームエンジン
http://japan.unity3d.com/

※いろんな職業の方が見る資料なので説明を簡単にしてある部分があります。正確には本来の意味と違いますが上記理由のためです。ご了承ください。
この資料内の一部の画像、一部の文章はUnity公式サイトから引用しています。

本日のプレゼン内容

・シェーダーについて

・シェーダーを作ってみよう

・まとめ

シェーダーとは？

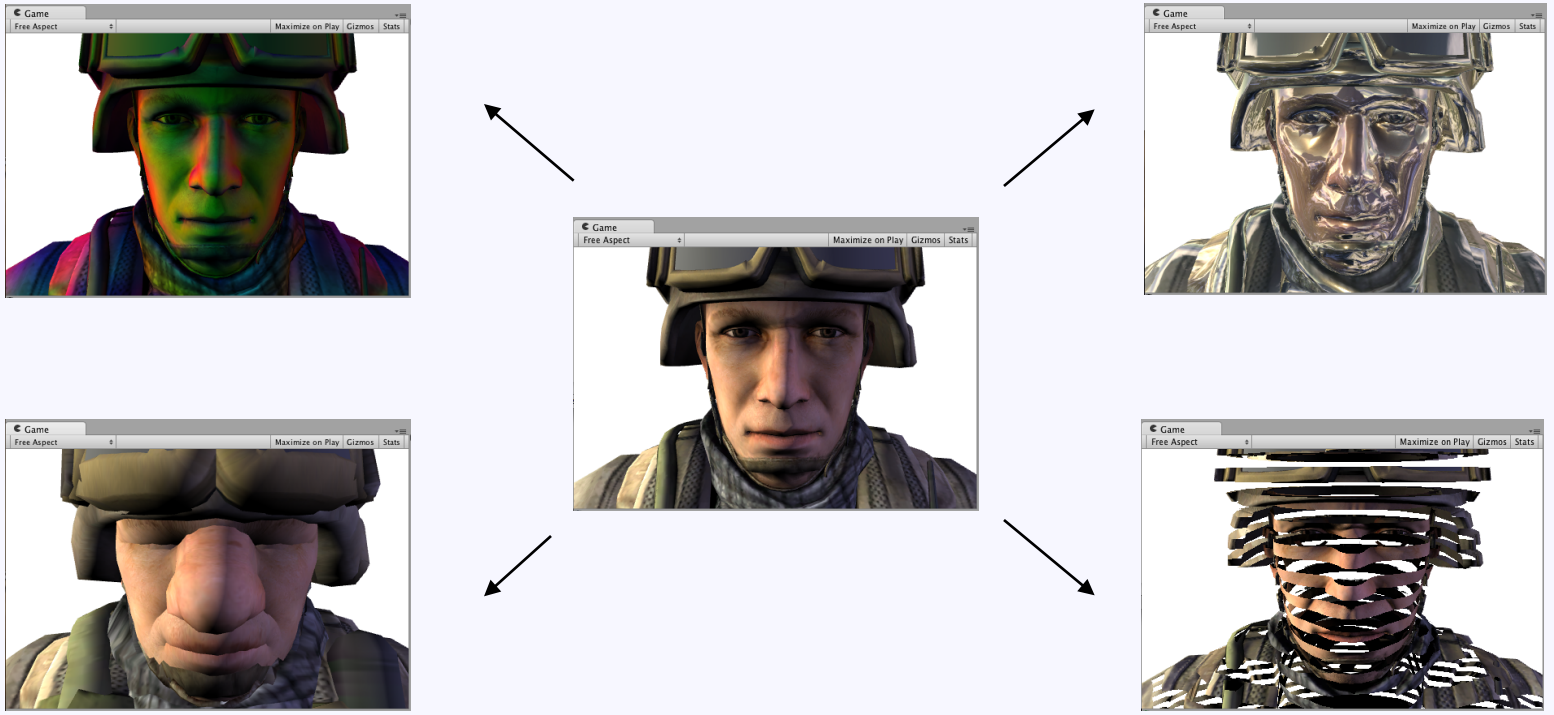
3Dオブジェクトを画面に描画するための
小さなプログラムのこと。

プログラマブルシェーダーが生まれてから、
劇的に表現の幅が広がった。

特徴
- GPUで実行される。
- 実行時にコンパイルする

シェーダーでできること

モデルの色や質感をいじったり、画面に特殊効果をつけたり、モデルの頂点を動かしたり。



公式マニュアルから画像引用
https://docs.unity3d.com/ja/540/Manual/SL-SurfaceShaderExamples.html

シェーダーの種類

- **頂点シェーダー**(バーテックスシェーダー)
  モデルの頂点やテクスチャ座標などの頂点情報をいじる。軽い。
- **ジオメトリシェーダー**(プリミティブシェーダー)
  頂点を増やしたり減らしたりする。
- **フラグメントシェーダー**(ピクセルシェーダー)
  画素の色を作る(最終的なレンダリングを行う)。重い。
- **テッセレーションシェーダー**
  頂点を分割して構成しモデルを作成。
- **コンピュートシェーダー**
  GPGPUに使う(GPUを汎用計算に使う)。

シェーダー言語

- **GLSL**
  OpenGL用
- **HLSL**
  DirectX用
- **MSL**
  iOSのMetal用
- **PSSL**
  PS4用
- **Cg**
  OpenGLでもDirectXでも使える

Unityでのシェーダーの種類

UnityにはShaderLabというシェーダープログラムを簡単に書く仕組みが備わっている。
最終的にはプラットフォームに合わせたシェーダー言語に変換される。

- **Standard Surface Shader**
  光と影の処理をUnityが自動で生成する。なのでシェーダーを簡単に記述できる。
  その代わり複雑な処理（トリッキーな方法など）には向いていない。
- **Unlit Shader**
  頂点シェーダーとフラグメントシェーダー。高度な知識が必要だが自由度が高い。
- **Image Effect Shader**
  ポストエフェクト向け。画面全体の色を変えたりボカしたりなど。
- **Compute Shader**
  GPGPU向け。

