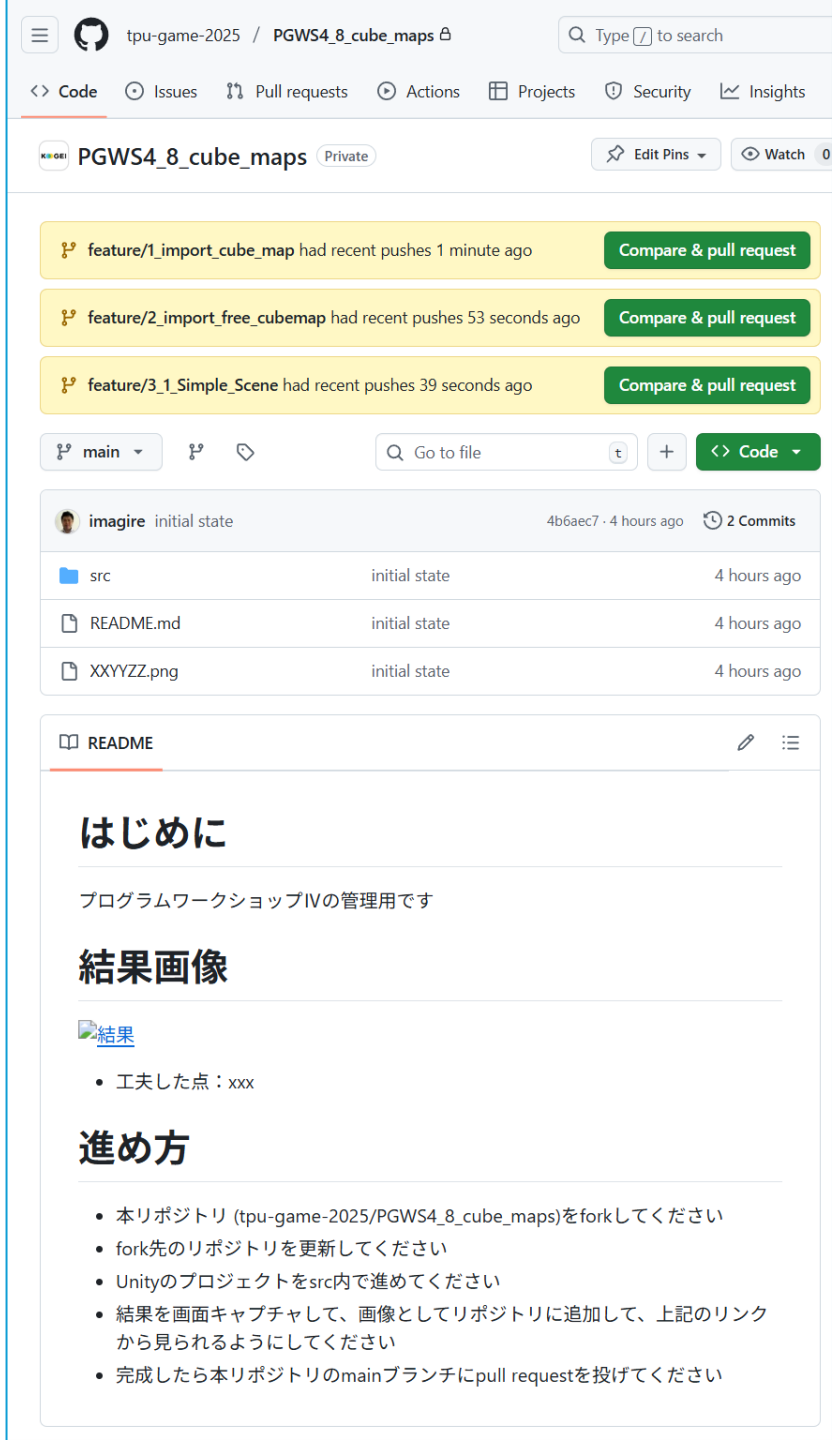
キューブマップ

2025年度 プログラムワークショップIV（8）

今回のリポジトリ

- https://github.com/tpu-game-2025/PGWS4_8_cube_maps



レジュメ

- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

レジュメ

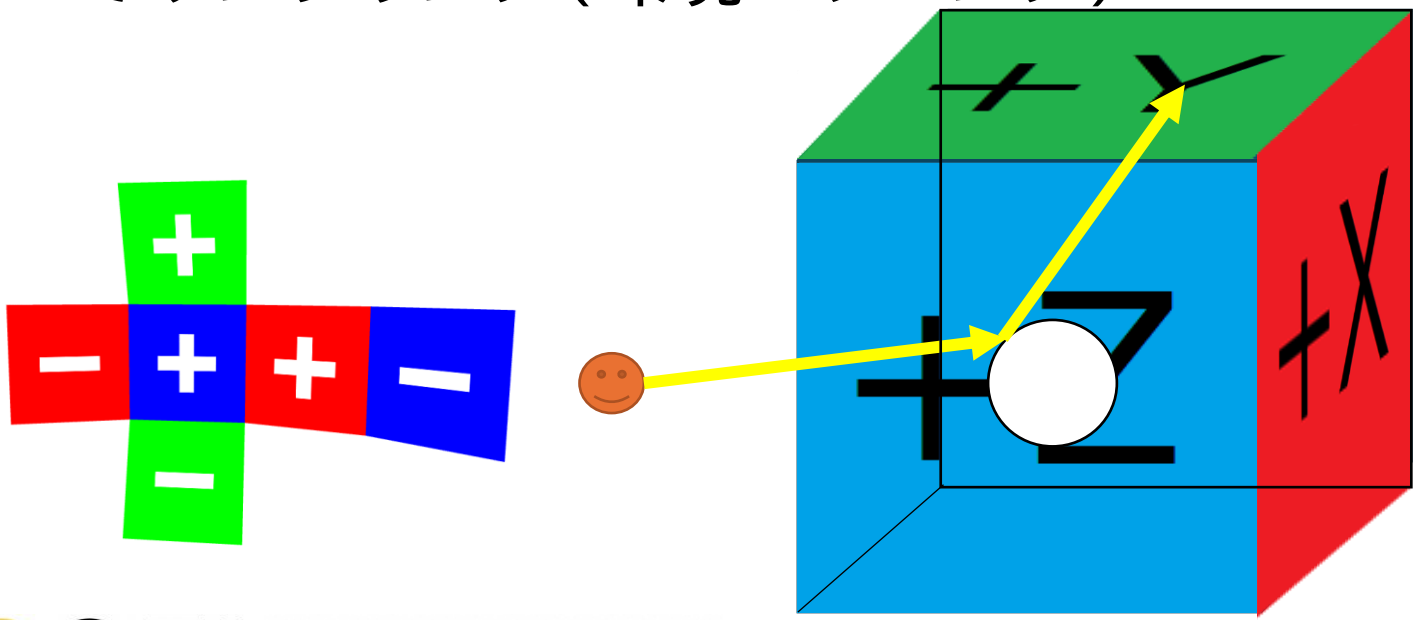
- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

今日の目的

- キューブマップについて学ぶ
- 周辺技術についても学ぶ

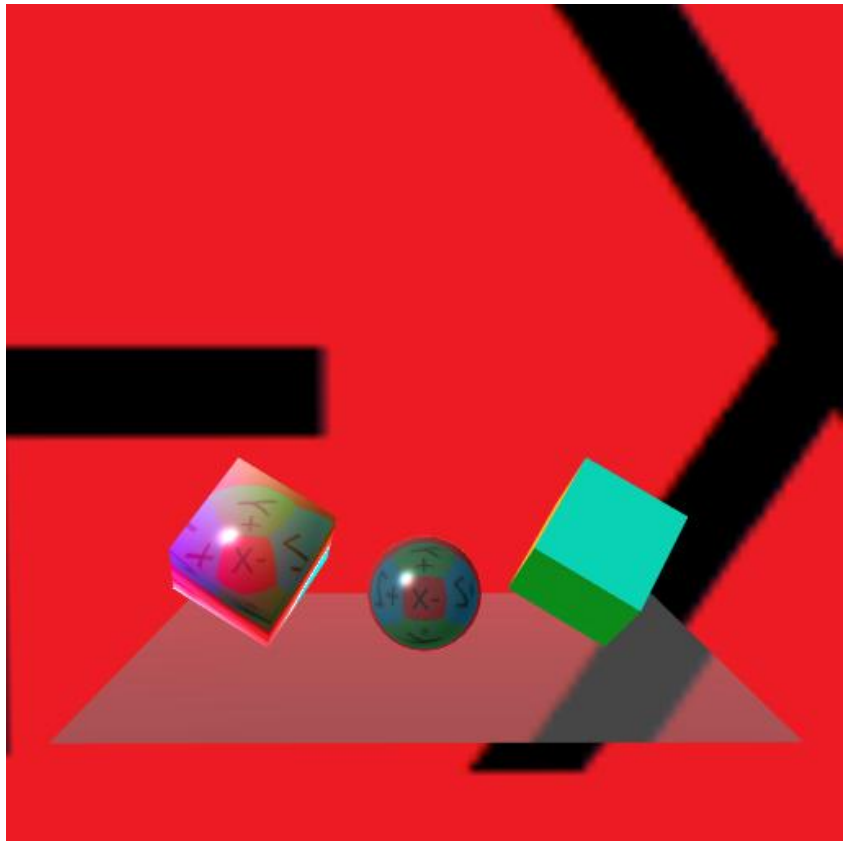
キューブマッピング

- キューブマップを用いたテクスチャマッピング
  - 立方体レイアウトで6枚のテクスチャが覆う
- 例えば、法線方向で反射した方向の向きでサンプリング(環境マッピング)



環境マッピングの考え方

環境マッピングの結果



レジュメ

- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

環境マップの使い方：環境(Environment)光源

- 非常に遠方にある景色(動いても変わらない)
  - カメラの位置を原点として、ワールド空間での各画素の方向をサンプリング
    1. ワールド原点中心に大きな箱(球でも可)を配置して、頂点位置から読み込む

$$Color = sampling(\vec{n})$$

$\vec{n} = W^{-1^T}\overrightarrow{n_l}$ ：ワールド空間での法線ベクトル

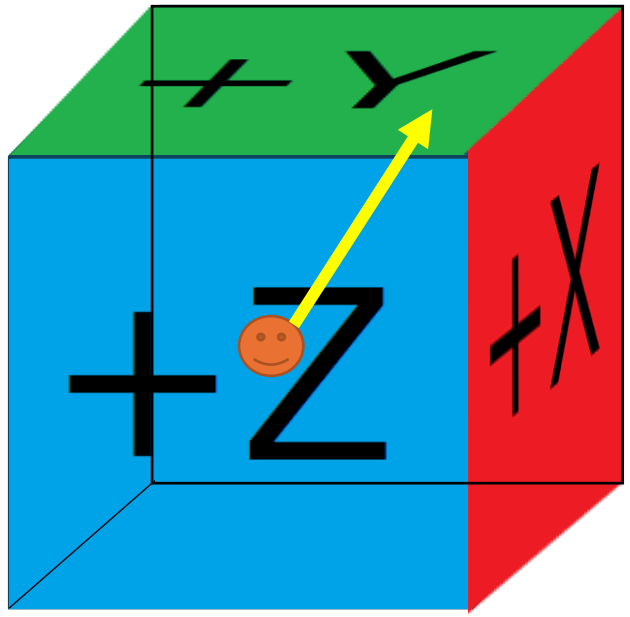
W: ワールド行列

$\overrightarrow{n_l}$: モデル空間での法線

    2. 全画面ポリゴンを描画し、クリップ空間での位置から
       ワールド空間での方向を逆算

$$\vec{v} = \vec{x}_{world} - V^{-1}\vec{o}$$

$$\vec{x}_{world} = \left(M_{Proj}M_{View}\right)^{-1}\vec{x}_{clip}$$



環境マップの使い方：鏡面反射

- ワールド空間での視線の反射方向でサンプリング
  - 正規化した方が行儀が良いが、正規化してもしなくても結果は同じ

$$Color = sampling(\vec{r})$$

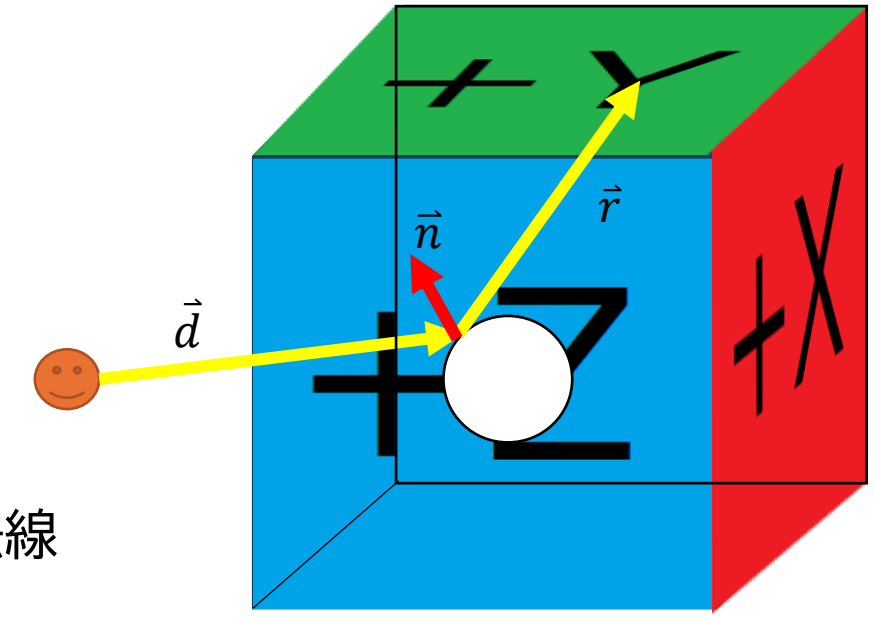
$$\vec{r} = \vec{d} - 2(\vec{d} \cdot \vec{n})\vec{n}$$

$$\vec{d} = W\overrightarrow{x_l} - V^{-1}\vec{o}$$ ：視線ベクトル

$$\vec{n} = W^{-1^T}\overrightarrow{n_l}$$ ：ワールド空間での法線ベクトル

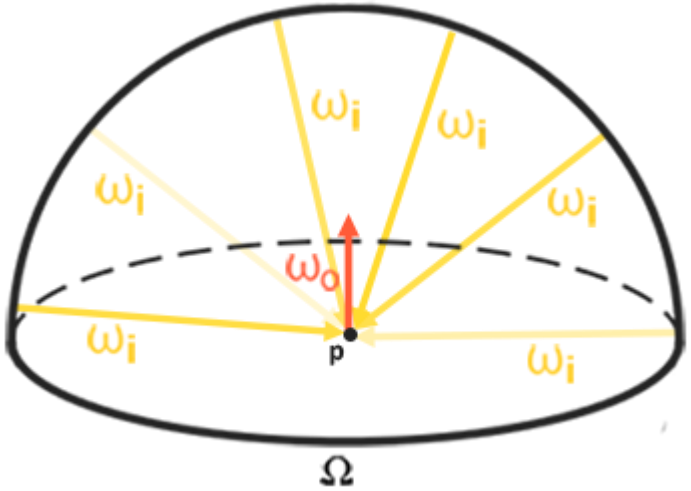
W：ワールド行列，　V：ビュー行列

$\overrightarrow{x_l}$：モデル空間での位置，$\overrightarrow{n_l}$：モデル空間での法線



IBL（Image Based Lighting）

- 鏡面反射指数が一定の場合や拡散反射では、事前にフィルタリングすることで、キューブマッピングだけで、全方位の拡散光やスペキュラ計算が可能となる



各画素において、半球方向に重みづけを行って、光の効果を事前に計算する

$$L_o(p,\omega_o) = k_d\frac{c}{\pi}\int_\Omega L_i(p,\omega_i)n\cdot\omega_i d\omega_i$$

$$\int_\Omega (k_s\frac{DFG}{4(\omega_o\cdot n)(\omega_i\cdot n)})L_i(p,\omega_i)n\cdot\omega_i d\omega_i$$

