



# Unityはじめるよ

## ～光と影～

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。

# 光と影

光と影を意識してセットアップした世界と  
なにも考えずにただ表示した世界とでは  
見た目で与える印象が大きく異なる。

# メリットデメリット

光に関する処理、影に関する処理は

総じて処理やメモリ、ストレージの負荷が高い。

最高なクオリティとパフォーマンスになるよう

実行環境に合わせて、

取り捨てる選択をする必要がある。

# やりたいこと

新興国を含む海外も対象にしたスマホゲームを作りたい。

そのためには、  
低スペックのAndroid端末を考慮する必要がある。

なので、CPU性能、GPU性能、通信能力、メモリ使用量、  
ストレージ使用量とも抑えめにしなくてはならない。

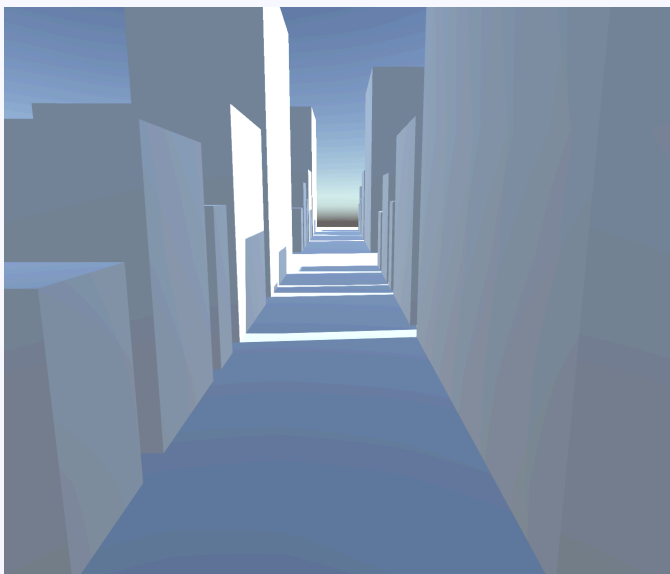
その上で、そこそこの見た目を実現したい。

光と影は見た目のクオリティを上げるのに効果的なので  
どんな方法があるかを調査&テストしている。

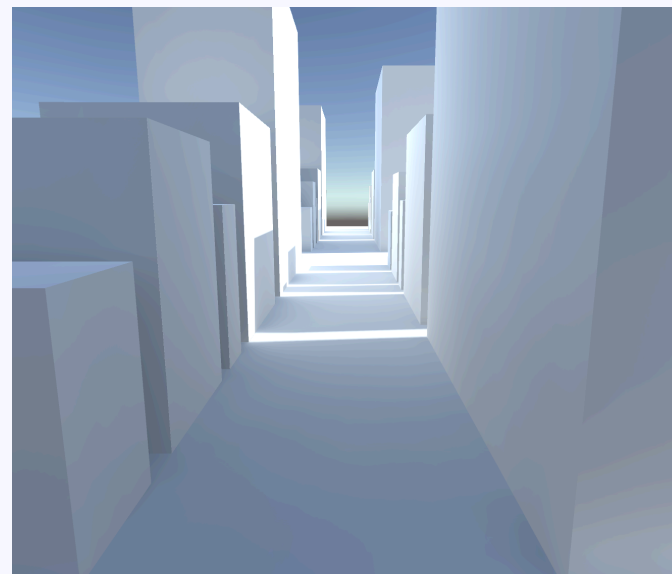
# ライトマップ

# ライトマップって？

高負荷である光と影の計算を事前に行い、その情報をテクスチャに書き込んでおき、実行時の負荷を軽くする方法。



リアルタイム



ライトマップ

# メリット・デメリット

## メリット

- ・ 実行時の負荷が少ない
- ・ 高品質

## デメリット

- ・ スタティックなオブジェクトにしか使えない
- ・ ライトマップ用テクスチャ分のメモリとストレージを消費
  - ※広大なステージの場合など、テクスチャのサイズを大きくしてクオリティを保つか、テクスチャのサイズを抑えてクオリティを下げるかの検討が必要。

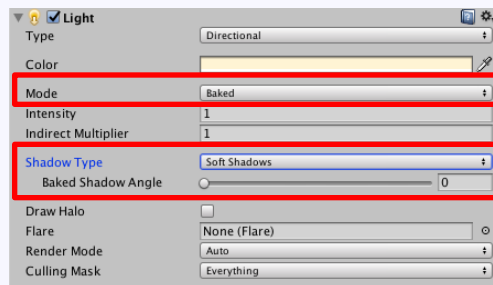
# ライトマップ設定方法

※2017/05/13現在Unity5.6のライティングのドキュメントが整備されていない状態なので、5.5以前の情報と、触ってみた結果の情報なので、正しい情報ではない可能性があります。

- ライトオブジェクトの設定
- ライトマップを適用したいオブジェクトの設定
- ライティングウィンドウでの設定
- プロジェクト設定(QualitySettings)
- ハマリ注意Tips