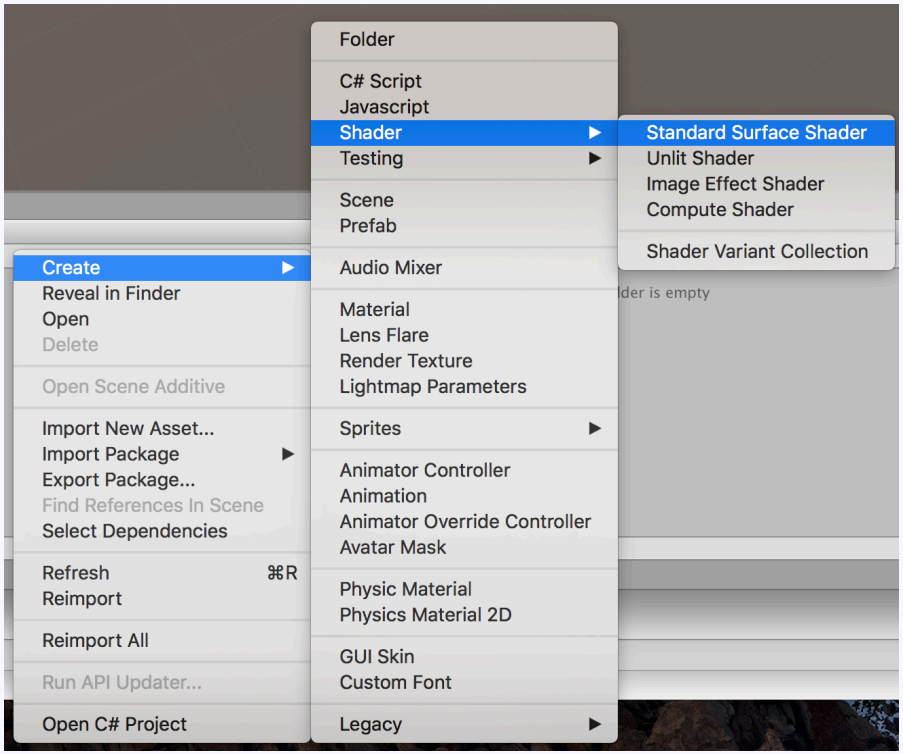
公式マニュアル　シェーダーを書く
https://docs.unity3d.com/ja/540/Manual/ShadersOverview.html

シェーダーを作ってみよう

公式マニュアル　シェーダーアセット
https://docs.unity3d.com/ja/540/Manual/class-Shader.html

Surface Shader を作ってみる

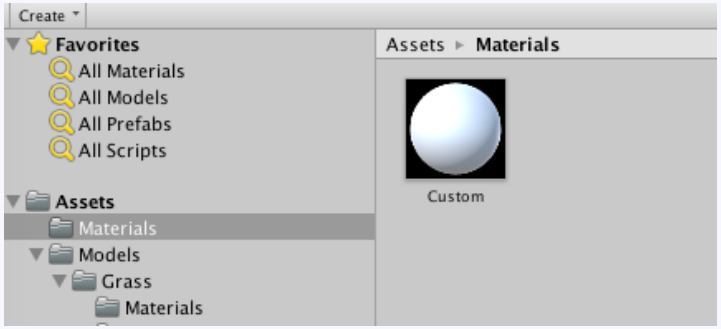
プロジェクトビューで右クリック
Create > Shader > Standard Surface Shader
でシェーダーファイルが作られます。名前は「**SimpleShader**」としました。



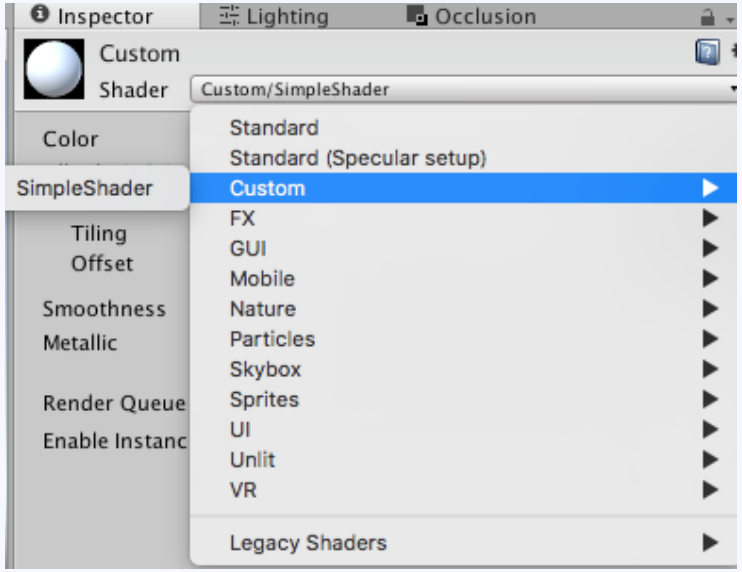
シェーダファイルが
作られる

作ったシェーダーをマテリアルに適用する

プロジェクトビューのMaterialsフォルダで右クリック
Create > Material
でマテリアルを作る。
名前は「**Custom**」とでもしておきましょう。



マテリアルを選択した状態でインスペクタを見る。
その中の、Shaderドロップダウンをクリックし、
Custom > SimpleShader
でシェーダーを選択する。