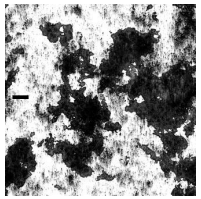
オリジナル

拡散反射

鏡面反射

よくある計算

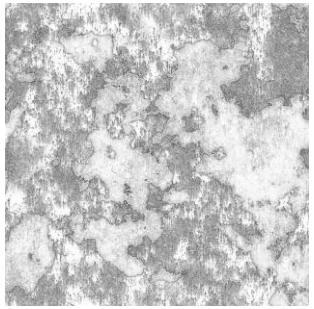
簡易的なエネルギー保存項

(1-F) * (1- )*

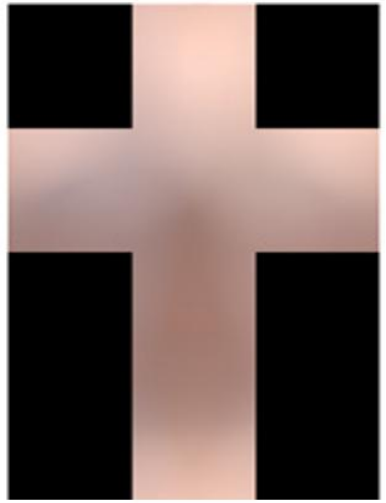
アルベド

*

AO



*

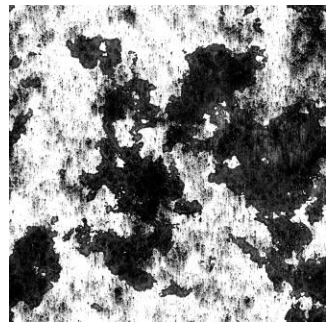


（法線ベクトル）

フレネル項

＋ F *

メタルネス



*

（反射ベクトル）

拡散項

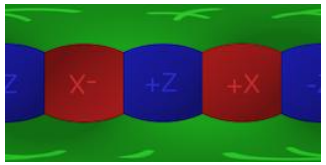
鏡面反射項

レジュメ

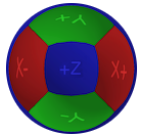
- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

Unityで使えるキューブマップ

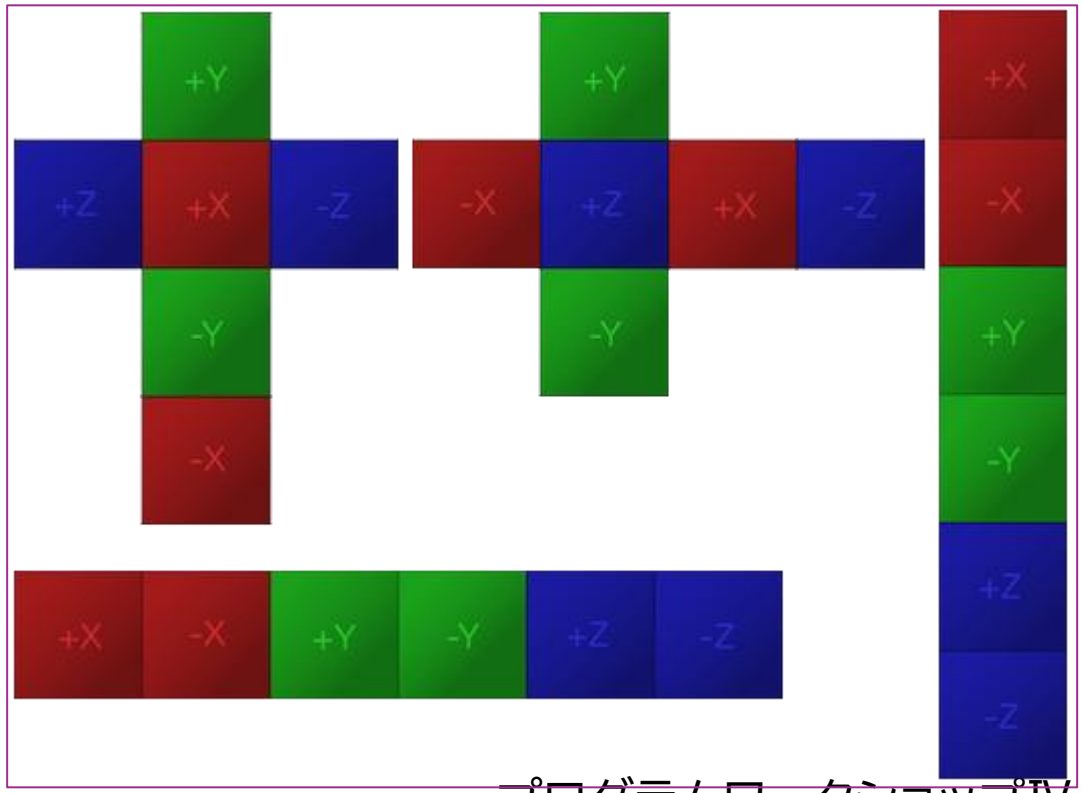
「大抵、Unity は自動的にそれらを検出します」
  - https://docs.unity3d.com/ja/current/Manual/class-Cubemap.html
- 6枚が一つになったレイアウト
- 緯度経度レイアウト



- 球状の環境マップ

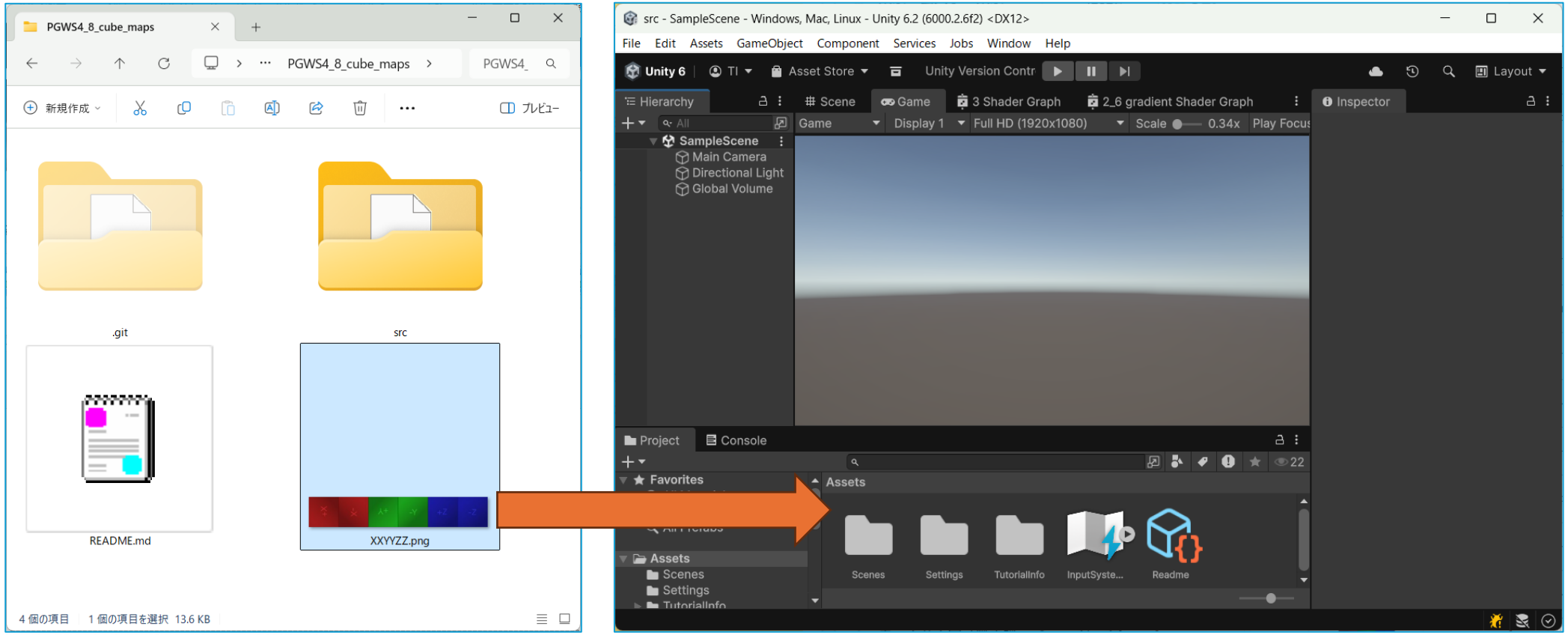


- 「6つの別々のテクスチャから
  キューブマップを作成することも
  できます。」
  - 古いキューブマップアセットとあるので、
    他の形式を使うのが無難