# ライトオブジェクトの設定



## Mode

### RealTime

ライトマップは使わずにリアルタイムに計算を行う

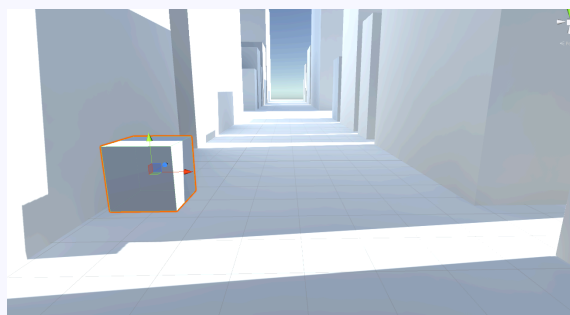
### Mixed

直接光はリアルタイムに、間接光はライトマップを使う

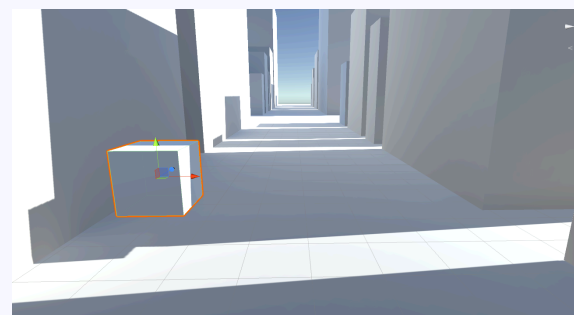
### Baked

ライトマップのみを使う（リアルタイムの影は落ちなくなる）

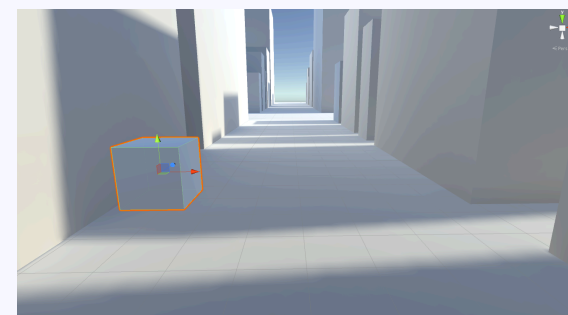
※Modeを変更してもライトマップを焼きなおさないと反映されないので注意！



RealTime



Mixed



Baked

オレンジ枠がついてるオブジェクトはスタティックでないオブジェクト

# ShadowType リアルタイムで計算される直接光での影生成設定

NoShadows

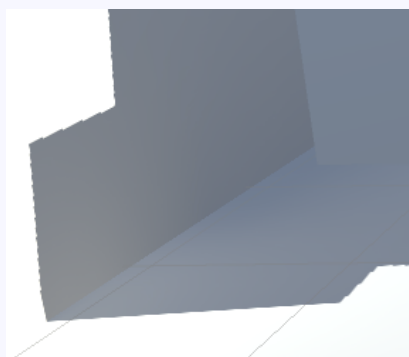
このライトでは影を生成しない

HardShadows

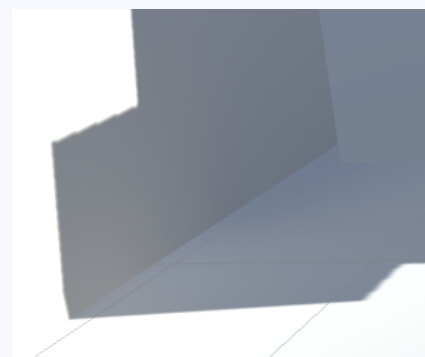
影の輪郭がくっきりする (処理負荷小さい)

SoftShadows

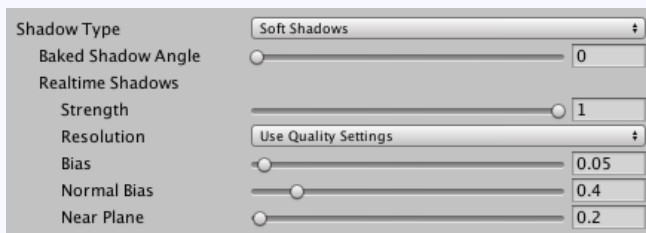
影の輪郭がぼんやりする (処理負荷大きい)



HardShadows



SoftShadows



Baked Shadow Angle  
Strength  
Resolution  
Bias

影の縁部分に疑似的なぼかし処理を追加  
影の強さ (濃さ)  
影表示用テクスチャの解像度  
影の位置と詳細度の微調整値

Normal Bias  
Near Plane

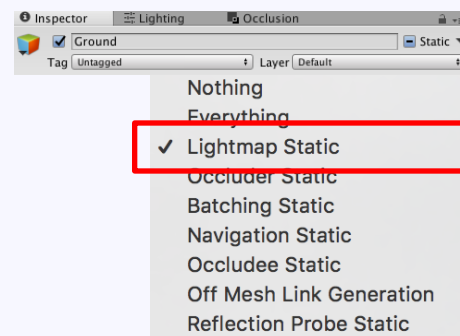
シャドウアクネという縞模様が出てしまうのを防ぐ役割  
影の位置と詳細度の微調整用 (値が大きすぎると影が小さくなる)  
影をレンダリングするときに前方クリップ面の値の設定  
ライトとの距離がこの距離より短いオブジェクトは投影しない

公式ドキュメントから一部抜粋 <https://docs.unity3d.com/jp/540/Manual/ShadowOverview.html>

# ライトマップを適用したいオブジェクトの設定

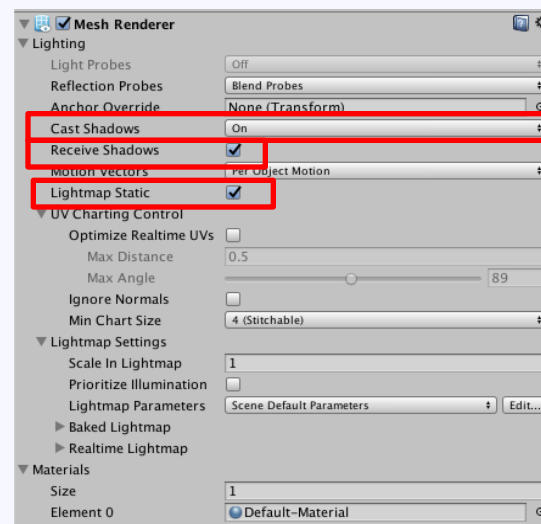
スタティクなオブジェクトでないとライトマップは利用できないので、インスペクタの右上のStaticから、