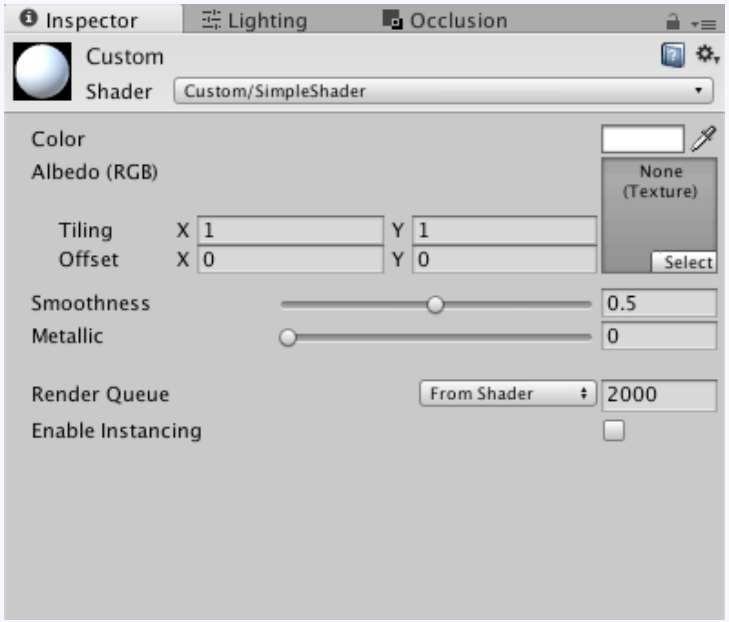


作ったシェーダーのプロパティ

デフォルトの状態で
右の図のようなプロパティがあります。

なにやら、いろいろな項目がありますね。

シェーダーソースを見てみましょう。



```
Shader "Custom/SimpleShader" {
    Properties {
        _Color ("Color", Color) = (1,1,1,1)
        _MainTex ("Albedo (RGB)", 2D) = "white" {}
        _Glossiness ("Smoothness", Range(0,1)) = 0.5
        _Metallic ("Metallic", Range(0,1)) = 0.0
    }
    SubShader {
        Tags { "RenderType"="Opaque" }
        LOD 200

        CGPROGRAM
        // Physically based Standard lighting model, and enable shadows on all light types
        #pragma surface surf Standard fullforwardshadows

        // Use shader model 3.0 target, to get nicer looking lighting
        #pragma target 3.0

        sampler2D _MainTex;

        struct Input {
            float2 uv_MainTex;
        };

        half _Glossiness;
        half _Metallic;
        fixed4 _Color;

        // Add instancing support for this shader. You need to check 'Enable Instancing' on materials that use the shader.
        // See https://docs.unity3d.com/Manual/GPUInstancing.html for more information about instancing.
        // #pragma instancing_options assumeuniformscaling
        UNITY_INSTANCING_CBUFFER_START(Props)
            // put more per-instance properties here
        UNITY_INSTANCING_CBUFFER_END

        void surf (Input IN, inout SurfaceOutputStandard o) {
            // Albedo comes from a texture tinted by color
            fixed4 c = tex2D (_MainTex, IN.uv_MainTex) * _Color;
            o.Albedo = c.rgb;
            // Metallic and smoothness come from slider variables
            o.Metallic = _Metallic;
            o.Smoothness = _Glossiness;
            o.Alpha = c.a;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```

シェーダーファイルの中身

シンプルなシェーダーに書き直そう

シェーダーファイルを作っただけで、あれだけのソースを自動で生成してくれます。

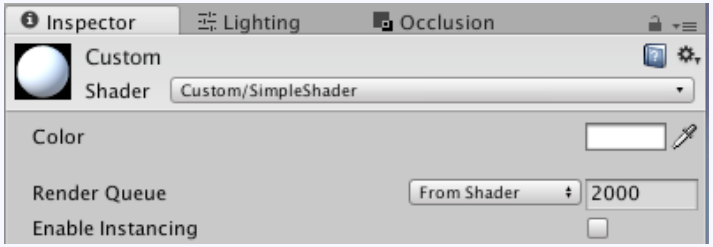
ただ、シェーダーの勉強をするには、ちと難しい。

なので、わかりやすくなるように、シンプルな色だけを設定できるシェーダーに

書き直してしまいましょう。

公式マニュアル　サーフェイスシェーダーの例
https://docs.unity3d.com/ja/540/Manual/SL-SurfaceShaderExamples.html

```
Shader "Custom/SimpleShader" {
    Properties {
        _Color ("Color", Color) = (1,1,1,1)
    }
    SubShader {
        Tags { "RenderType"="Opaque" }

        CGPROGRAM
        #pragma surface surf Lambert

        struct Input {
            float4 color : COLOR;
        };

        fixed4 _Color;

        void surf (Input IN, inout SurfaceOutput o) {
            o.Albedo = _Color;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```