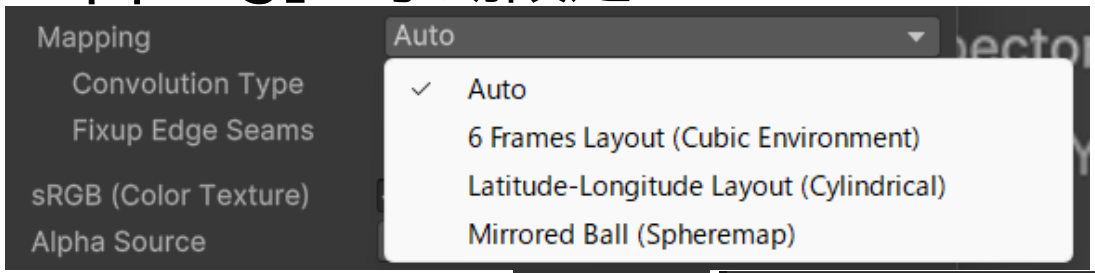
キューブマップのインポート

- キューブマップの画像ファイルをProjectウィンドウにD&D

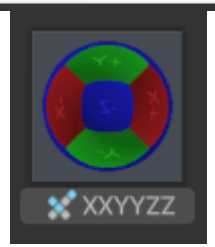


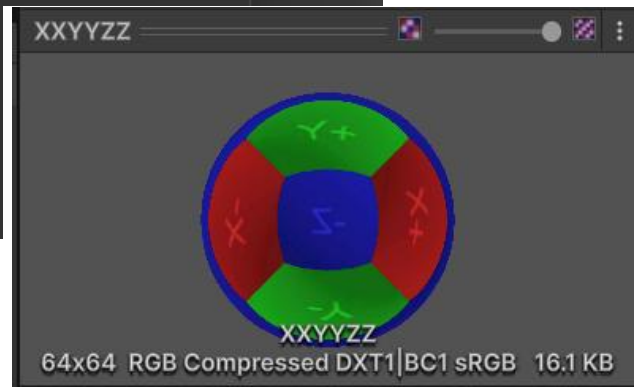
キューブマップの設定

- Texture Shapeは「Cube」
  - Apply後にキューブマップに変換
  - 種類の認識を間違えていたら「Mapping」を手動設定

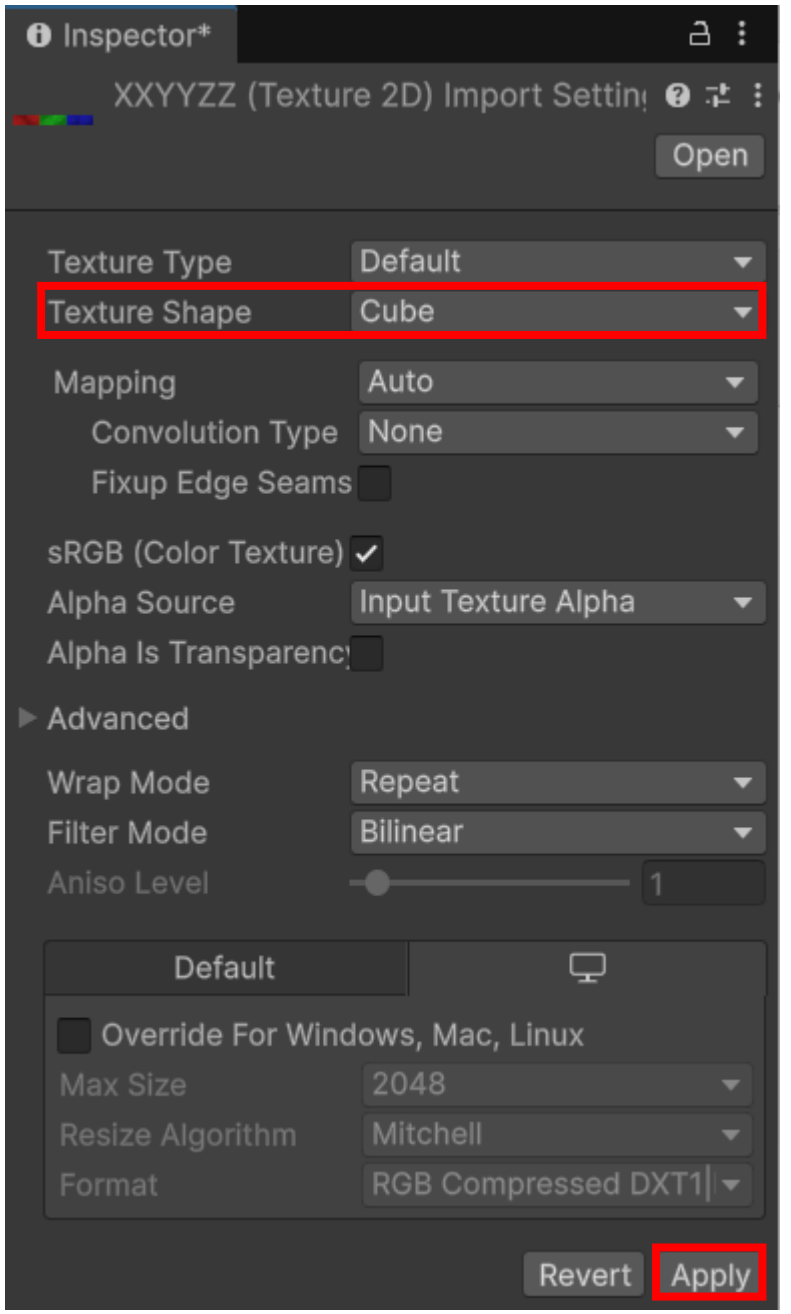


アイコンが2Fから球の表示になる





Inspectorの下のプレビューでぐるぐる回せればOK



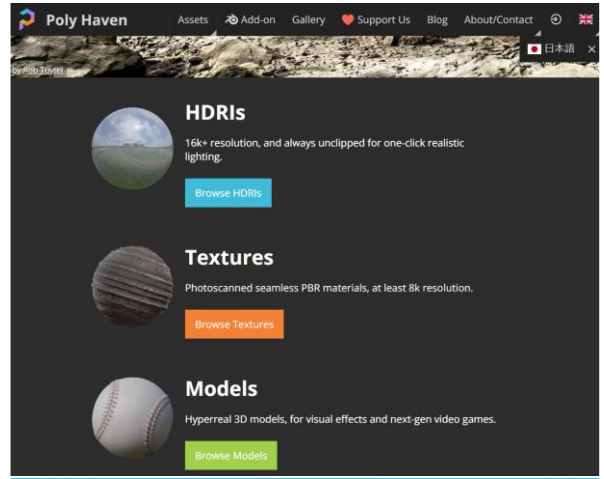
レジュメ

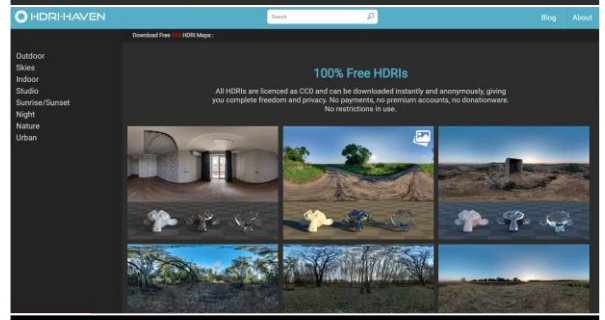
- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

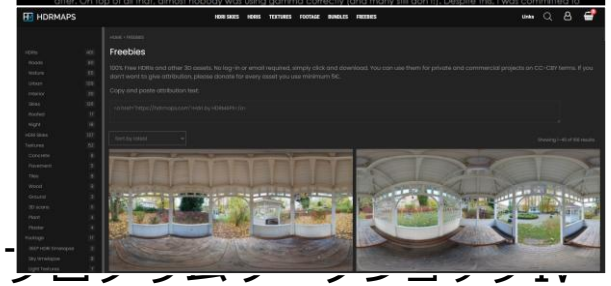
テクスチャデータ

配布しているサイトがある(ライセンスに注意)

- 前回も紹介
  - Poly Haven
    - https://polyhaven.com/
  - ambientCG
    - https://ambientcg.com/
  - cgbookcase.com
    - https://www.cgbookcase.com/
  - Share Textures
    - https://www.sharetextures.com/
- NoEmotion HDRs
  - https://noemotionhdrs.net/
- HDRI Maps:有料サイトだが無料の物もある
  - https://hdrmaps.com/freebies/

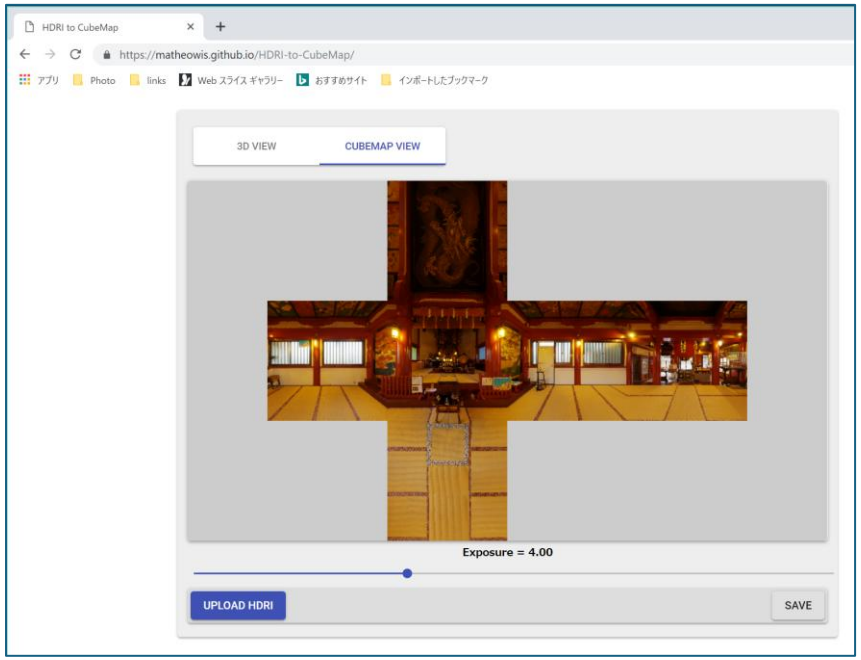


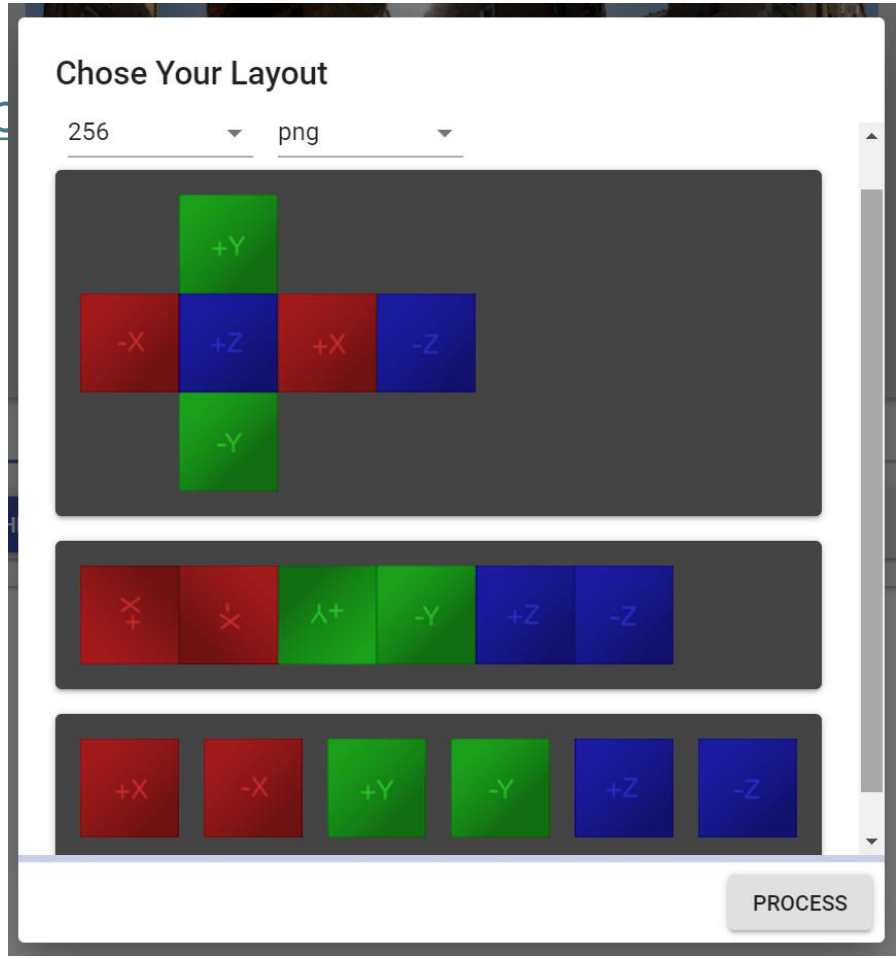




形式があっていなければ変換

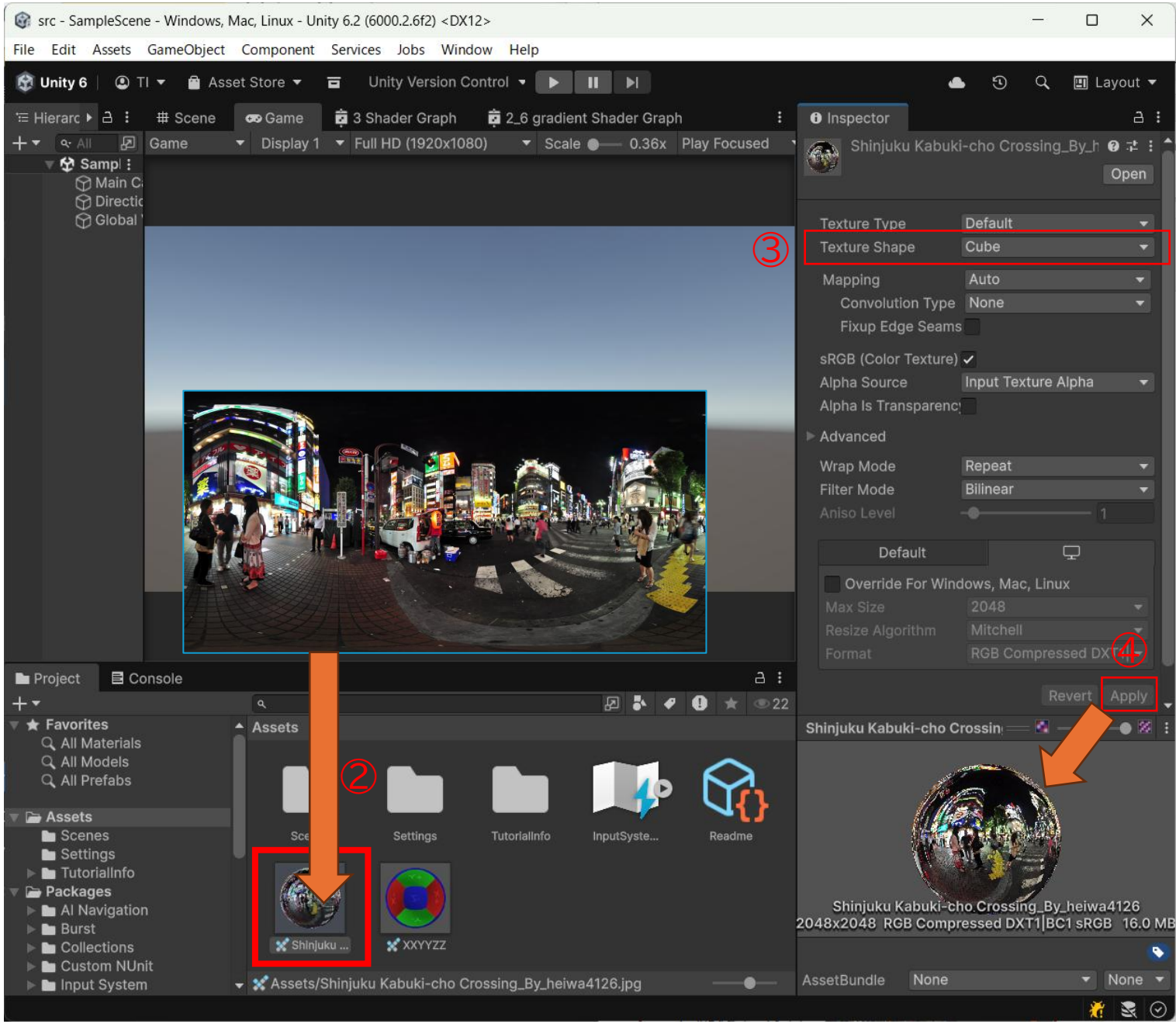
- 変換：HDRI-to-CubeMap
  - https://github.com/matheowis/HDRI-to-CubeMap
  - 変換サービス
    - https://matheowis.github.io/HDRI-to-CubeMap/





やってみよう

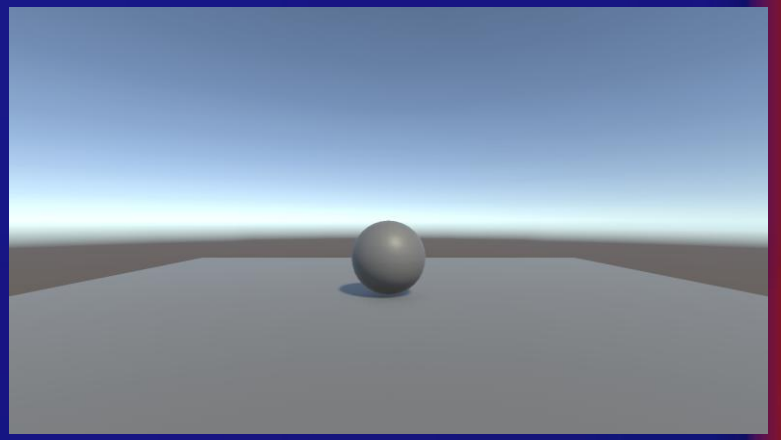
1. HDRIフリー画像をダウンロード
2. Unityにドラッグ＆ドロップする
3. 設定をキューブマップにする
   - 変更して、インスペクター外をクリックすると、変換のダイアログが出る
4. 「Apply」を押す