**LightmapStatic**のチェックを入れる。  
または、MeshRendererコンポーネントの  
**LightmapStatic**のチェックを入れる。



**CastShadows**をONにすることで、影を落とすようになる。

**ReceiveShadow**のチェックを入れることで、影を受けるようになる。



# ライティングウィンドウでの設定

Environment

環境光の設定

Realtime Lighting

リアルタイムライティングの設定

Mixed Lighting

リアルタイムとライトマップを  
同時に使う場合の設定

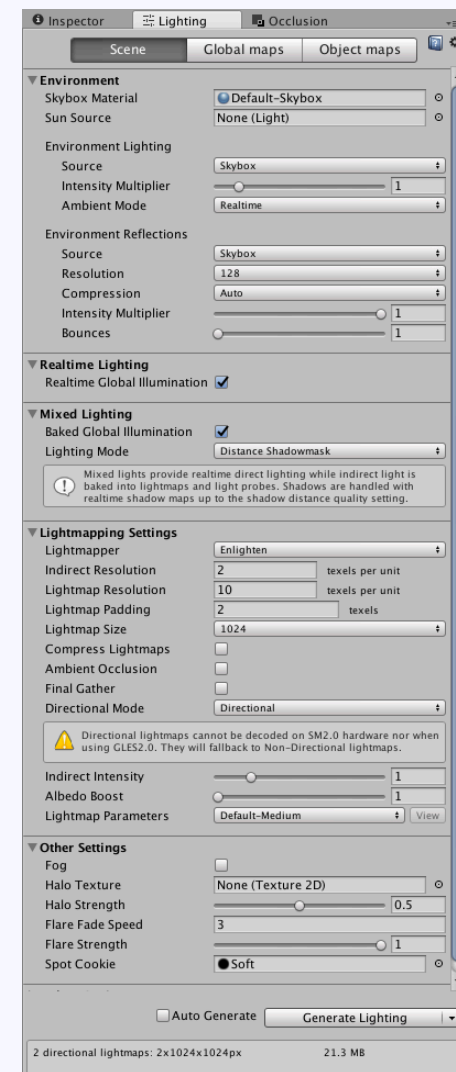
Lightmapping Settings

ライトマップの設定

Other Settings

Halo、Fog、Flareの設定

この中のLightmapping Settingsをもう少し詳しく見てみる。



# Lightmapping Settings (ライトマップに関する設定項目)

## Lightmapper

Enlighten

Progressive

ライトマップを焼く装置の選択

Unity5世代から使えるようになった、ライティングエンジン  
シーンビューに表示されている範囲を優先的にベイク (Editorのみ)  
画質設定 (数値が大きいほど高品質。ただしベイクに時間がかかる)

## Indirect Resolution

## Lightmap Resolution

ライトマップの解像度 (数値が大きいほど高品質だが、  
ベイクの時間がかかり、また大きなテクスチャ領域を必要とする)  
テクスチャに焼いた際の、画像同士の間隙の設定。

## Lightmap Padding

## Lightmap Size

ライトマップテクスチャの解像度 (一枚のテクスチャに収まらない  
場合は、複数のテクスチャが生成される)

## Compress Lightmaps

## Ambient Occlusion

## Final Gather

## DirectionalMode

Non-Directional

Directional

ライトマップテクスチャを圧縮テクスチャにするかフラグ  
アンビエントオクルージョンを使い高品質にする (ベイク時間増)  
光のバウンスの計算解像度を上げ高品質にする (ベイク時間増)  
光の計算に法線と反射の情報を持たせるかの設定

法線と反射情報を持たせない

法線と反射情報を持たせ高品質に (ストレージ・メモリ使用量増)

## Indirect Intensity

## Albedo Boost

## Lightmap Parameters

光の強さ (シーン全体の明るさが変化しているように見える)

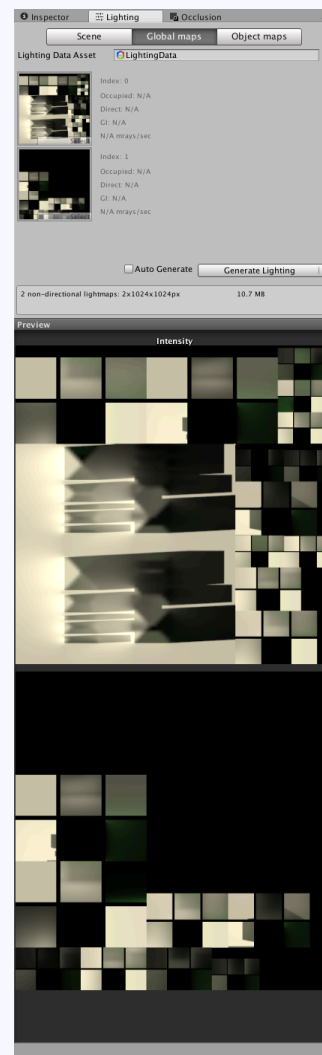
影以外の明るさが変化しているように見えるが・・・

明るい部分の明るさが変化しているように見えるが・・・

## 焼かれたライトマップを確認

Lightingウィンドウの Global mapsで

焼かれたテクスチャの中身と  
ファイルサイズを確認することができます。



# プロジェクト設定(QualitySettings)

上部メニューのEdit > ProjectSettings > Quality  
からQualityセッティングウィンドウを開く

## Shadowの項目

### Shadows

- Disable 影を表示しない
- Hard Shadow Only ハードシャドウのみ利用
- Hard And Soft Shadows ハードな影もソフトな影も利用する