インスペクタでの表示

カラーピッカーが表示されている

Render Queue は描画の順番

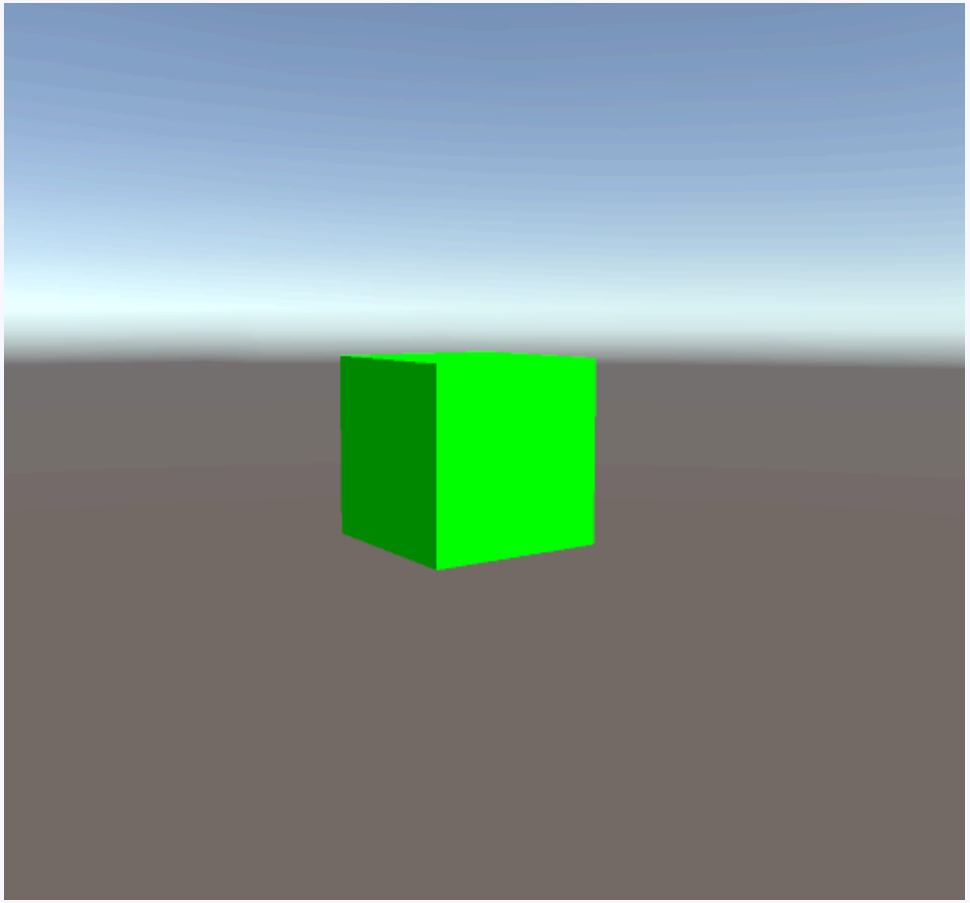
Enable Instancing はGPUでの描画バッチ処理

詳細は以下の公式リンクを見てね

公式マニュアル レンダリング順
https://docs.unity3d.com/jp/540/Manual/SL-SubShaderTags.html

公式マニュアル GPUインスタンシング
https://docs.unity3d.com/ja/540/Manual/GPUInstancing.html

シェーダーをつくってみよう



```
Shader "Custom/SimpleShader" {
    Properties {
        _Color ("Color", Color) = (1,1,1,1)
    }
    SubShader {
        Tags { "RenderType"="Opaque" }

        CGPROGRAM
        #pragma surface surf Lambert

        struct Input {
            float4 color : COLOR;
        };

        fixed4 _Color;

        void surf (Input IN, inout SurfaceOutput o) {
            o.Albedo = _Color;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```

Properties:
Unityエディタから
シェーダーに渡したい値

SubShader:
シェーダー本体。
ゲームを対応させたい
プラットフォーム分、
SubaShaderを作る。
その中の動くシェーダー
が利用される。

どのSubShaderでも動かなかった場合の最終シェーダー

Propertiesに注目してみる

```
Properties {
    _Color ("Color", Color) = (1,1,1,1)
}
```

Unityからシェーダーに渡したい値を記述する。
変数名(インスペクタ表示名,型)=デフォルト値
となる。型の種類は以下の通り。

型	記述方法	内容
Float	変数名(インスペクタ表示名, Float)=実数値	実数値
Vector	変数名(インスペクタ表示名, Vector)=(実数値,実数値,実数値,実数値)	x,y,z,wの４つのFloat値
Color	変数名(インスペクタ表示名, Color)=(正規化値,正規化値,正規化値,正規化値)	カラーピッカー表示される
Range	変数名(インスペクタ表示名, Range(最小値,最大値))=実数値	スライダー表示
2D	変数名(インスペクタ表示名, 2D)=“white”{}	2のベキ乗テクスチャ指定
Rect	変数名(インスペクタ表示名, Rect)=“white”{}	自由サイズテクスチャ指定
Cube	変数名(インスペクタ表示名, Cube)=“white”{}	Cubeテクスチャ指定

※テクスチャ系の”white”{}は白を表す

記述例、

```
Properties {
        _Float (”Float", Float) = 2.0
        _Vector (”Vector", Vector) = (2.0,1.0,1.5,2.2)
        _Color ("Color", Color) = (1,1,1,1)
        _Range (”Range", Range(0,1)) = 0.0
        _2D (”2D", 2D) = "white" {}
        _Rect (”Rect", Rect) = "white" {}
        _Cube (”Cube", Cube) = "white" {}