レジュメ

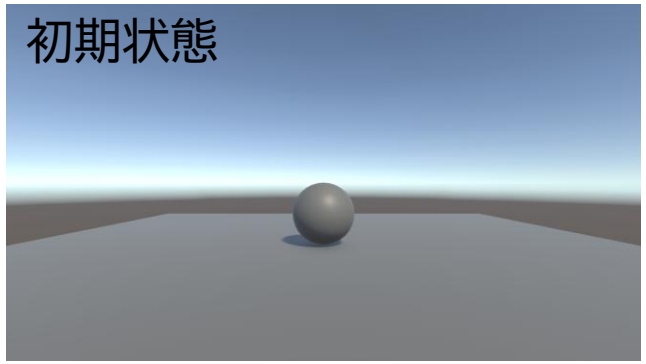
- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

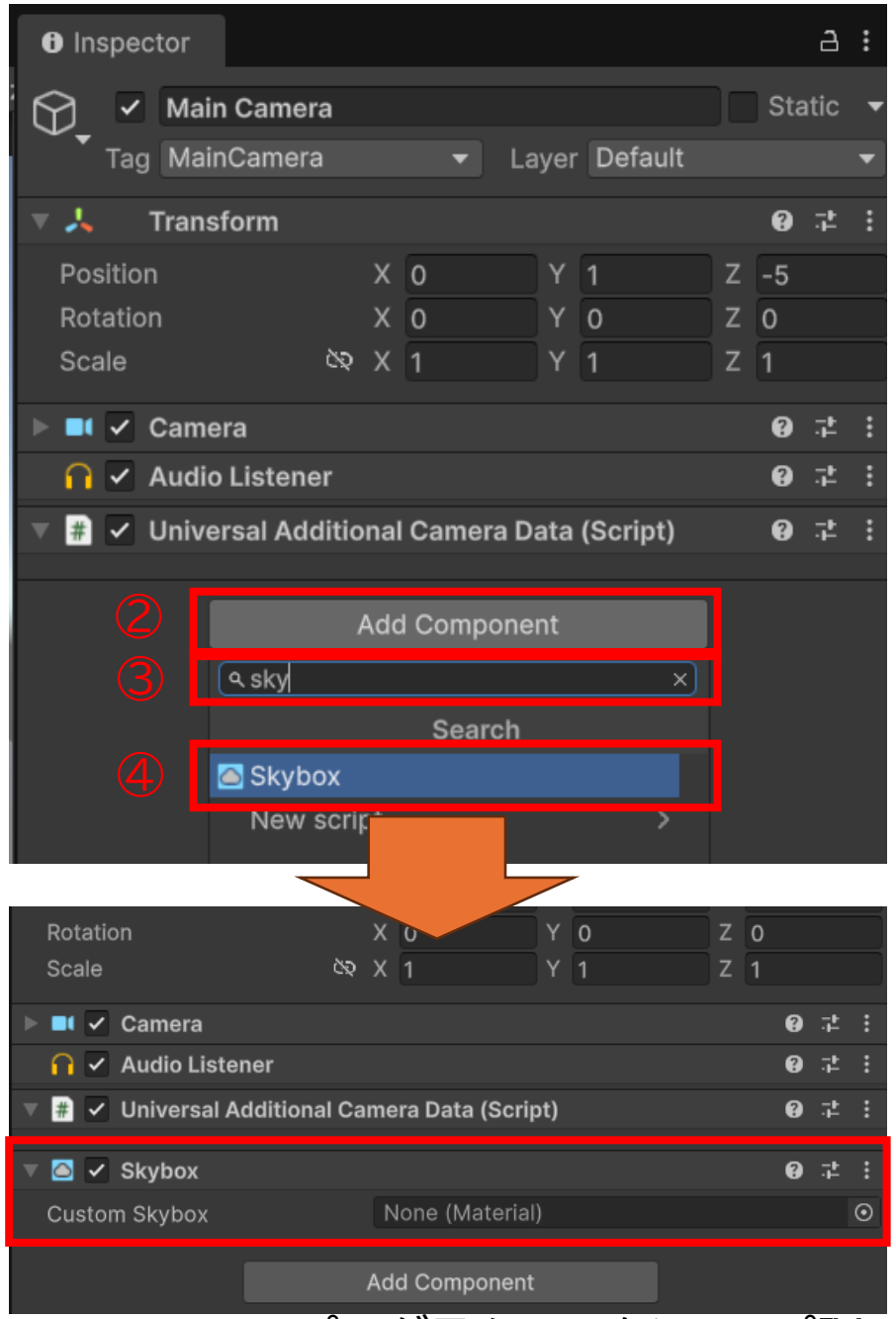
ゴール:1 Simple Sceneをこの状態にしよう！



天球の設定 その1

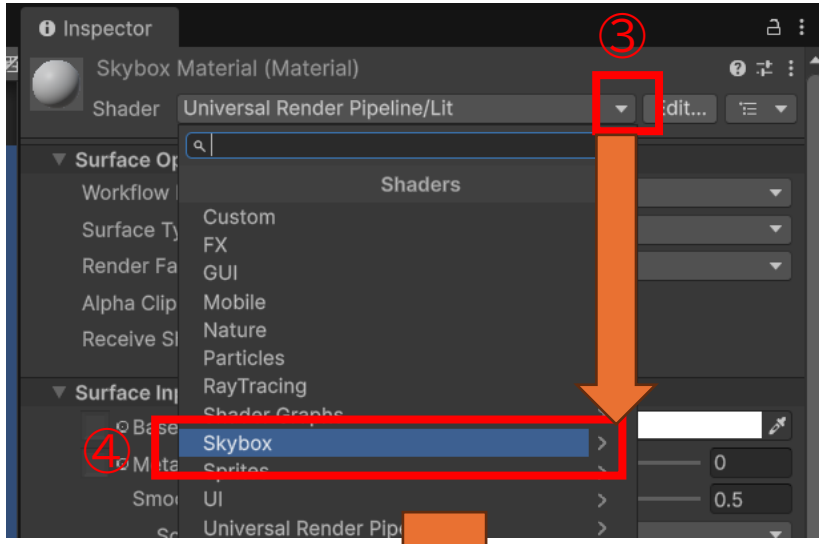
- カメラにスカイボックスコンポーネントを追加
  1. Hierarchyで「Main camera」を選択
  2. Inspectorの「Add Component」を選択
  3. 「sky」などと入力して候補を出す
  4. Skyboxを選択

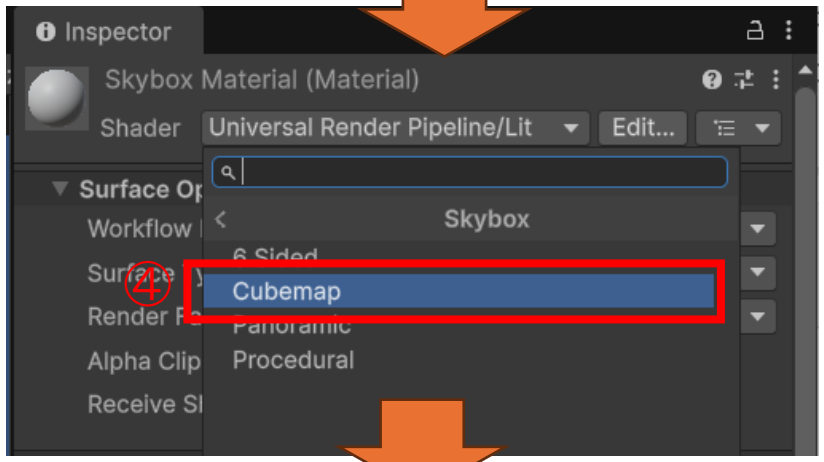


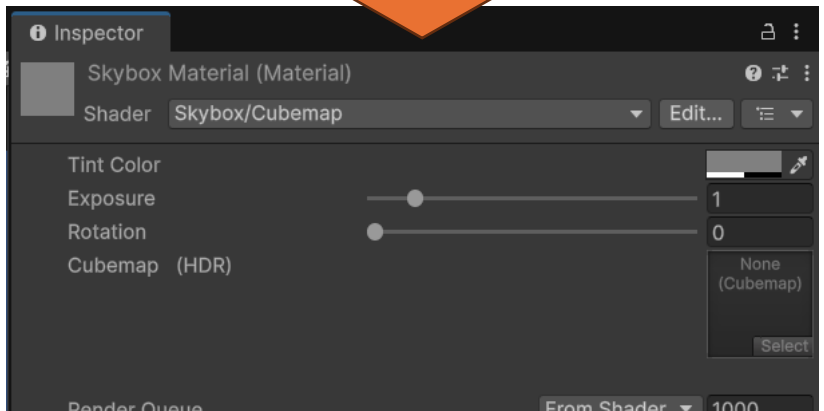


天球の設定 その2

- マテリアルを追加
  1. 名前を「Skybox Material」にする
- 定義済みのシェーダをアサインする
  2. Skybox Material の選択
  3. InspectorのShaderの▼を選択
  4. 「Skybox」→「Cubemap」を選択



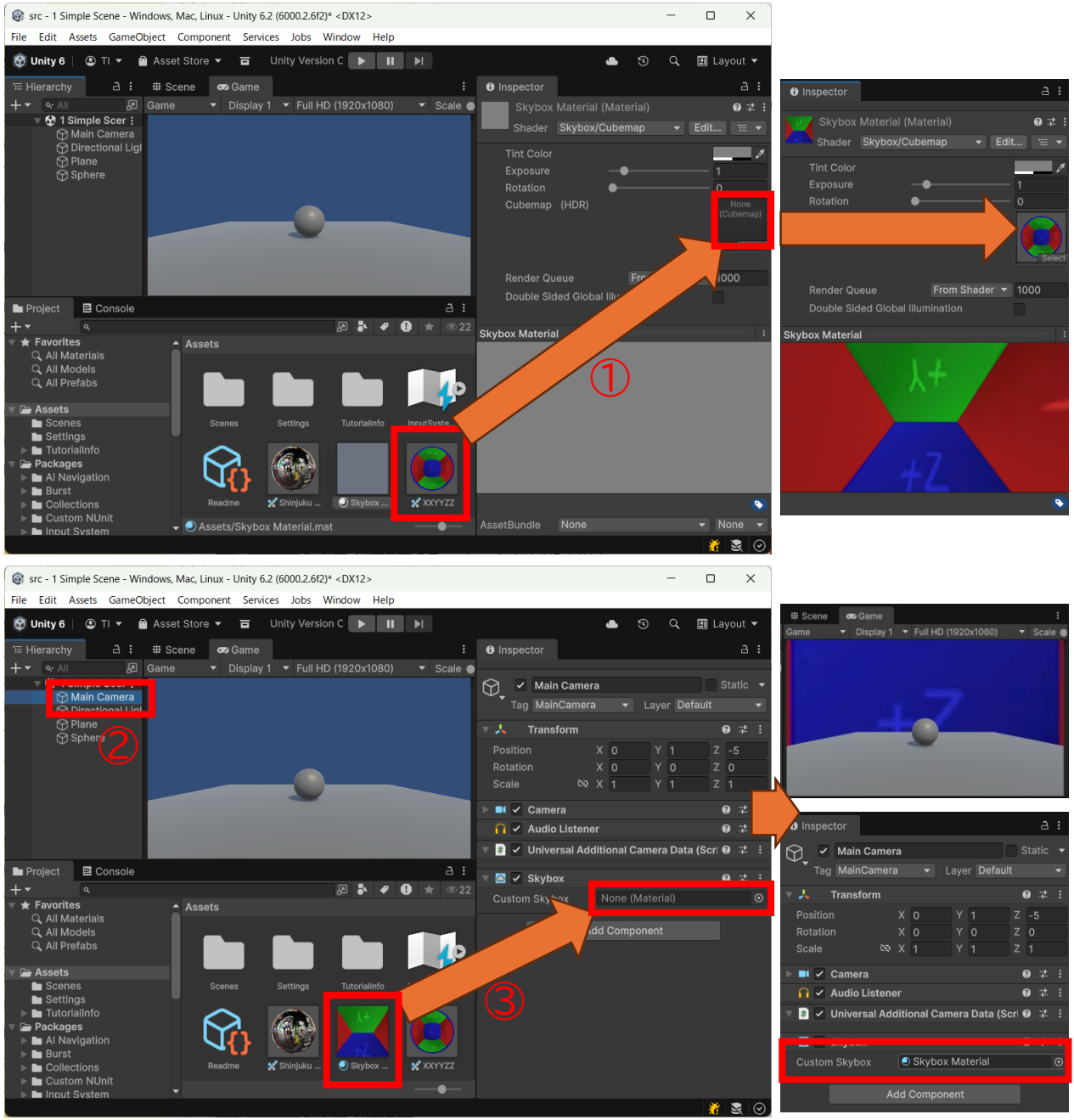




天球の設定 その3

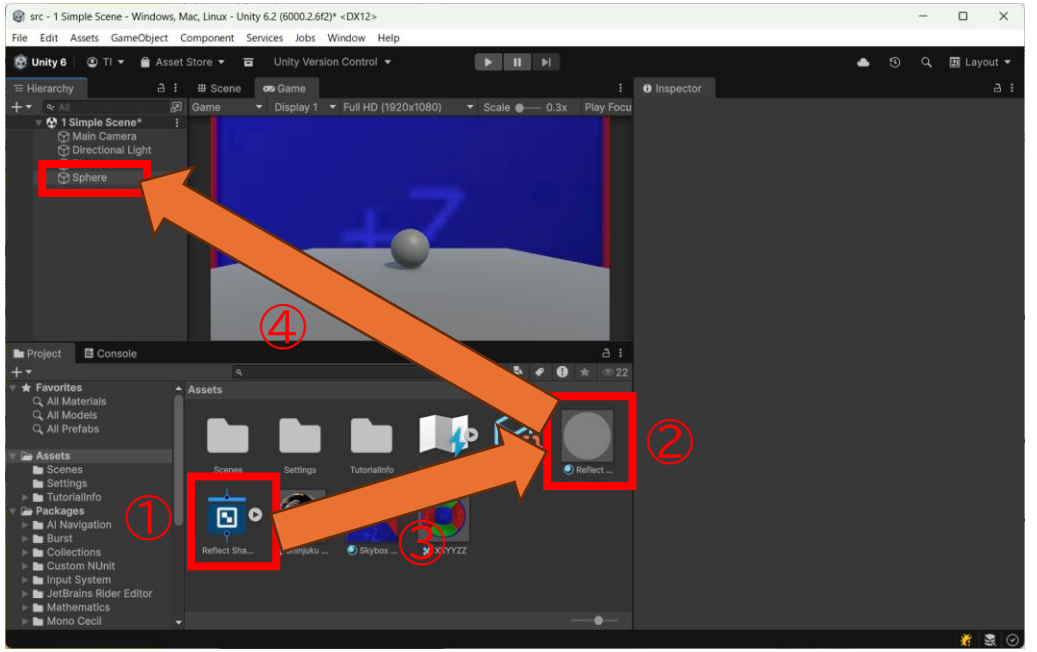
- スカイボックスコンポーネントにマテリアルを設定
  1. テクスチャにキューブマップを指定