### Shadow Resolution

影生成で利用するテクスチャの解像度設定

### Shadow Projection

- Close Fit シーン全体から影の描画サイズを決める
- Stable Fit カメラに写っている範囲から影の描画サイズを決める

### Shadow Distance

カメラから影が見える距離。この範囲外の影は描画されない。

### Shadow Near Plane Offset

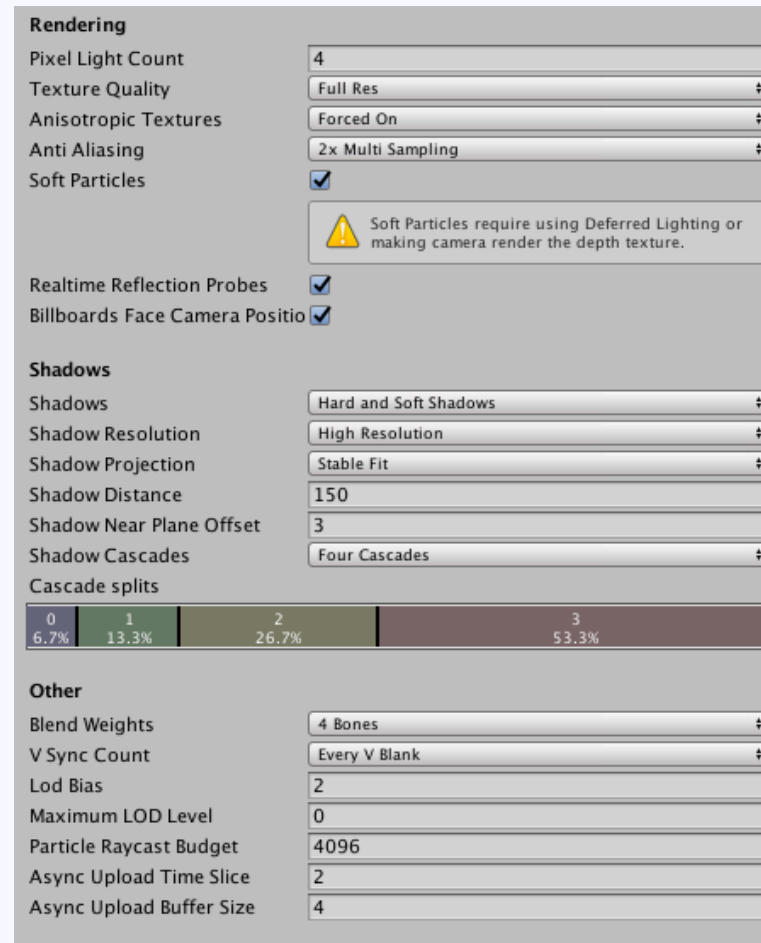
シャドウパンケーキ（丸影）によって歪んでいる  
大きな三角形をはっきり見えるように平面に近い影を弱める  
(と書いてあったがよくわからんね)

### Shadow Cascades

カメラからの距離に応じてテクスチャの何%を利用するか決める際の  
何分割するか設定

### Cascade splits

カメラからの距離に応じてテクスチャの何%を利用するか決める際の  
分割の割合設定



# ハマり注意Tips

- ・シェーダの種類により、影が落ちる落ちないがある  
PixelLighting系を使えばOK。

VertexLit系は頂点単位での処理になるので精度と表現の幅が狭い  
Unlit系は影が出ない

- ・自作モデルやアセットストアのモデルモデルデータを使うと、  
影が出なかったり、おかしい影になる場合がある。

そんな時は、モデル読み込み設定の

**Model > Generate Lightmap Uvs**

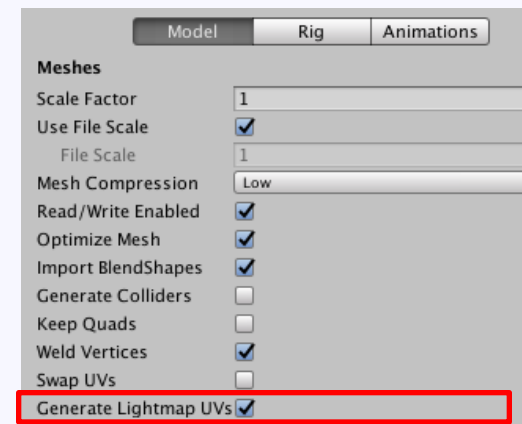
にチェックを入れ読み込み直すと直る場合がある。

それでも治らない場合はモデリングソフト上で、  
テクスチャのUV設定を行うと直るかも。

- ・ハードによって使える機能が異なる。

モバイル端末の大部分では、

**Lightmapping Settings**の**DirectionalMode**を**Non-Directional**でないと動かなかったり。

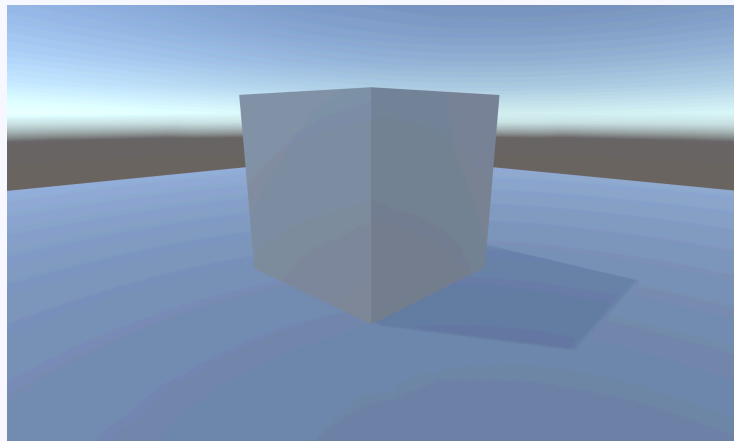




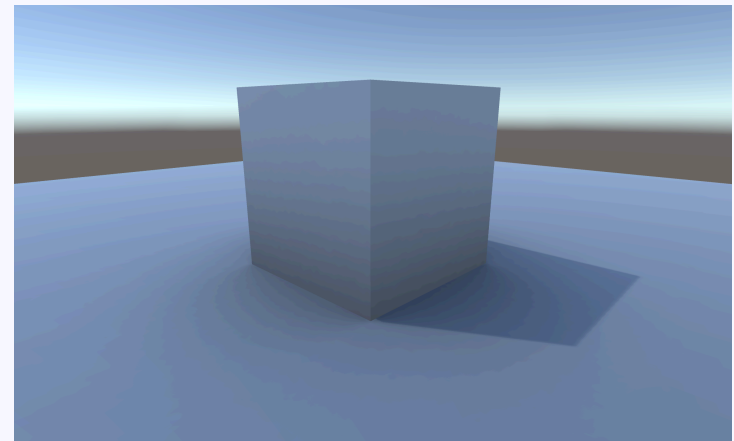
# AO (Ambient Occlusion)