}
```

SubShaderに注目してみる

ここは説明する内容が多すぎるので、詳細は
Unity公式マニュアル
「○×つくろーどっとコム」さんのサイト
を参考にしてください。ここでは主要な部分のみ紹介します。

```
SubShader {
    Tags { "RenderType"="Opaque" }

    CGPROGRAM
    #pragma surface surf Lambert

    struct Input {
        float4 color : COLOR;
    };

    fixed4 _Color;

    void surf (Input IN, inout SurfaceOutput o) {
        o.Albedo = _Color;
    }
    ENDCG
}
```

描画タイプを指定。Unityのどのパイプラインに適用するかを決めるらしい。
Opaque=不透明　Transparent=半透明　TransparentCutout=抜き
Transparentを使うときは、下のpragmaに「alpha」をつける必要がある

CGPROGRAM　～　ENDCG　この中にシェーダーを書く

Surfaceと書くことででサーフェスシェーダーであることを宣言する
surfの部分は関数名、Lambertの部分はライティングモードを指定
ライトモードは、Lambert=拡散反射光　BlinnPhongは鏡面反射光

Unityからシェーダーに渡される構造体

：　以降の部分(この例ではCOLOR)をセマンティクスという(COLORは頂点カラーを意味する)

Propertiesで宣言した変数を同名で宣言。UnityEditorから渡された値を扱えるようになる

#pragmaで宣言したsurface関数
この関数の中を書き換えて質感を制御する

この例ではAlbedo(拡散反射光)にUnityEditorから受け取った色をそのまま設定している

RenderType	内容	処理負荷
Opaque	不透明な描画しか行わない	軽い
Transparent	半透明の描画を行う	重い
TransparentCutout	網のようなテクスチャの抜きがある描画を行う	GPUによっては重い。

ライティングモード	内容
Lambert	拡散反射光
BlinnPhong	鏡面反射光

Input構造体のセマンティクス	内容
POSITION	頂点位置
NORMAL	頂点の法線
TEXCOORD0, TEXCOORD1,2,3	テクスチャのUV座標。数字は何番目のテクスチャかを表す
TANGENT	接線
COLOR	頂点カラー

Output構造体	内容	型
Albedo	拡散反射光	half3
Normal	頂点の法線	half3
Emission	自発光度合い	half3
Specular	鏡面反射	half
Gloss	輝き	half
Alpha	透明度	half

シェーダー内で利用する型について

型	精度	内容
float	高	ワールド空間での座標や、テクスチャのUV値など、正確な値が必要なときに使う。
half	中	ローカル座標や、向き、HDRカラーなどに利用。
fixed	低	1/256精度。-2.0~2.0に収まる値に利用。色に利用される。
int	整数	カウンタなどに利用。サポートしていないGPUがあるので注意。モバイル系は使わないほうが良い。

fixed4やhalf3など、後ろに数字がつく場合は、
(fixed, fixed, fixed, fixed)、(half, half, half)ということ。
同じ型の数字を3つや、4つ持っているという意味。
色や座標を表しやすい。
.rgb、.xyz、.rgba、.xyzw、のように複数の値に同時にアクセス可能。

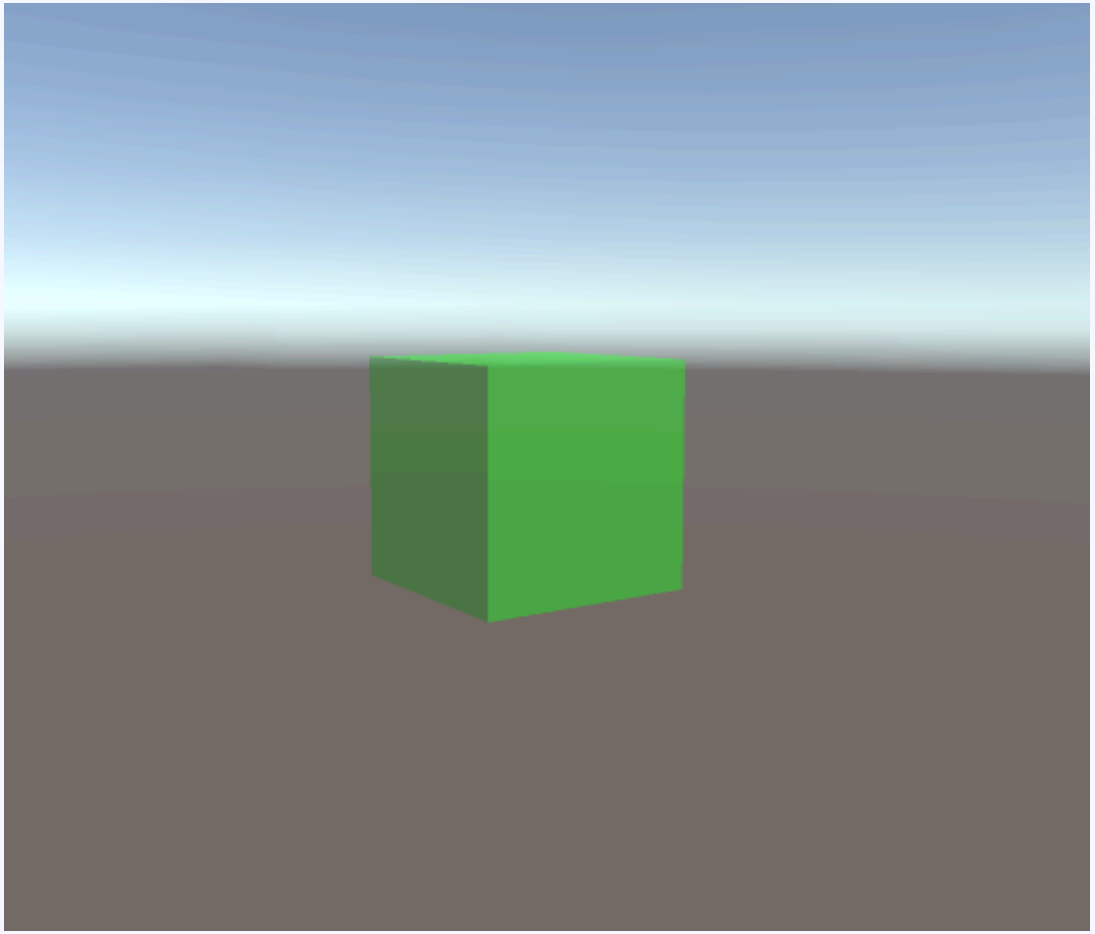
半透明描画できるシェーダーを書こう

半透明描画できるシェーダーを書こう