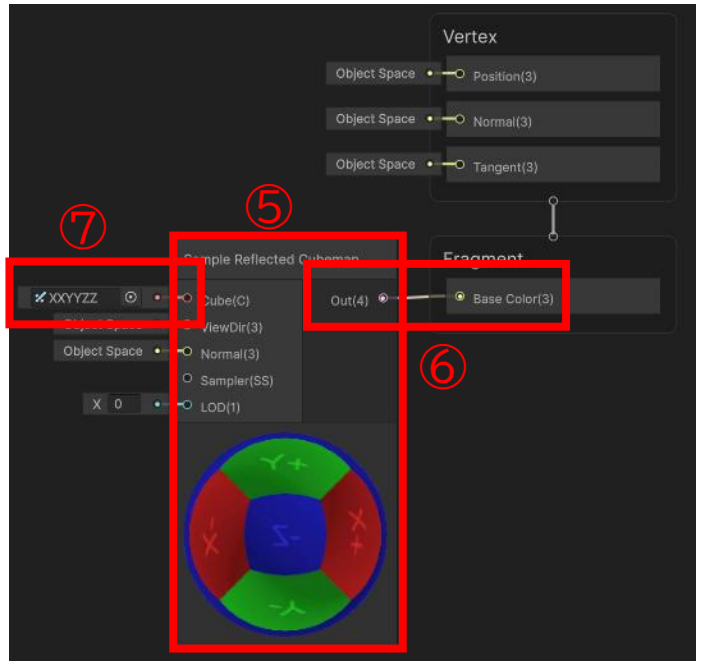
- カメラにSkyboxを設定
  2. Mainカメラの選択
  3. Skybox MaterialをSkyboxコンポーネントに設定



オブジェクト(球)のSG

- Unlit Shader Graph を追加
  1. 名前「Reflect Shader Graph」
- マテリアルも追加
  2. 名前「Reflect Material」
  3. Reflect Shader Graph を登録
  4. Reflect Materialを球にバインド
5. Reflect Shader Graphに「Sample Reflected Cubemap」ノードを追加
  6. FragmentシェーダのBaseColorに繋げる
  7. キューブマップを「Cube」入力に指定する





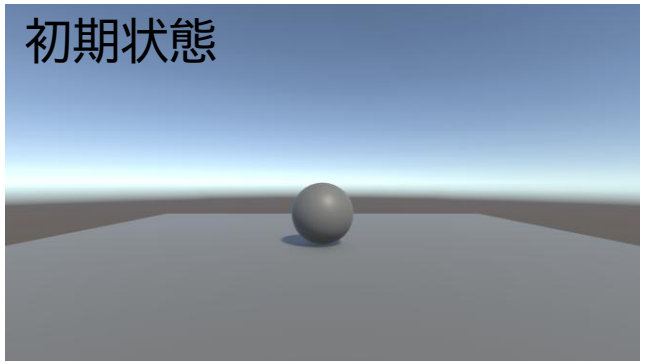
ゴール:1 Simple Sceneをこの状態にしよう！

レジュメ

- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

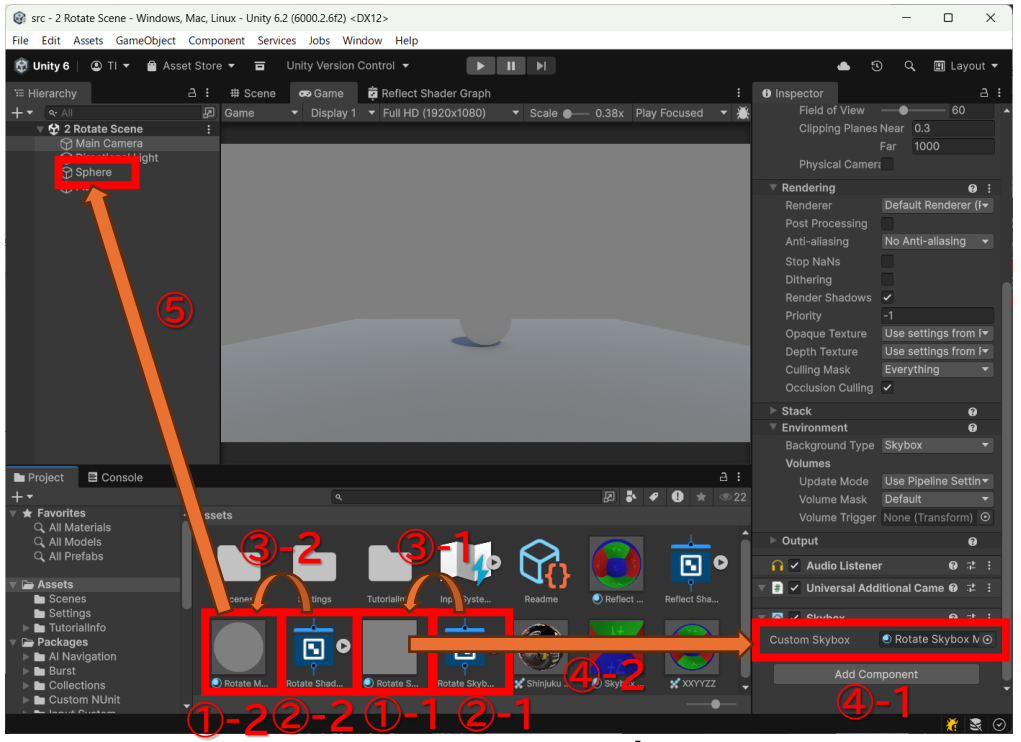
ゴール:2 Rotate Scene で回転しよう

2 Rotate Scene で回転しよう



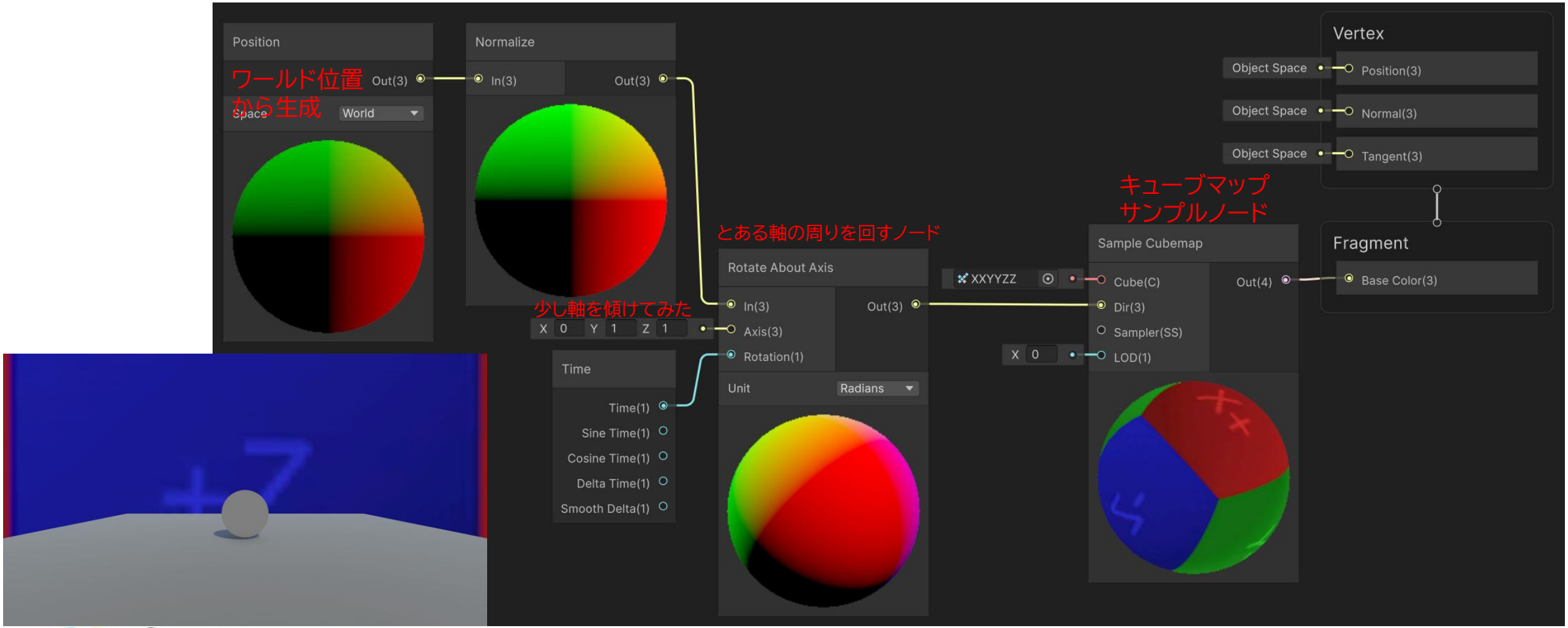
1. マテリアルを2つ作成
    1. Rotate Skybox Material
    2. Rotate Material
2. Unlit Shader Graphを2つ作成
    1. Rotate Skybox Shader Graph
    2. Rotate Shader Graph
3. マテリアルにShader Graphを設定
    1. Rotate Skybox MaterialにRotate Skybox Shader Graph
    2. Rotate MaterialにRotate Shader Graph
4. スカイボックスコンポーネントにRotate Skybox Materialを設定する
5. 球のオブジェクトにマテリアル(Rotate Material)を設定する
6. Rotate Skybox Shader Graphを編集して空を回す
7. Rotate Shader Graphを編集して空の動きに合わせる

2 Rotate Scene で回転しよう



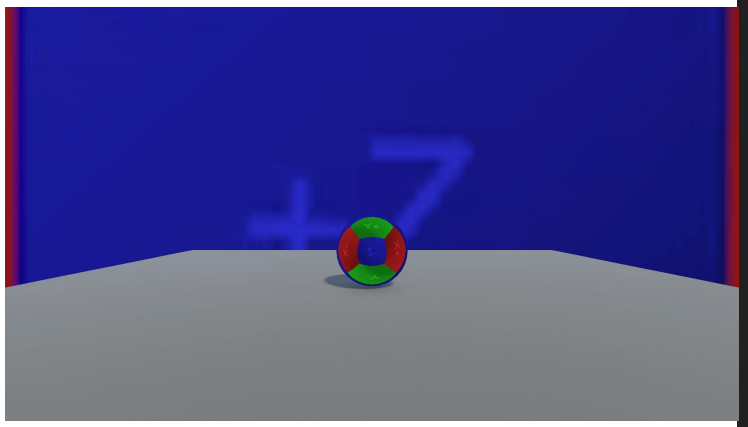
1. マテリアルを2つ作成
    1. Rotate Skybox Material
    2. Rotate Material
2. Unlit Shader Graphを2つ作成
    1. Rotate Skybox Shader Graph
    2. Rotate Shader Graph
3. マテリアルにShader Graphを設定
    1. Rotate Skybox MaterialにRotate Skybox Shader Graph
    2. Rotate MaterialにRotate Shader Graph
4. スカイボックスコンポーネントにRotate Skybox Materialを設定する
    1. カメラにSkyboxコンポーネントを追加する
    2. Rotate Skybox MaterialをCustom SkyboxにD&D
5. 球のオブジェクトにマテリアル(Rotate Material)を設定する
6. Rotate Skybox Shader Graphを編集して空を回す
7. Rotate Shader Graphを編集して空の動きに合わせる