# AOって？

現実世界では、建物と地面が接している付近や、穴、折り目などの入り組んだ部分が互いに接近していると、環境光が遮られて暗く見える。それを表現する処理。  
物を地面に置いた時の接地感が増す効果もある。



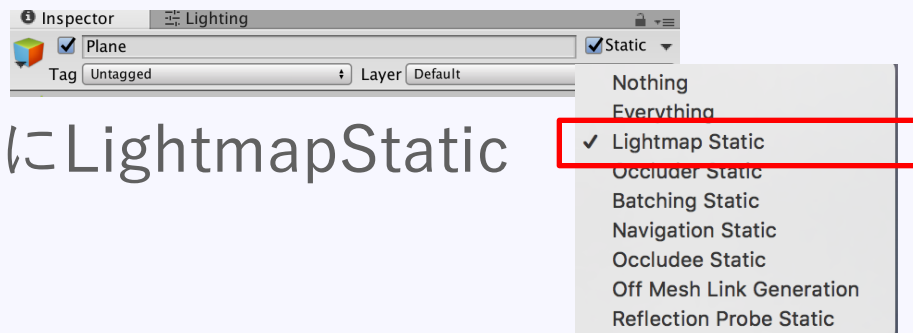
AOなし



AOあり

# 設定方法

適用したいオブジェクトにLightmapStatic  
のチェックを入れる



Window > Lighting > BakedGI > AmbientOcclusion  
のチェックを入れ、Bakeする

## MaxDistance

光源計算を行う為のレイを飛ばす距離。

## Indirect

間接光に照らされた時のAOの強さ。

値を大きくすると入り組んだ箇所付近の面が暗くなる。

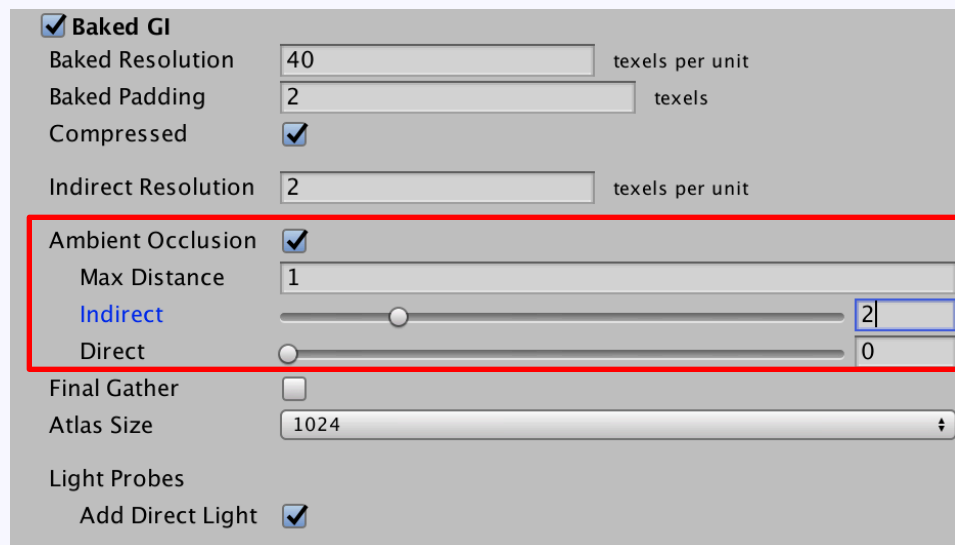
間接照明にのみAOをつける方が現実的な絵となる。

## Direct

直接光に照らされた時のAOの強さ。

値を大きくすると入り組んだ箇所付近の面が暗くなる。

現実的な絵にはならないが芸術的な絵を作りたいときに使う。



# メリット・デメリット

AOはライトマップに焼くことで利用できる。

※なので、メリットデメリットはライトマップと同じ

## メリット

- ・ 実行時の負荷は少ない
- ・ 高品質

## デメリット

- ・ スタティックなオブジェクトにしか使えない
- ・ ライトマップ用テクスチャ分のメモリとストレージを消費
  - ※ライトマップを使う前提ならAOを使うデメリットはない
  - ※広大なステージの場合など、テクスチャのサイズを大きくしてクオリティを保つか、テクスチャのサイズを抑えてクオリティを下げるかの検討が必要。

# ライトマップ使いたくないよ

って場合はどうするか。

考えてみた結果、メッシュ側に情報を持たせてはどうか。

Unityのシェーダーは頂点カラーを使っていない。

なので、メッシュの頂点カラー部分に他の情報を入れておくという方法は、割と使われる手法である。

メッシュの頂点カラーにAO情報（暗さ）を持たせておけば、AOに近いことが再現できるのではないだろうか。

実はこの方法、  
グラフィック性能が低かった初代PSでよく使われた。  
と言ってもAOとしてではないのだが、  
テクスチャ+頂点カラー(または法線)で表現の幅を拡げていた。