```
Shader "Custom/SimpleShader" {
    Properties {
        _Color ("Color", Color) = (1,1,1,1)       // Unity側で色をセット
    }
    SubShader {
        // 描画タイプ。Opaque=不透明  Transparent=半透明  TransparentCutout=抜き
        Tags { "RenderType"="Transparent" "Queue" = "Transparent" }

        // CGPROGRAM  ～  ENDCG  この中にシェーダーを書く
        CGPROGRAM

        // Surfaceと書くことででサーフェスシェーダーであることを宣言する。
        // surfの部分は関数名、Lambertの部分はライティングモードを指定。
        // ライトモードは、Lambert=拡散反射光  BlinnPhongは鏡面反射光
        #pragma surface surf Lambert alpha

        // Unityからシェーダーに渡される構造体
        struct Input {
            float4 color : COLOR;    // 色
        };

        // Propertiesで宣言した変数を同名で宣言。UnityEditorから渡された値を扱えるようになる
        fixed4 _Color;


        // #pragmaで宣言したsurface関数
        // この関数の中を書き換えて質感を制御する
        void surf (Input IN, inout SurfaceOutput o) {
            o.Albedo = _Color;
            o.Alpha = _Color.a;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```

赤文字の部分を追加・変更

シェーダーをつくってみよう



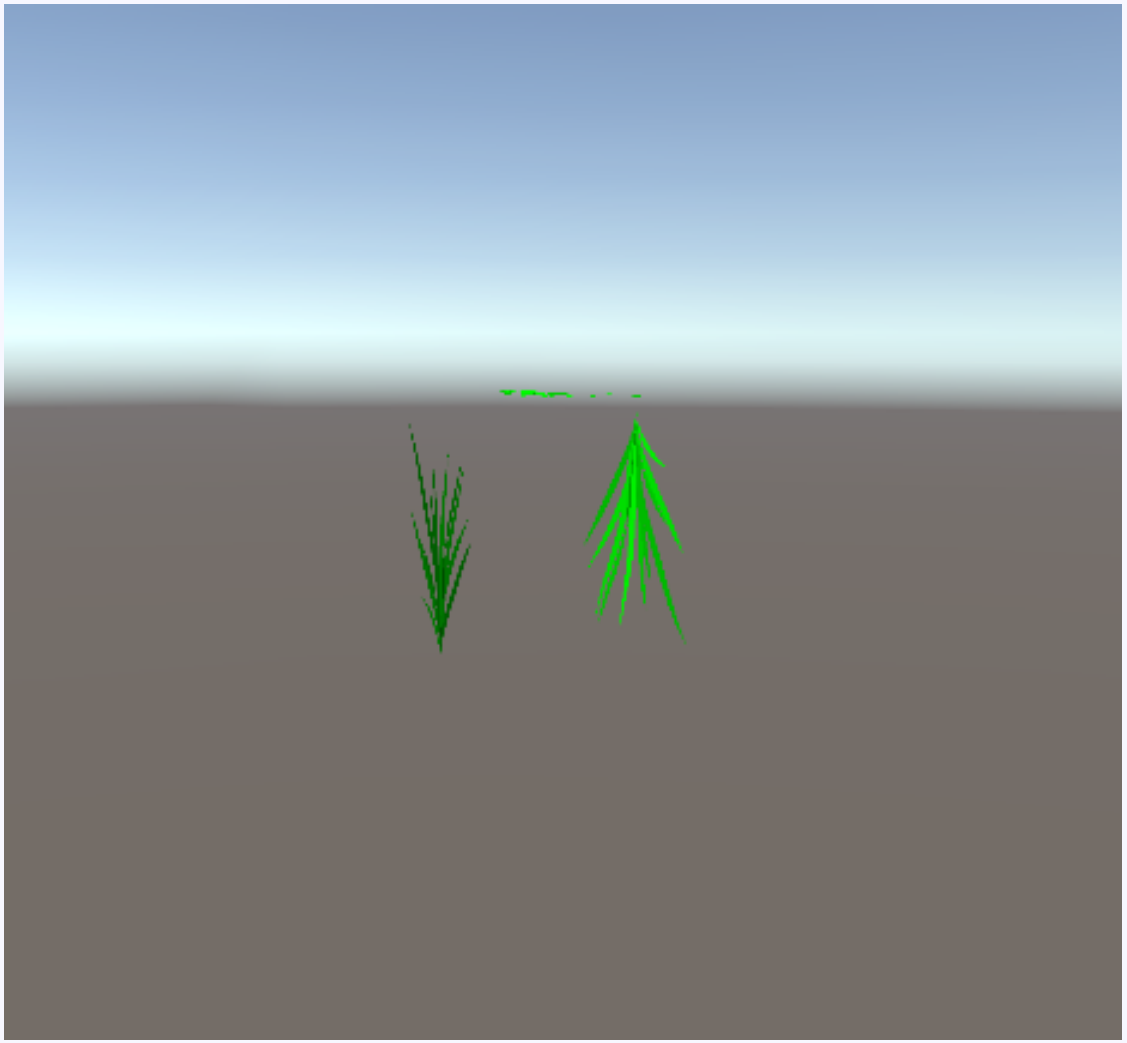
テクスチャ描画ができる
シェーダーを書こう

テクスチャ描画ができるシェーダーを書こう

```
Shader "Custom/SimpleShader" {
    Properties {
        _Color ("Color", Color) = (1,1,1,1)        // Unity側で色をセット
        _MainTex ("Texture", 2D) = "white" {}     // Unity側でテクスチャをセット
    }
    SubShader {
        // 描画タイプ。Opaque=不透明   Transparent=半透明   TransparentCutout=抜き
        Tags { "RenderType"="Transparent" "Queue" = "Transparent"  }

        // CGPROGRAM   ～   ENDCG  この中にシェーダーを書く
        CGPROGRAM

        // Surfaceと書くことででサーフェスシェーダーであることを宣言する。
        // surfの部分は関数名、Lambertの部分はライティングモードを指定。
        // ライトモードは、Lambert=拡散反射光   BlinnPhongは鏡面反射光
        #pragma surface surf Lambert alpha

        // Unityからシェーダーに渡される構造体
        struct Input {
            float4 color : COLOR;    // 色
            float2 uv_MainTex;       // Propertiesで宣言した変数の前に、uv をつけることで、マテリアル情報が適用された座標を受け取れる
        };

        // Propertiesで宣言した変数を同名で宣言。UnityEditorから渡された値を扱えるようになる
        fixed4 _Color;
        sampler2D _MainTex;


        // #pragmaで宣言したsurface関数
        // この関数の中を書き換えて質感を制御する
        void surf (Input IN, inout SurfaceOutput o) {
            // tex2D関数：UV座標(IN.uv_MainTex)からテクスチャ(_MainTex)上のピクセルの色を計算して返す
            o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb * _Color;
            o.Alpha = _Color.a * tex2D (_MainTex, IN.uv_MainTex).a;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```

赤文字の部分を追加・変更

シェーダーをつくってみよう



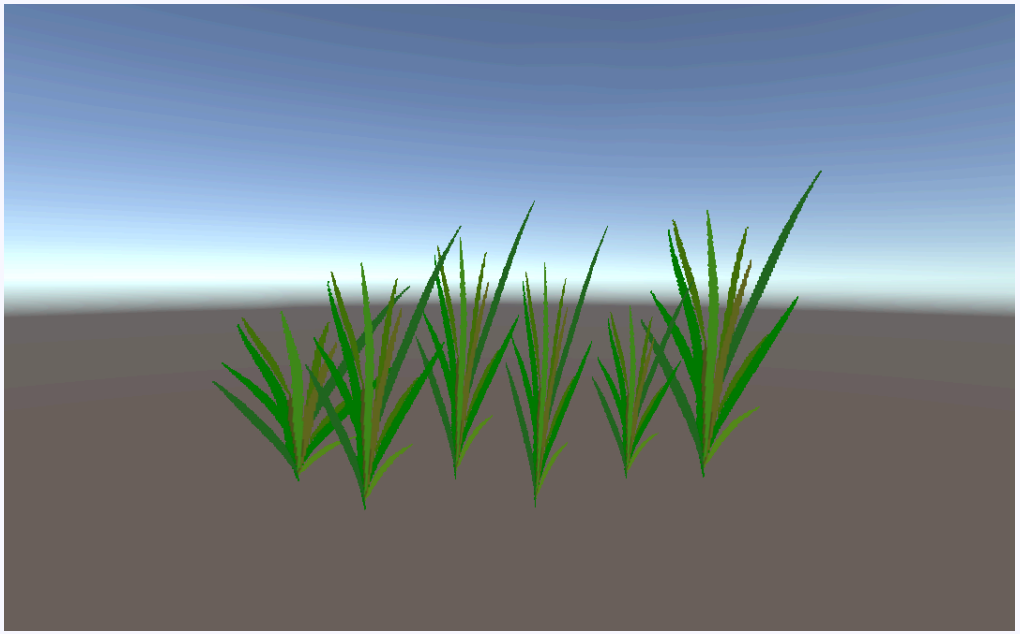
頂点シェーダーをいじってみる

風に揺れる草を表現する
シェーダーを書こう

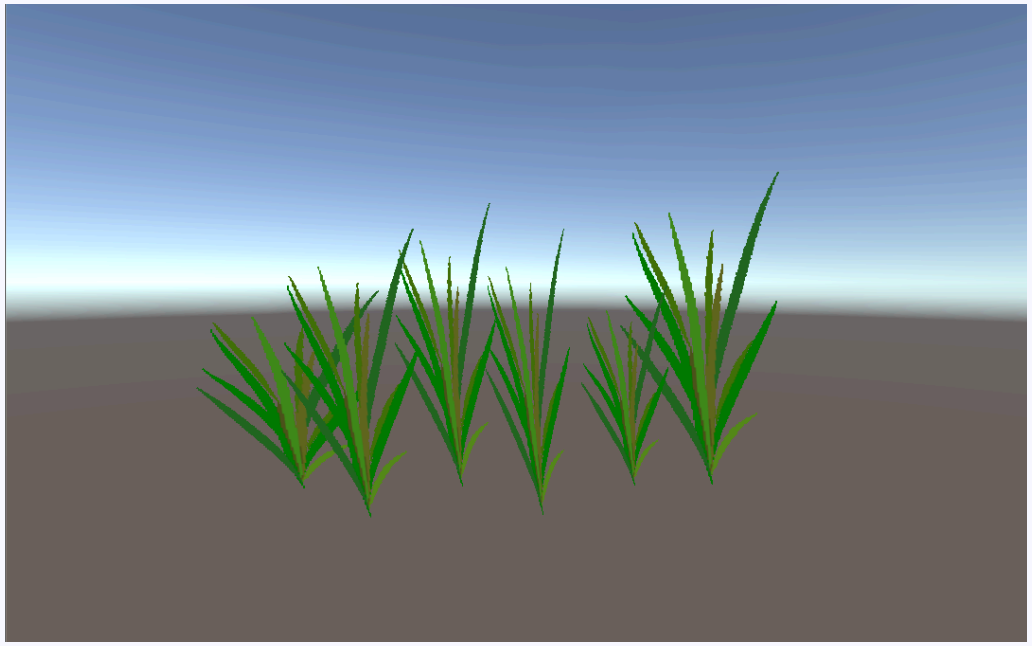
風に揺れる草を表現するシェーダーを書こう