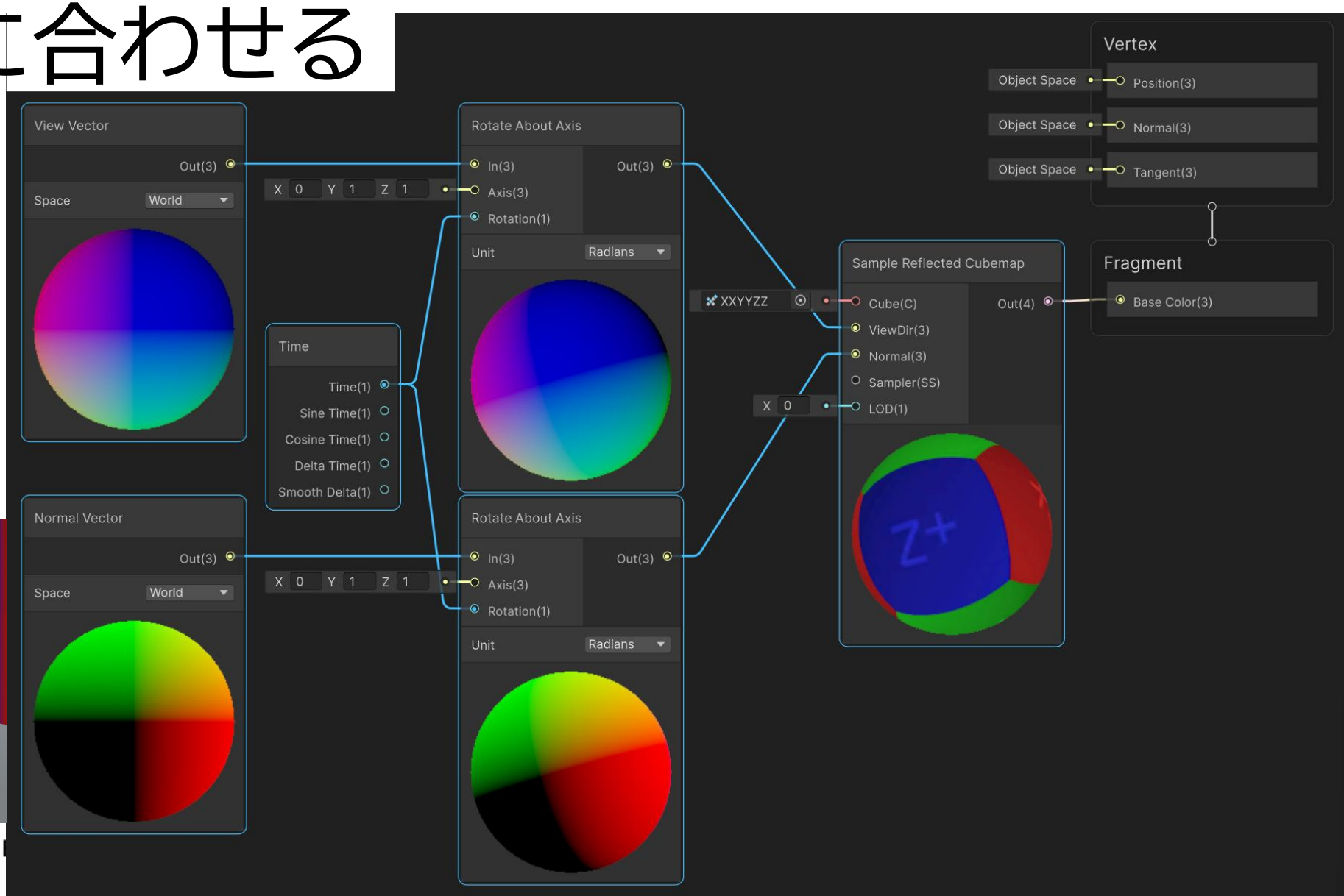
Rotate Skybox Shader Graphを編集して空を回す



Rotate Shader Graphを編集して空の動きに合わせる

- 同じ回転させた視線ベクトルと法線ベクトルで回す





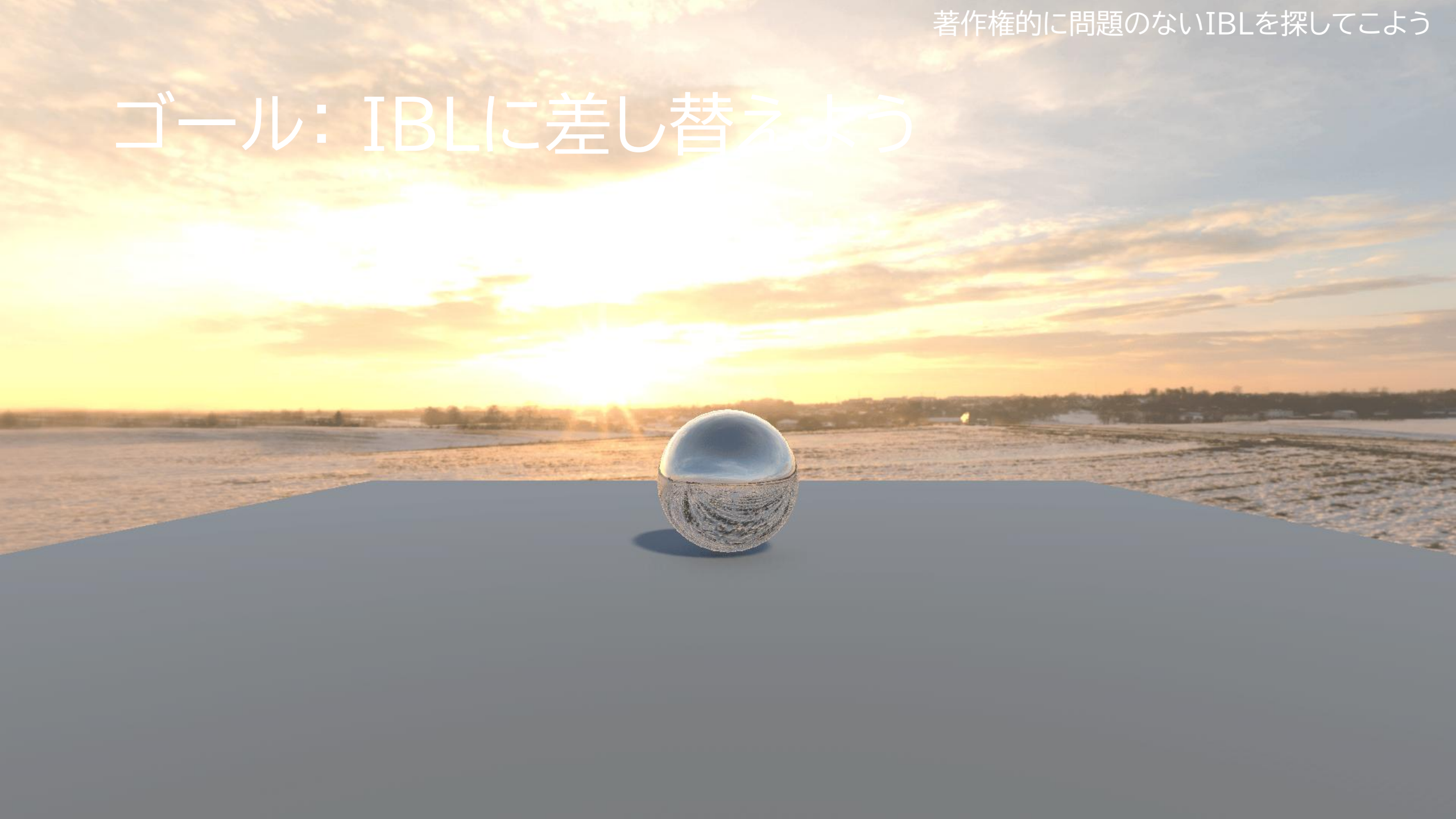
ゴール:2 Rotate Scene で回転しよう

レジュメ

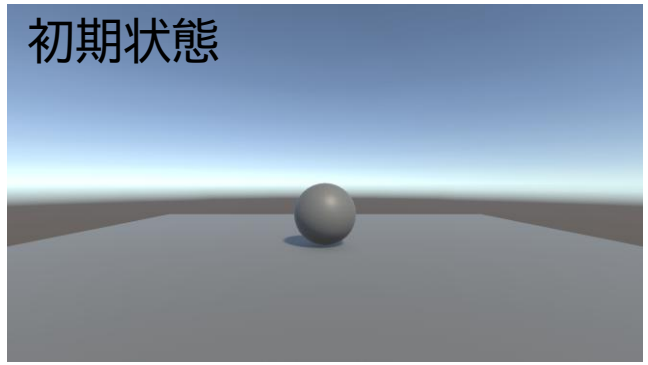
- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
    - 著作権に注意
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

著作権的に問題のないIBLを探してこよう

ゴール：IBLに差し替えよう



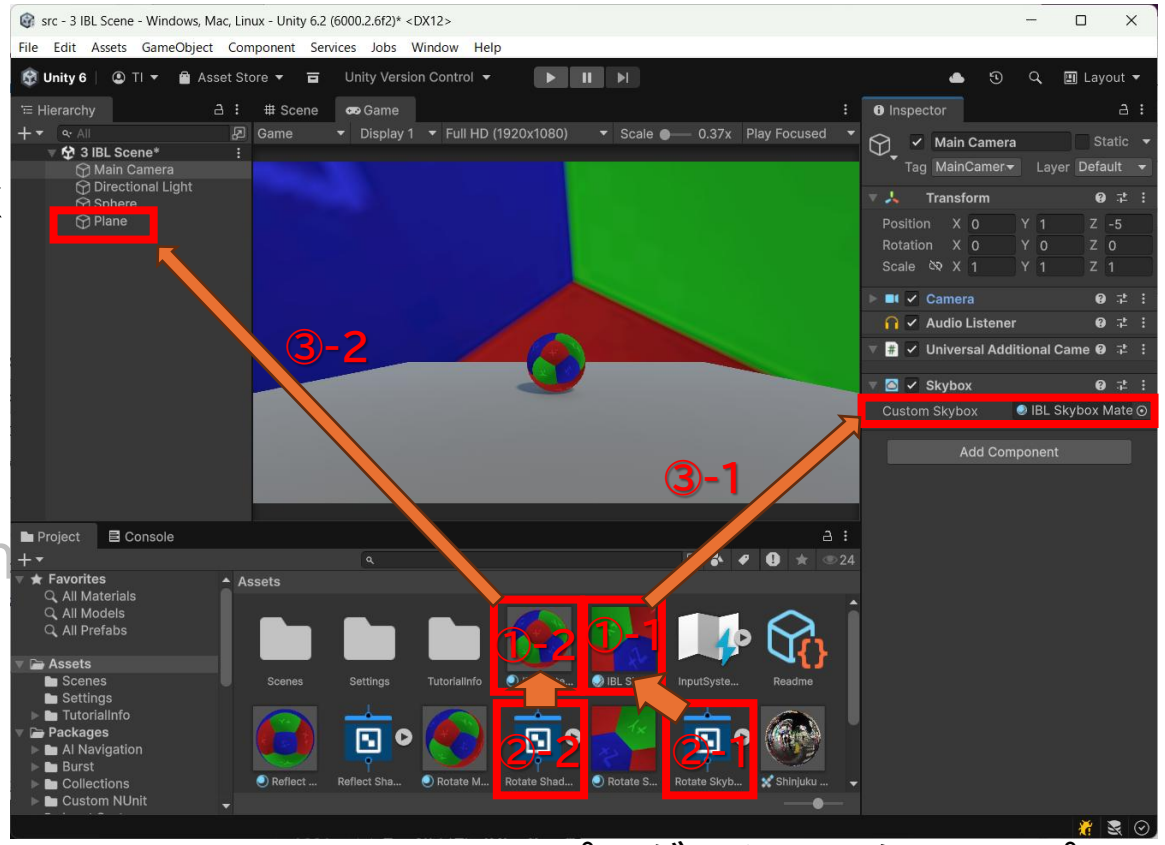
“3 IBL Scene”でIBLに差し替えよう



1. マテリアルを作成
    1. IBL Skybox Material
    2. IBL Material
2. マテリアルに(すでに作成した)Shader Graphを設定
    1. IBL Skybox MaterialにRotate Skybox Shader Graph
    2. IBL MaterialにRotate Shader Graph
3. スカイボックスと球にマテリアルをアサイン
    - Skybox: IBL Skybox Material
    - 球: IBL Material
4. Shader Graph の PropertyにCubemapを追加
    1. Rotate Skybox Shader Graph
    2. Rotate Shader Graph
5. Cubemapテクスチャを設定

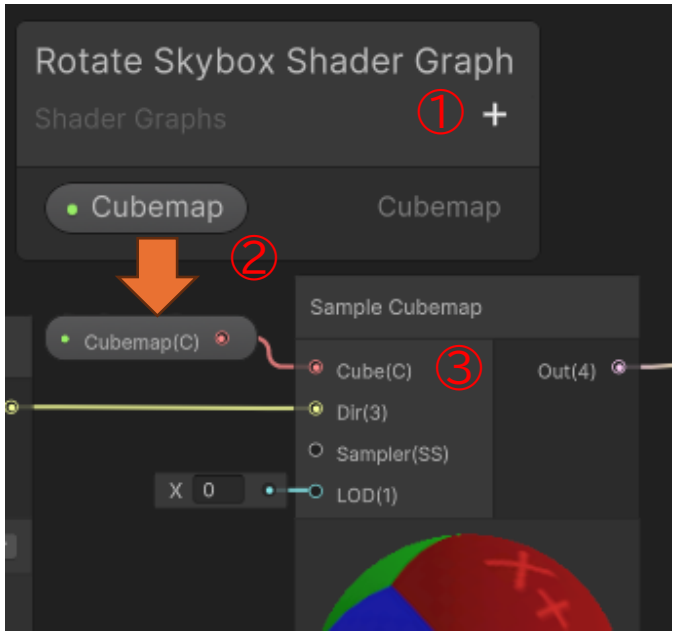
“3 IBL Scene”でIBLに差し替えよう

1. マテリアルを作成
    1. IBL Skybox Material
    2. IBL Material
2. マテリアルに(すでに作成した)Shader Graphを設定
    1. IBL Skybox MaterialにRotate Skybox Shader Graph
    2. IBL MaterialにRotate Shader Graph
3. スカイボックスと球にマテリアルをアサイン
    - Skybox: IBL Skybox Material
    - 球: IBL Material
4. Shader Graph の PropertyにCubem
    1. Rotate Skybox Shader Graph
    2. Rotate Shader Graph
5. Cubemapテクスチャを設定