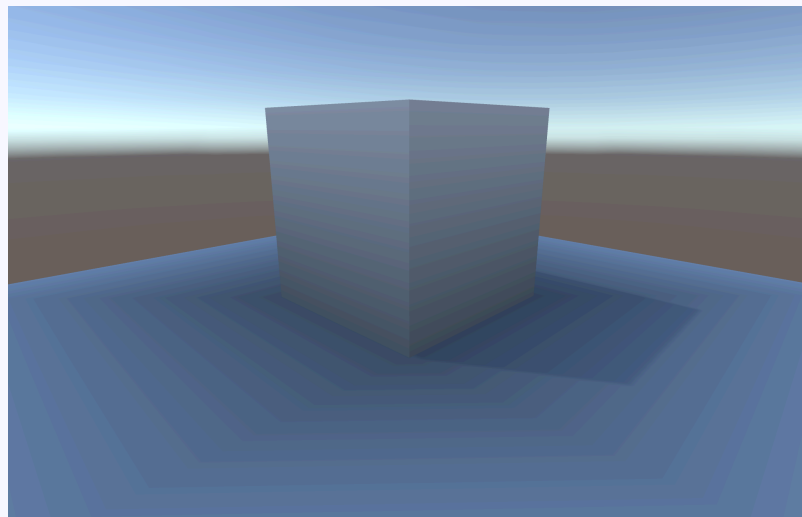


画像はPS史上最高グラフィッククオリティのレースゲーム『R4』

やってみた

Blenderで地面とCubeを作って頂点カラーをセット。  
頂点カラーの影響を受けるようにシェーダーをいじる。

※頂点カラーのR成分をAOに利用



そこそこの見栄えになったんじゃない？

# 頂点カラー用シェーダー

```
Shader "Custom/VertexColorAO" {
  SubShader {
    Tags { "RenderType"="Opaque" }
    LOD 200

    CGPROGRAM
    #pragma surface surf Lambert vertex:myvert
    #pragma target 3.0

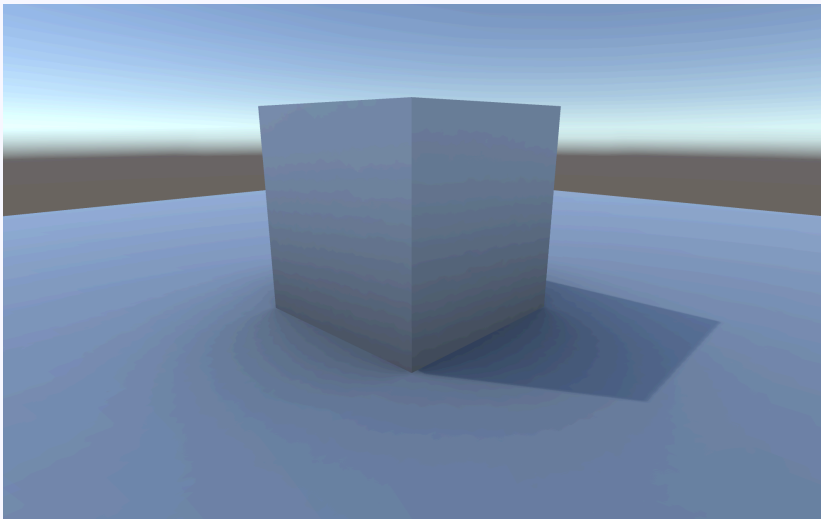
    struct Input {
      float4 vertColor;
    };

    void myvert(inout appdata_full v, out Input o){
      UNITY_INITIALIZE_OUTPUT(Input, o);
      o.vertColor = v.color;
    }

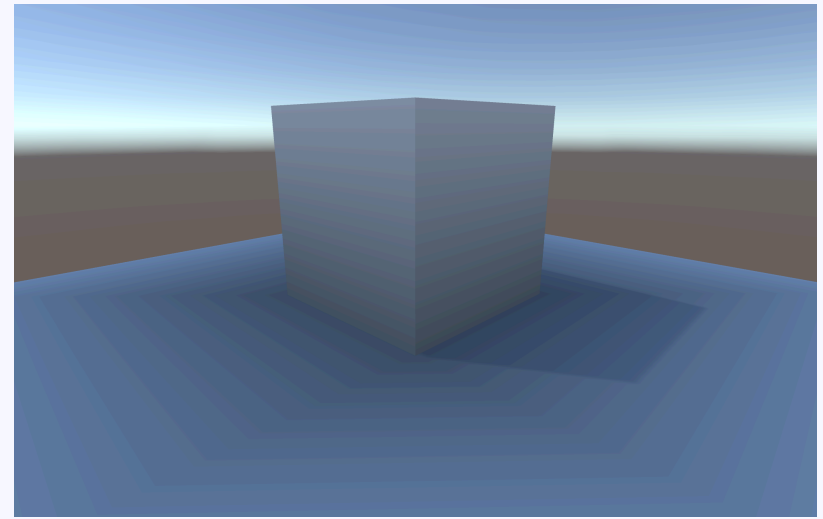
    void surf (Input IN, inout SurfaceOutput o) {
      o.Albedo = 1.0 * IN.vertColor.r;
    }
  }
  ENDCG
}
FallBack "Diffuse"
```