```
Shader "Simple/SimpleGrassWave" {
    Properties {
        _Color ("MainColor", Color) = (1,1,1,1)              // Unity側で色をセット
        _MainTex ("MainTexture", 2D) = "white" {}             // Unity側でテクスチャをセット
        _Cutoff ("AlphaCutoff", Range(0,1)) = 0.5             // Unity側でテクスチャの抜きレベルを調整
        _Cycle ("Cycle", Range(0.0,5.0) ) = 1.0               // Unity側で揺れのサイクルをセット
        _Amplitude ("Amplitude", Range(0,1.0) ) = 0.02        // Unity側で揺れの量をセット
    }

    SubShader {
        // Queue 描画順(Background は 1000、Geometry は 2000、AlphaTest は 2450、Transparent は 3000、そして Overlay は 4000。数字が小さいほど先に描画される)
        // 描画タイプ。Opaque=不透明  Transparent=半透明  TransparentCutout=抜き
        Tags { "Queue"="AlphaTest" "RenderType"="TransparentCutout" }

        // CGPROGRAM  ～  ENDCG  この中にシェーダーを書く
        CGPROGRAM

        // Surfaceと書くことでサーフェスシェーダーであることを宣言する。
        // surfの部分は色をいじる関数名を指定、vertの部分は頂点をいじる関数名を指定、Lambertの部分はライティングモードを指定。
        // alphatest:_Cutoffの部分で指定したアルファ値より大きいピクセルのみ描画するようにしている
        // ライトモードは、Lambert=拡散反射光  BlinnPhongは鏡面反射光
        #pragma surface surf Lambert alphatest:_Cutoff vertex:vert

        // Propertiesで宣言した変数を同名で宣言。UnityEditorから渡された値を扱えるようになる
        sampler2D _MainTex;
        float4 _Color;
        float _Cycle;
        float _Amplitude;


        // Unityからシェーダーに渡される構造体
        struct Input {
            float2 uv_MainTex;
            fixed4 color : COLOR;
        };

        // #pragmaで宣言したsurface関数 色に関する処理はここに書く
        // この関数の中を書き換えて質感を制御する
        void surf (Input IN, inout SurfaceOutput o) {
            half4 col = tex2D(_MainTex, IN.uv_MainTex) * _Color;
            o.Albedo = col;
            o.Alpha = col.a;
        }

        // #pragmaで宣言したvertex関数  頂点をいじる処理はここに書く
        void vert (inout appdata_full v) {
            // _TimeはShader内でアニメーションを行いたい時に利用  (x, y, z, w) が (t/20、t、t×2、t×3)となる。つまりyを使えばtそのままの値となる
            // 最後にv.vertex.yを乗算することで根元は揺らさず上部の揺れ幅を大きくしている
            v.vertex.x += sin((_Time.y * _Cycle)) *  _Amplitude * v.vertex.y;
            v.vertex.z += cos((_Time.y * _Cycle)) * _Amplitude * v.vertex.y;
        }