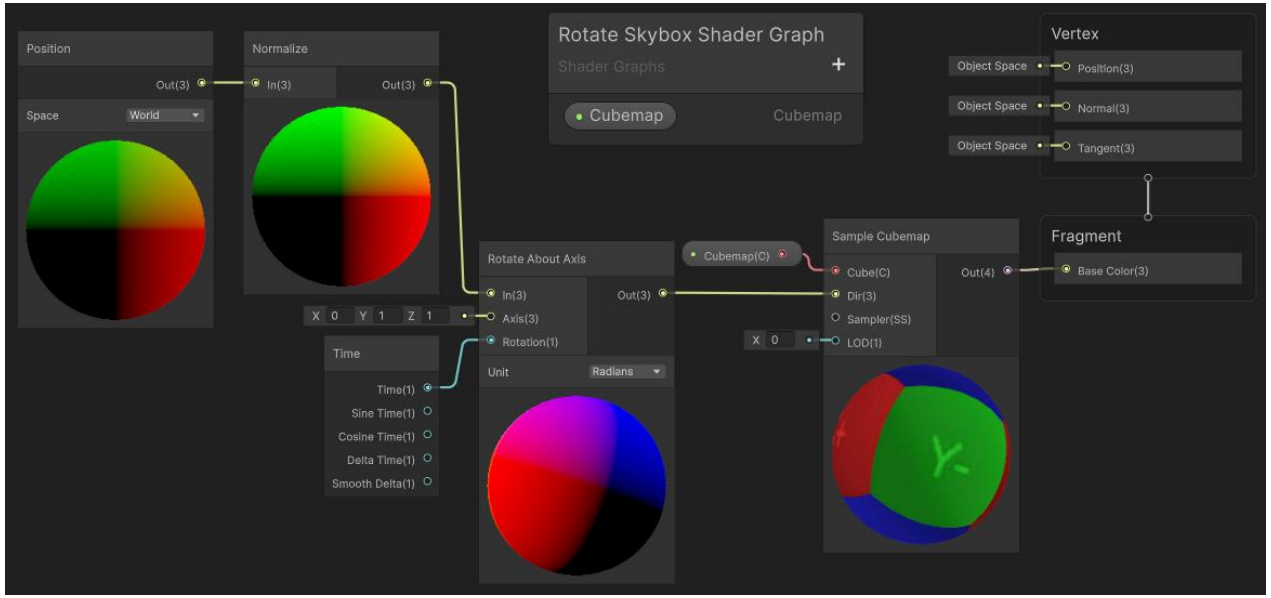
4. Shader Graph の Propertyに Cubemapを追加

1. Blackboardに「Cubemap」を追加
    1. わかりやすい名前をつける（「Cubemap」よりも「Skymap」が良い）
2. D&Dして、Blackboardからグラフウィンドウに変数を追加
3. キューブマップをサンプルする関数の「Cube」入力に挿入

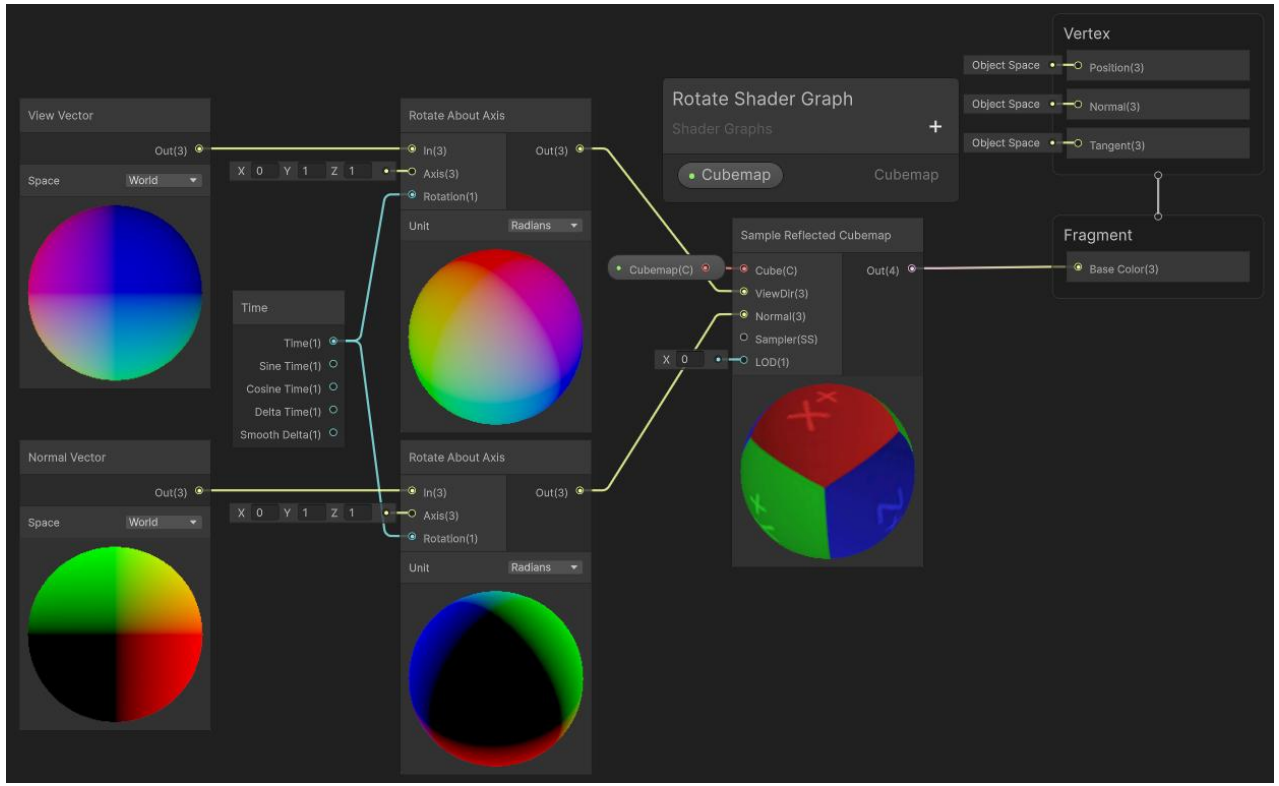


変更後のシェーダ

- 両方のシェーダに設定
  1. IBL Skybox Material
  2. IBL Material



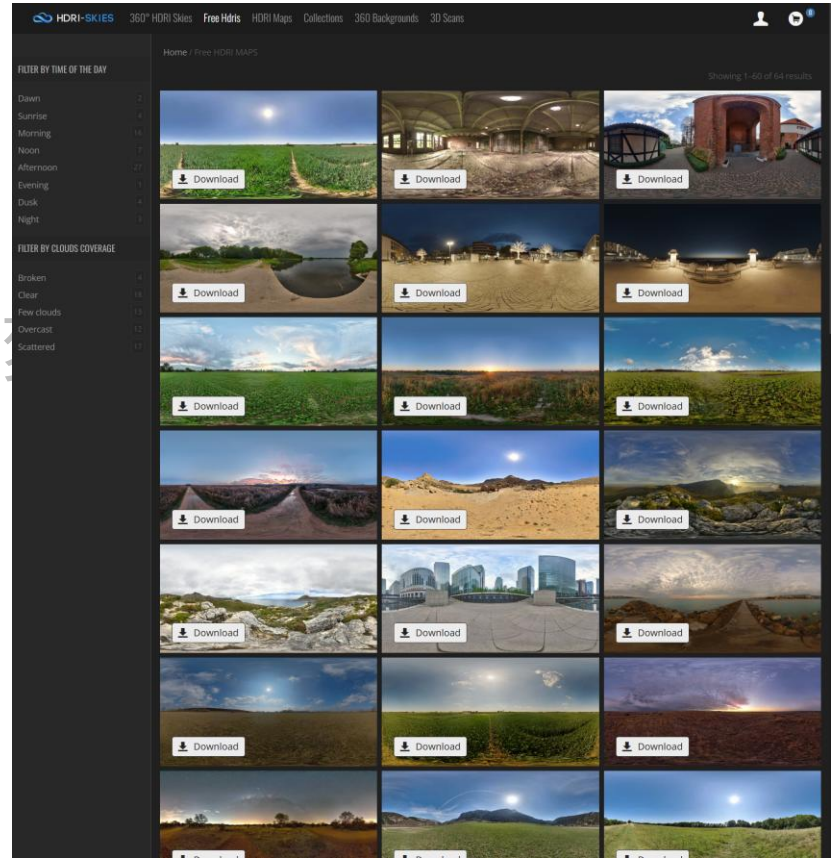
Rotate Skybox Shader Graph



Rotate Shader Graph

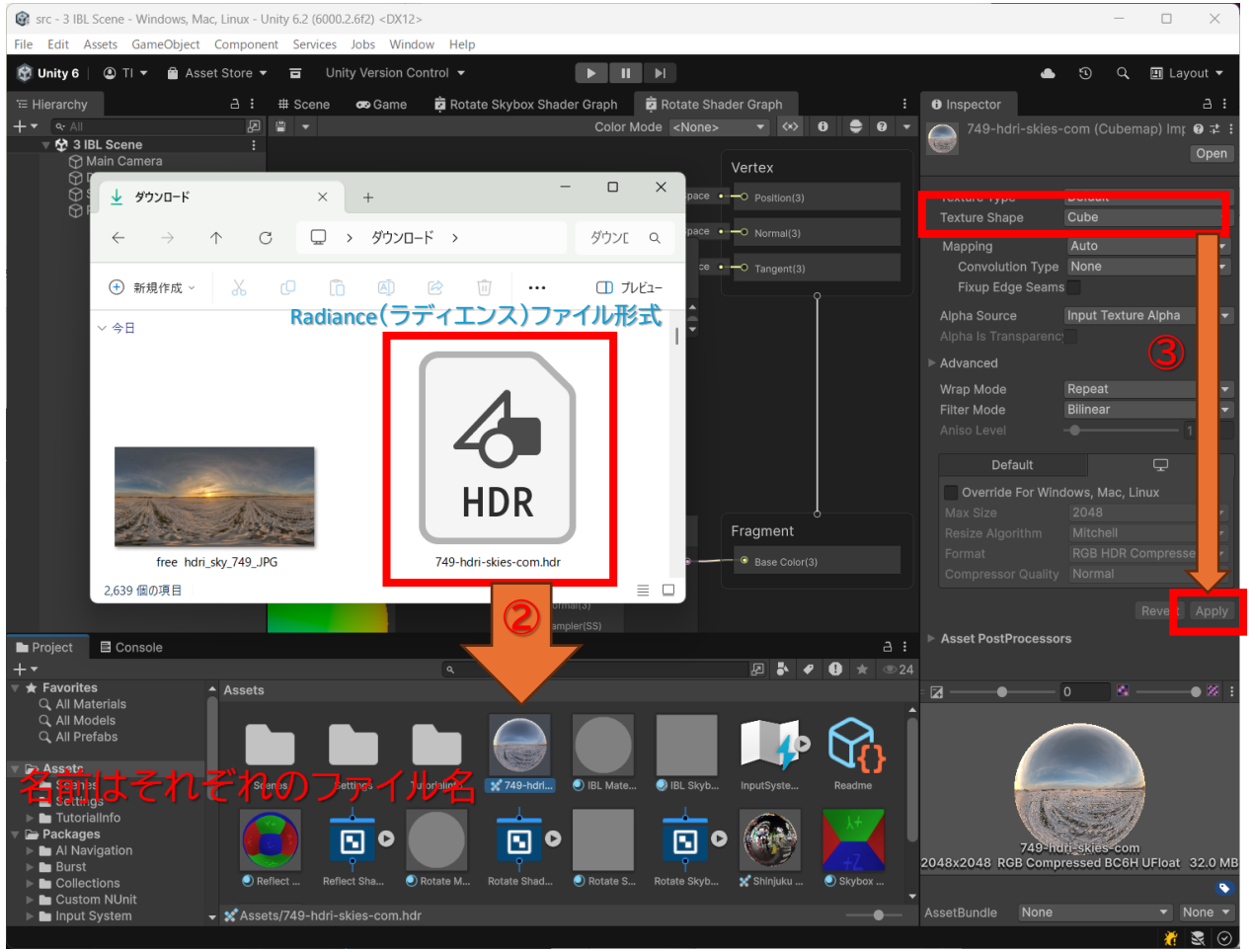
5. Cubemapテクスチャを設定

1. Poly Havenなどで環境マップ等をダウンロード
   - 最初に入手したマテリアルでも良い
   - ここでは、HDRI-SKIESからcc0の画像を取得
2. ファイルをProjectウィンドウにD＆D
3. Texture ShapeをCubeにしてApplyで
4. マテリアルのテクスチャに設定