# 比較



BakedGI AO



頂点カラーでAO

# メリット・デメリット

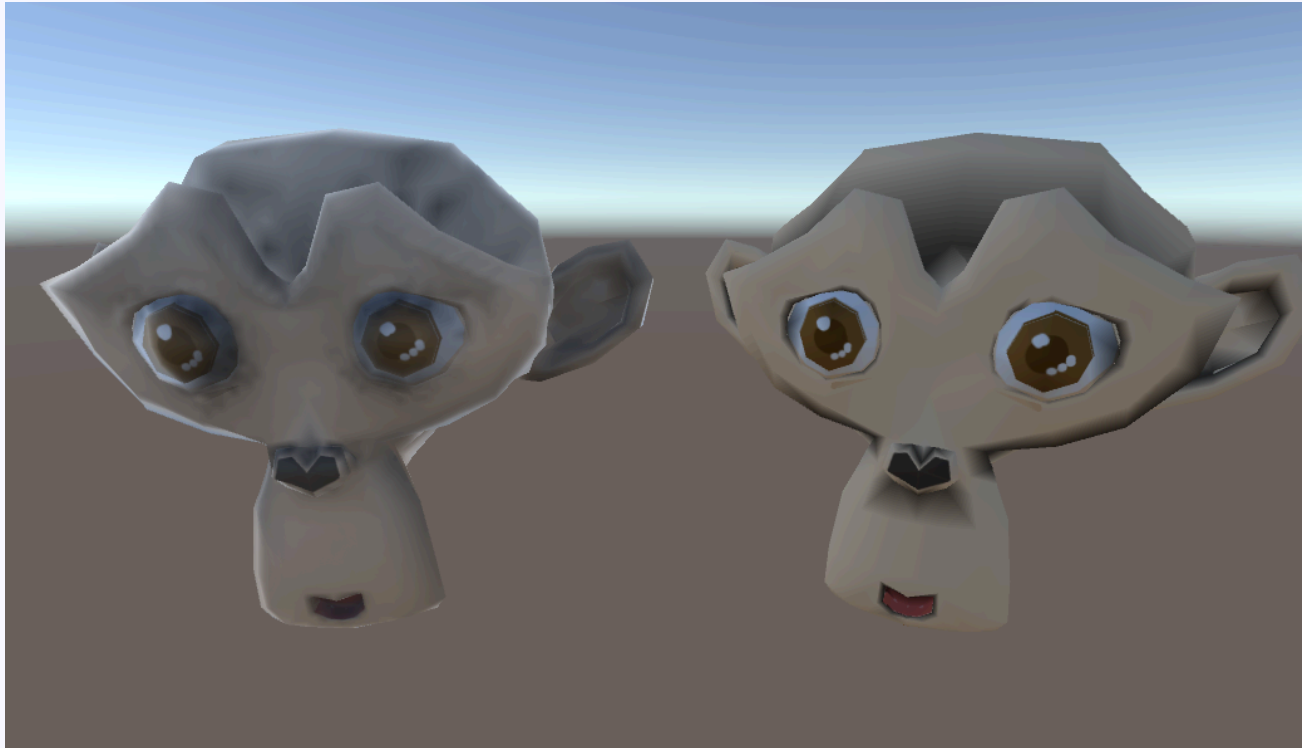
## メリット

- ・ 実行時の負荷はほぼなし（メモリ・CPU・GPUとも）

## デメリット

- ・ モデル作成時に頂点カラー情報を持たせなくてはならない
- ・ 地面側に影を落とすためには地面側のポリゴンの調整が必要
- ・ シェーダーを用意する必要がある
- ・ 見た目はBakedGIには敵わない
- ・ 動かないオブジェクトにのみ使うべき