        ENDCG
    }
}
```

赤文字の部分を追加・変更



風に揺れる草を表現するシェーダーを書こう

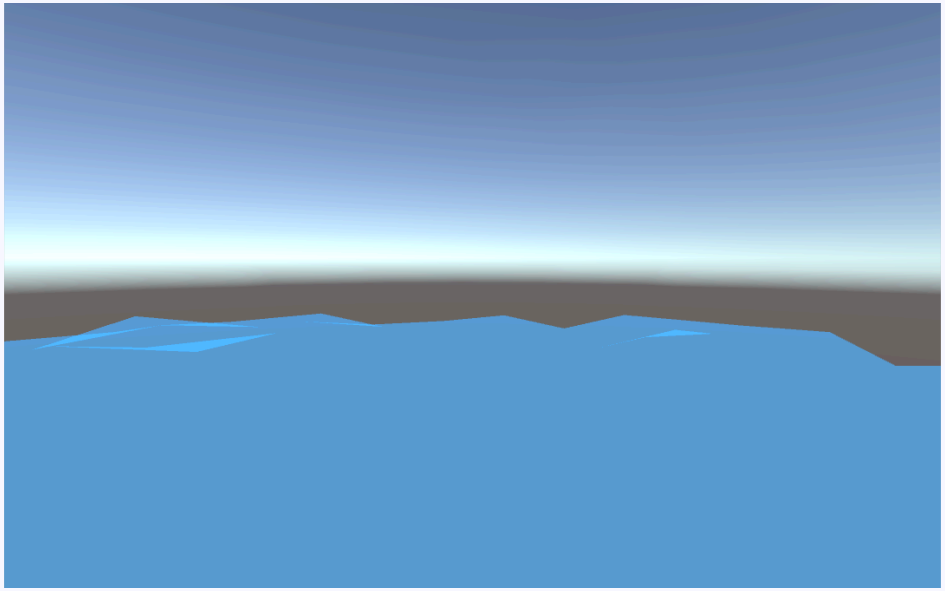


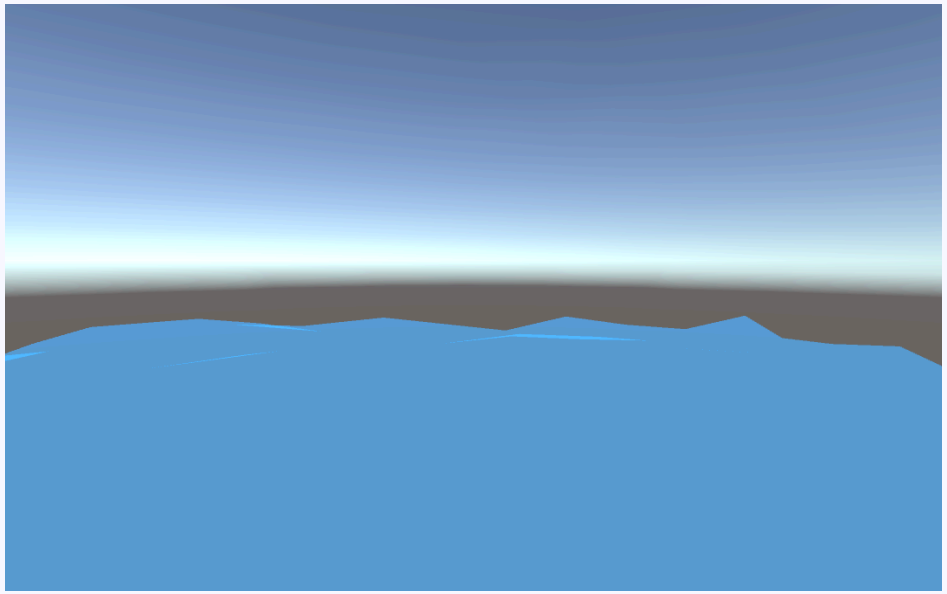
波の揺れを表現するシェーダーを書こう

波の揺れを表現するシェーダーを書こう

```
Shader "Simple/SimpleWaterWave.shader" {
    Properties {
        _Color ("MainColor", Color) = (1,1,1,1)                 // Unity側で色をセット
        _Cycle ("Cycle", Range(0.0,5.0) ) = 1.0                 // Unity側で揺れのサイクルをセット
        _Amplitude ("Amplitude", Range(0,1.0) ) = 0.02         // Unity側で揺れの量をセット
    }

    SubShader {
        // Queue 描画順(Background は 1000、Geometry は 2000、AlphaTest は 2450、Transparent は 3000、そして Overlay は 4000。数字が小さいほど先に描画される)
        // 描画タイプ。 Opaque=不透明 Transparent=半透明 TransparentCutout=抜き
        Tags { "Queue"="Transparent" "RenderType"="Transparent" }

        // CGPROGRAM ～ ENDCG  この中にシェーダーを書く
        CGPROGRAM

        // Surfaceと書くことででサーフェスシェーダーであることを宣言する。
        // surfの部分は色をいじる関数名を指定、vertの部分は頂点をいじる関数名を指定、Lambertの部分はライティングモードを指定。
        // ライトモードは、Lambert=拡散反射光 BlinnPhongは鏡面反射光
        #pragma surface surf Lambert vertex:vert alpha

        // Propertiesで宣言した変数を同名で宣言。UnityEditorから渡された値を扱えるようになる
        float4 _Color;
        float _Cycle;
        float _Amplitude;

        // Unityからシェーダーに渡される構造体
        struct Input {
            fixed4 color : COLOR;
        };

        // #pragmaで宣言したsurface関数 色に関する処理はここに書く
        // この関数の中を書き換えて質感を制御する
        void surf (Input IN, inout SurfaceOutput o) {
            o.Albedo = _Color;
            o.Alpha = _Color.a;
        }

        // #pragmaで宣言したvertex関数 頂点をいじる処理はここに書く
        void vert (inout appdata_full v) {
            // _TimeはShader内でアニメーションを行いたい時に利用 (x, y, z, w) が (t/20、t、t×2、t×3)となる。つまりyを使えばtそのままの値となる
            // sin関数内でv.vertex.xとv.vertex.zを乗算することで頂点の位置によってい揺れるタイミングを変えている
            v.vertex.y += sin((_Time.y * _Cycle * v.vertex.x * v.vertex.z)) * _Amplitude;
        }

        ENDCG
    }
}
```

波の揺れを表現するシェーダーを書こう





おすすめシェーダーと参考サイト紹介

MatCap(質感と反射を表現したテクスチャを用意するだけ。処理も軽い)
参考：http://tsubakit1.hateblo.jp/entry/2016/06/30/073000

ToonShader(アニメ調の表現を行う。セルルックともいう)
参考：https://www.slideshare.net/Unite2017Tokyo/unite-2017-tokyounity-75800622
参考：http://unity-chan.com/

Curved World(ロール状の地面を表現する)
参考： http://tsubakit1.hateblo.jp/entry/2017/06/20/234608

まとめ

シェーダーは難しいイメージがあるけど、
Surfaceシェーダーを使えば、敷居がぐっと下がる。
※光と影の計算をUnityが自動でやってくれるから

とは言えやっぱり難しいです。
そのかわりしっかり覚えれば、
表現の幅がめちゃくちゃ広がること間違いなし！

■参考サイト

公式マニュアル　シェーダーを書く
https://docs.unity3d.com/ja/540/Manual/ShadersOverview.html

公式マニュアル　シェーダーアセット
https://docs.unity3d.com/ja/540/Manual/class-Shader.html

公式マニュアル　サーフェイスシェーダーの例
https://docs.unity3d.com/ja/540/Manual/SL-SurfaceShaderExamples.html

公式マニュアル レンダリング順
https://docs.unity3d.com/jp/540/Manual/SL-SubShaderTags.html

公式マニュアル GPUインスタンシング
https://docs.unity3d.com/ja/540/Manual/GPUInstancing.html

○×つくろーどっとコム

http://marupeke296.com/UNI_main.html

凹みTips

http://tips.hecomi.com/entry/2014/03/16/233943

Unityのシェーダーセマンティクスまとめ

http://qiita.com/sune2/items/fa5d50d9ea9bd48761b2

■プロジェクト一式はここ

シェーダーをいじってみる前編
http://monolizm.com/sab/src/shader.zip

ご清聴ありがとうございました