## 別モデルでも比較してみた



BakedGI AO

頂点カラーでAO

サル顔だと、どんなに調整を頑張ってもBakedGI AOを綺麗にすることができなかった・・・

# 頂点カラー+テクスチャ+カラー用シェーダー

```
Shader "Custom/VertexColorTexture" {
    Properties {
        _ColorTint ("Tint", Color) = (1,1,1,1)
        _MainTex ("Base (RGB)", 2D) = "white" {}
    }
    SubShader {
        Tags { "RenderType"="Opaque" }
        LOD 200

        CGPROGRAM
        #pragma surface surf Lambert vertex:myvert finalcolor:mycolor
        #pragma target 3.0

        struct Input {
            float4 vertColor;
            float2 uv_MainTex;
        };

        fixed4 _ColorTint;
        void mycolor (Input IN, SurfaceOutput o, inout fixed4 color)
        {
            color *= _ColorTint;
        }

        sampler2D _MainTex;
        void myvert(inout appdata_full v, out Input o){
            UNITY_INITIALIZE_OUTPUT(Input, o);
            o.vertColor = v.color;
        }

        void surf (Input IN, inout SurfaceOutput o) {
            o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb * IN.vertColor.rgb;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```

# SSAO

## (Screen Space Ambient Occlusion)

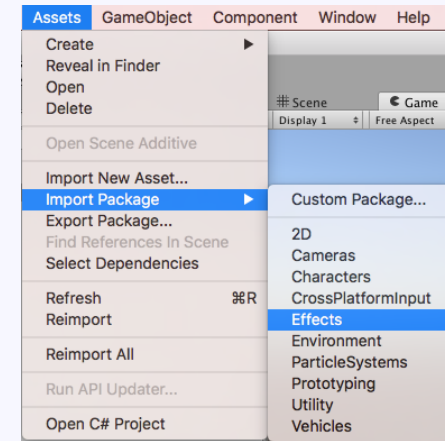
# SSAOって？

AOをスクリーンスペースで計算しようというもの。  
スクリーンスペースとは画面の範囲でということ。  
リアルタイムでAOを行うために、画面内に映っているもの  
に限定して計算範囲を狭めてAOを行なう手法。  
なので、計算時間は解像度に依存する。  
また、計算方法も簡易的となる。

# 設定方法

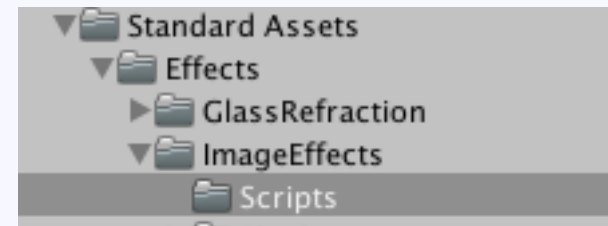
パッケージのインポート

**Assets > ImportPackage > Effects**



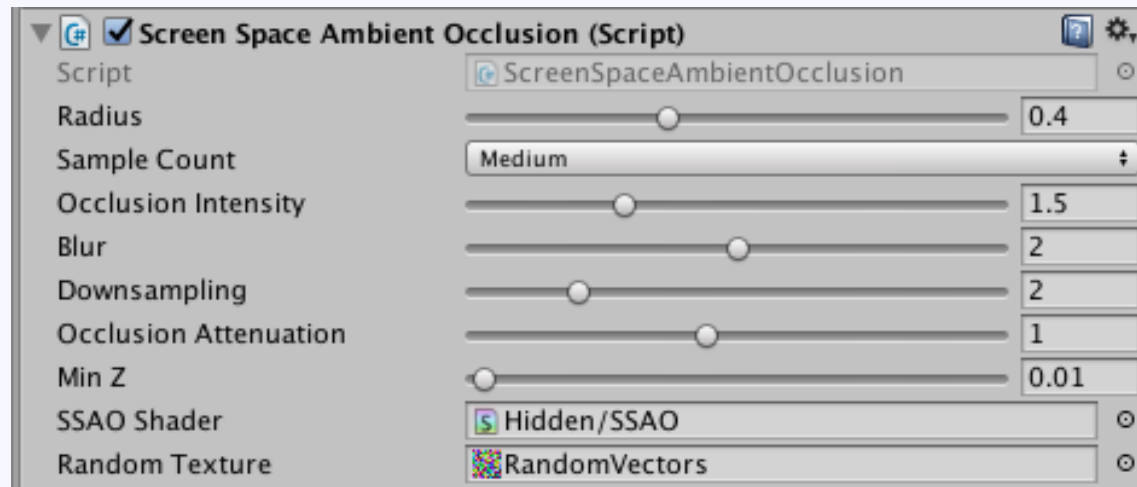
スクリプトをカメラに追加する。  
プロジェクトビューの、

**StandardAssets > Effects > ImageEffects > Script > ScreenSpaceAmbientOcclusion.cs**



# 設定項目 ※公式マニュアルから引用

プロパティ:	説明:
<b>Radius</b>	この半径より隣接距離が短い場合アンビエントオクルージョンを適用します
<b>Sample Count</b>	アンビエントオクルージョンのサンプリング回数 より高い値にすることで品質が上がりますが、オーバヘッドの負荷も上がります
<b>Occlusion Intensity</b>	アンビエントオクルージョンが適用する暗さの度合い
<b>Blur</b>	暗くする際に適用するにじみ にじみなし(値を0)にすることで処理速度は上がりますが、暗くした場所のノイズが増えます
<b>Downsampling</b>	計算を行う際の解像度(例えば値を2に設定した場合は画面解像度の半分) より大きな値にすることでレンダリング処理の時間を短縮できますが、品質は低下します
<b>Occlusion Attenuation</b>	オクルージョンを距離に応じて減衰させる強度
<b>Min Z</b>	画像の乱れが発生する場合はこの値を増加してください





## 実行結果



SSAO

動くとチラチラするがリアルタイムでAOが行えるのはグラフィッククオリティを上げる意味では効果が高い。

# メリット・デメリット

## メリット

- ・ キャラなどの動くモデルにAOを付けられる

## デメリット

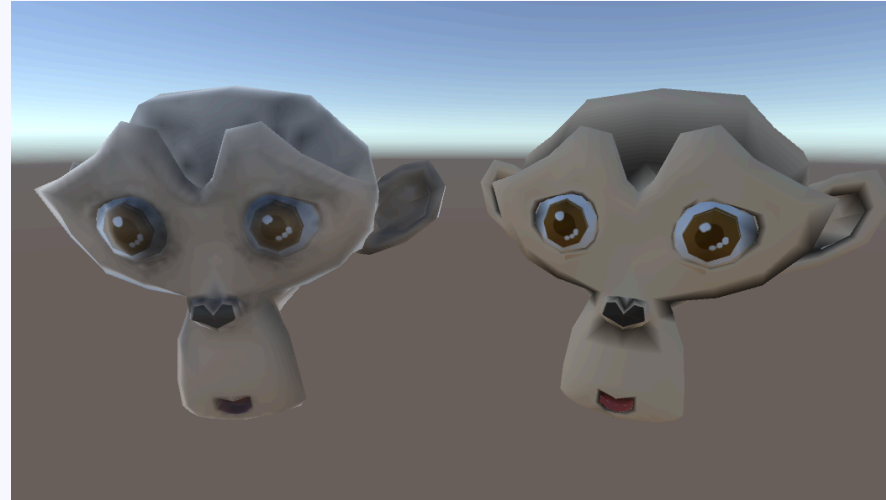
- ・ 処理が重い
- ・ 処理できるハードが限定される

シェーダーモデル3 と depth textures をサポートするグラフィックスカードが必要

## 比較



SSAO



BakedGI AO

頂点カラーでAO

モデルデータがよくなかったので  
SSAOとBakedGI AOが良い感じしないけど、  
背景オブジェクト系ならとても良く見えるようになる  
素晴らしい機能ですよ。

## 高速に動くという以外に、 頂点カラーAOを使うメリットがないか考えてみる

※光源を無視する簡易AOであることが前提

- たくさんのキャラで装備品を共通のテクスチャにしたい場合  
テクスチャ側に影を持たせるよりも、キャラモデル側に影を持たせておいたほうが、そのキャラに合う自然な影を作れる。



# アンビエントオクルージョンまとめ

BakedGIのAOを使えば、  
お手軽に見た目のクオリティを高めることができる。  
ただし、ライトマップのサイズをどうするかは調整が必要。

頂点カラーAOなら低負荷でAOっぽいことができる。  
ただし、モデルデータ作成時に工夫する必要がある、  
開発コストは増えてしまう。

Unityのエディタ拡張でシーン上のGameObject一式に、  
頂点カラーAOを適用できそうな気もする。これは要調査だな。  
できればモデルデータ作成時のコストがなくなる。

## ■ プロジェクト一式はこちら

光と影①(AO/SSAO)

<http://monolizm.com/sab/src/ao.zip>

## ■参考サイト

Unity公式ドキュメント

<https://docs.unity3d.com/Manual/LightingBakedAmbientOcclusion.html>

<https://docs.unity3d.com/jp/540/Manual/script-ScreenSpaceAmbientOcclusion.html>

ご清聴ありがとうございました