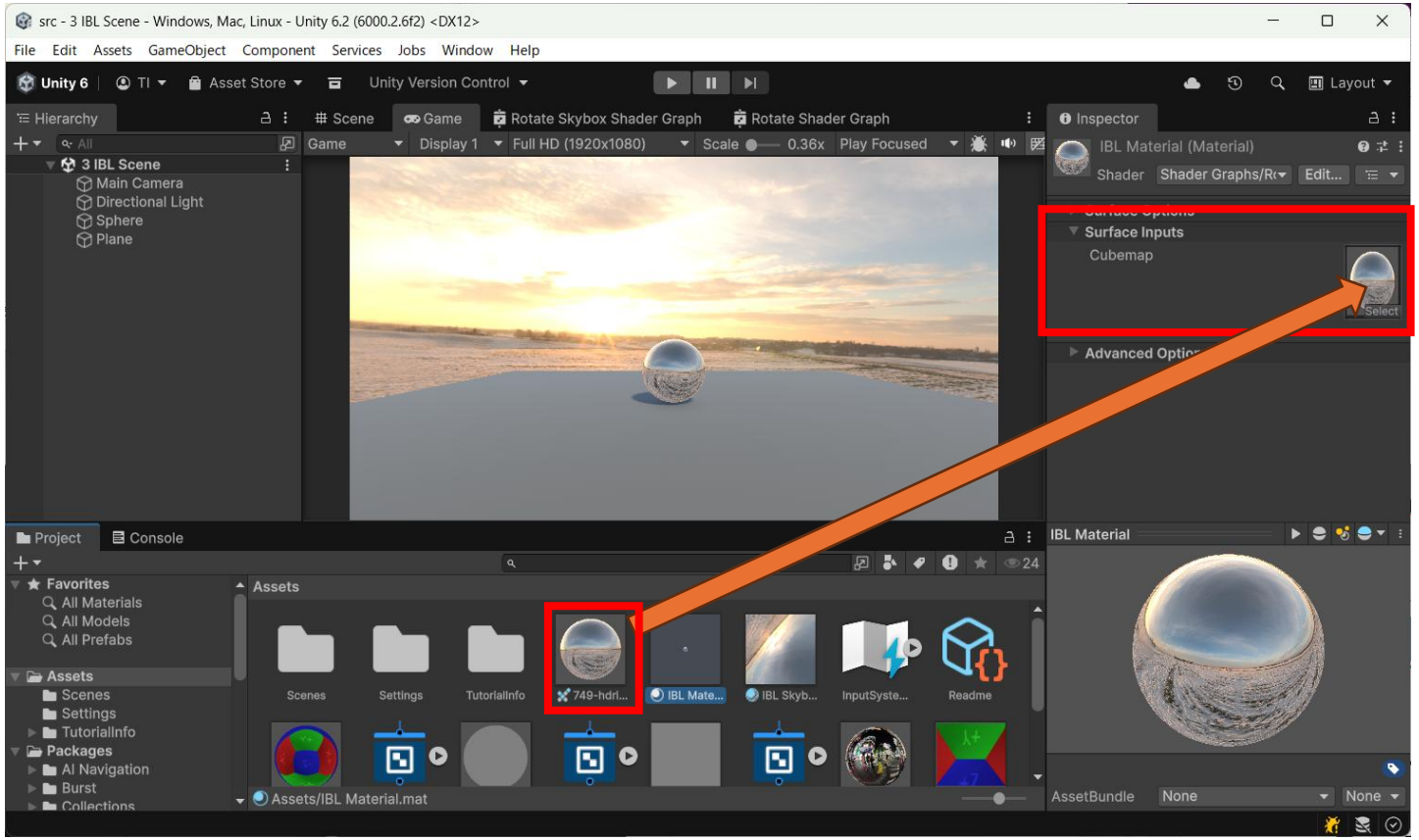
5-2,3. Cubemapテクスチャを設定

2. ファイルを解凍してProjectウィンドウにD&D
3. Texture ShapeをCubeにしてApplyで変換
4. マテリアルのテクスチャに設定



5-4. マテリアルのテクスチャに設定

- 2つのマテリアルで設定
  1. IBL Skybox Material
  2. IBL Material



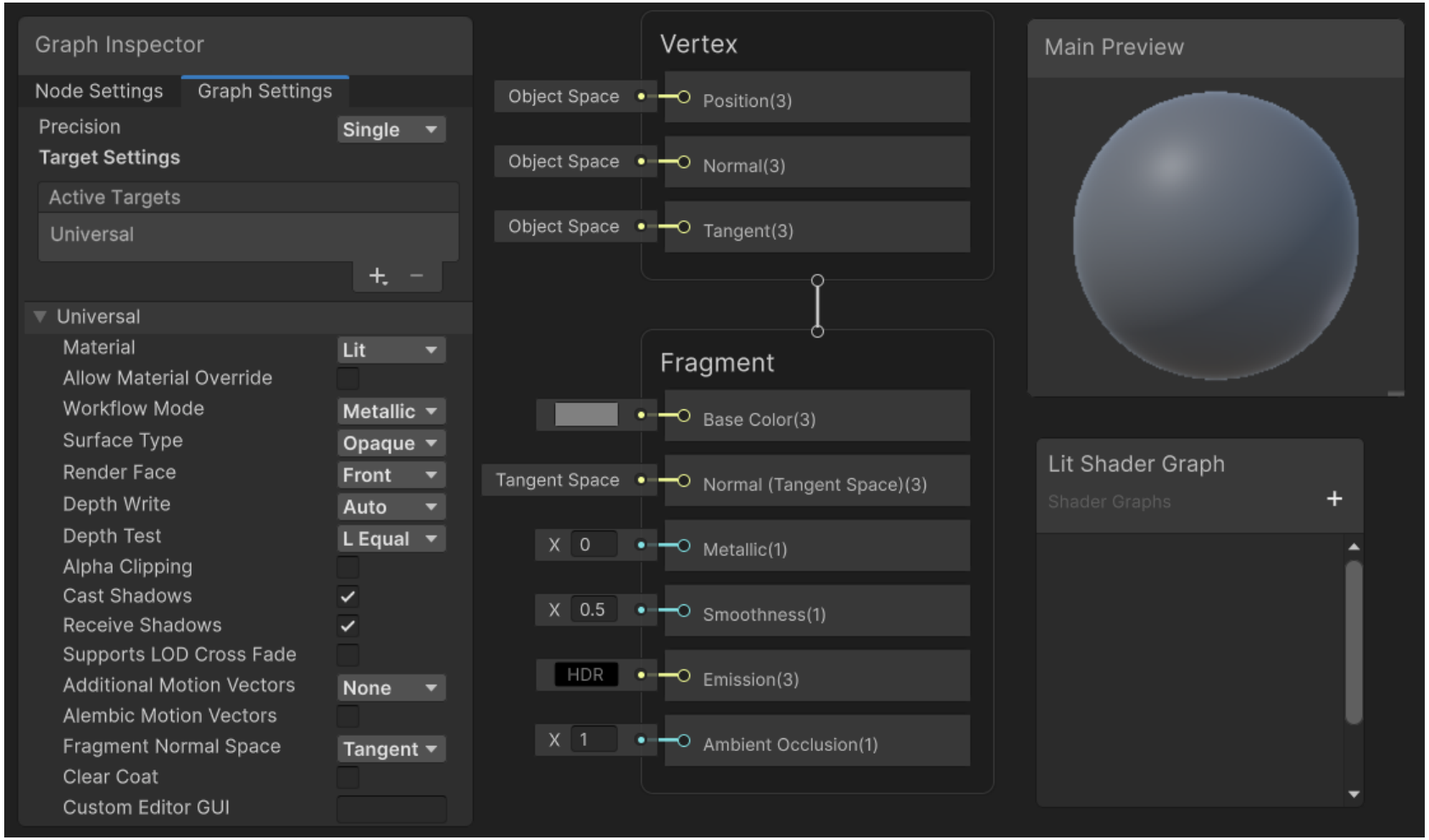

著作権的に問題のないIBLを探してこよう
ゴール：IBLに差し替えよう
やってみよう
このような環境マップだと目が回るので、星空の画像貼り付けてみましょう

レジュメ

- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
    - キューブマップの効果をシーンに適応する
  - 反射プローブ
  - 屈折

ゴール:4 Lit SceneのPBRで表示

やること:Lit Shader Graphでキューブマップを取り込んだIBLテクスチャの反映

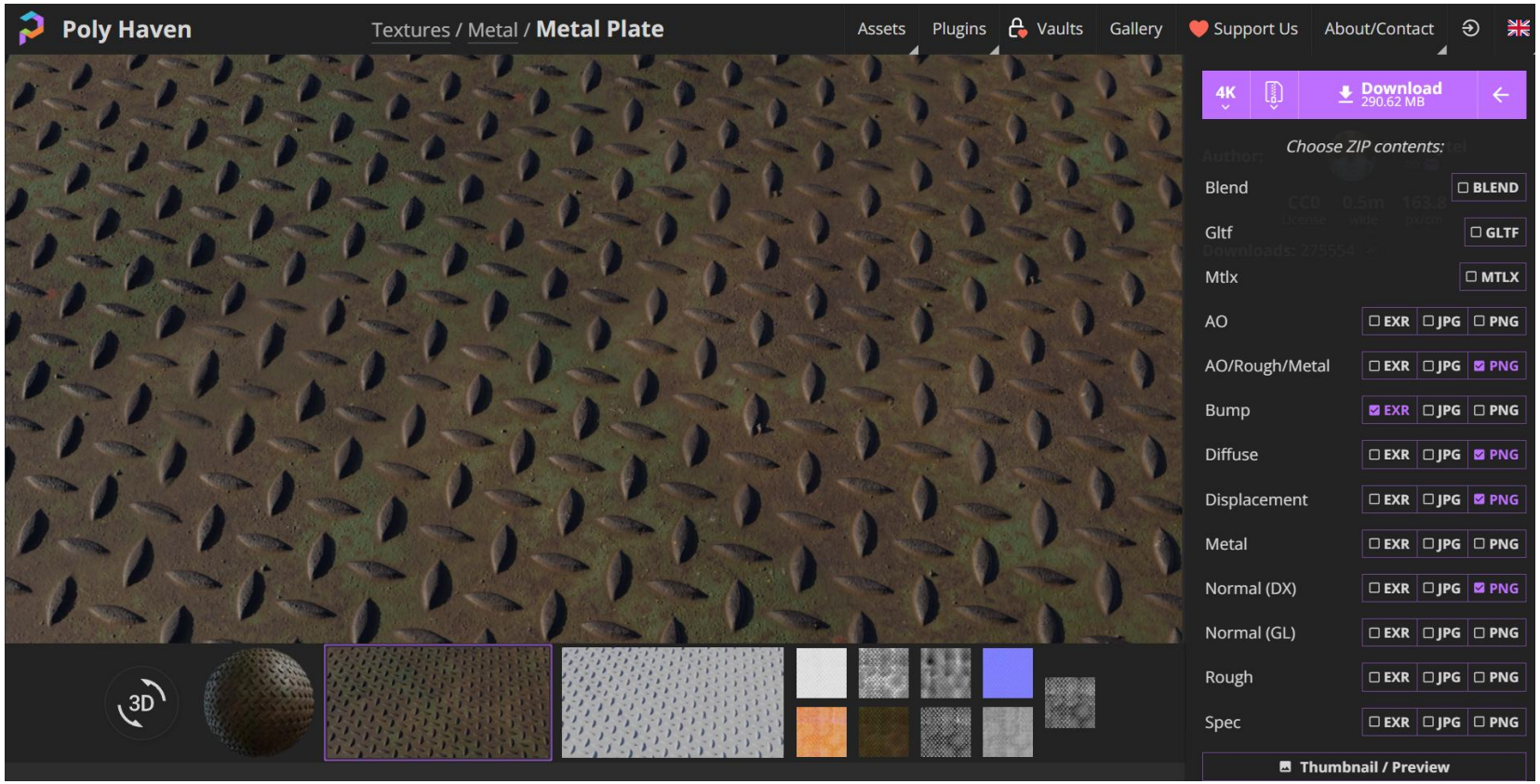


Lit Sceneの球をPBRで表示

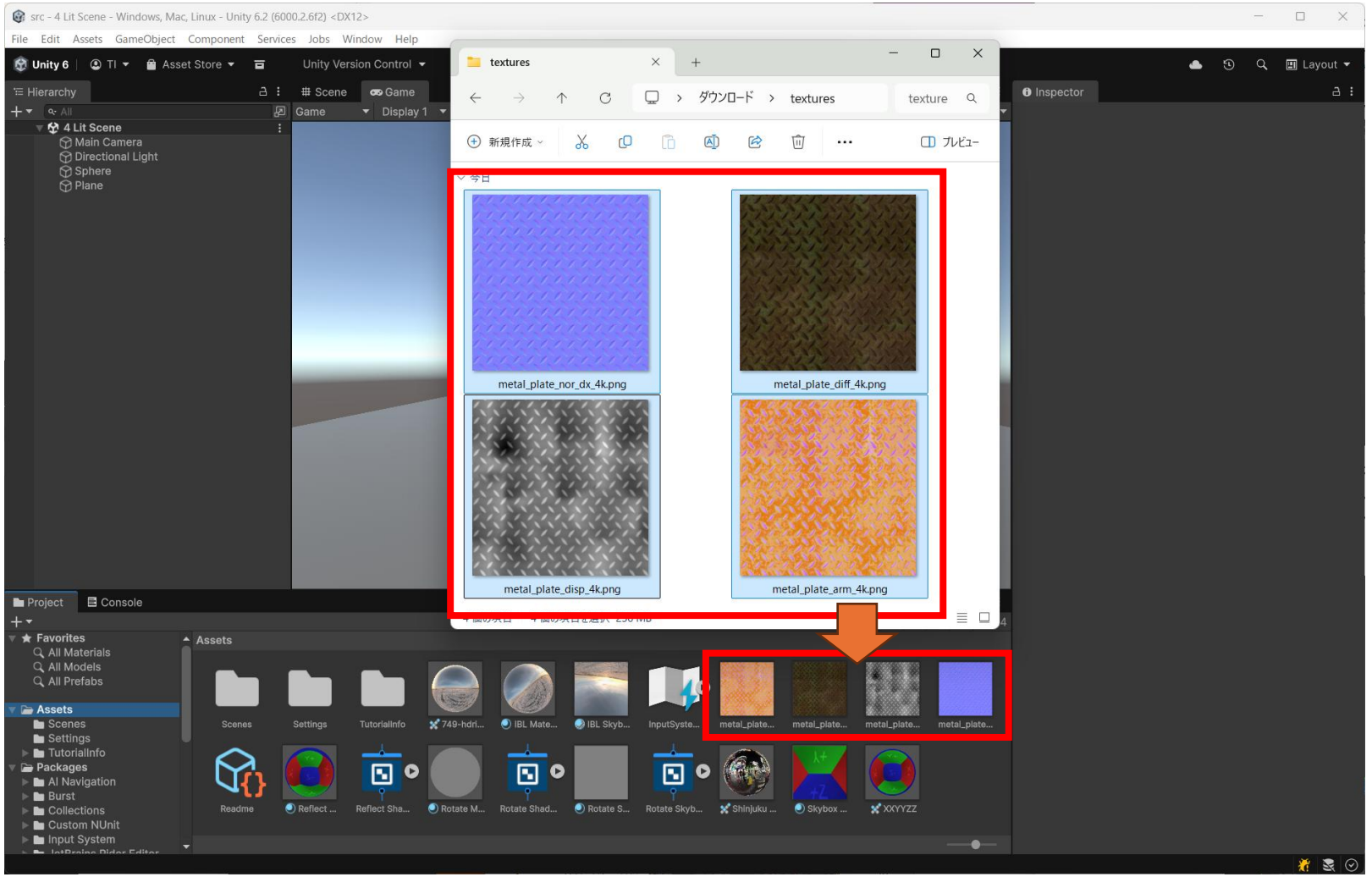
1. IBLテクスチャをダウンロード
2. IBLテクスチャを反映
    1. ファイルをプロジェクトに追加
    2. マテリアルを追加
    3. Lit Shader Graphを追加
    4. マテリアルにShader Graphを設定
    5. オブジェクトにマテリアルを設定
    6. Shader Graphでテクスチャを反映
3. 背景を設定

1.IBLテクスチャをダウンロード

1. Poly Heavenなどでテクスチャをダウンロード
   - 好きなテクスチャを

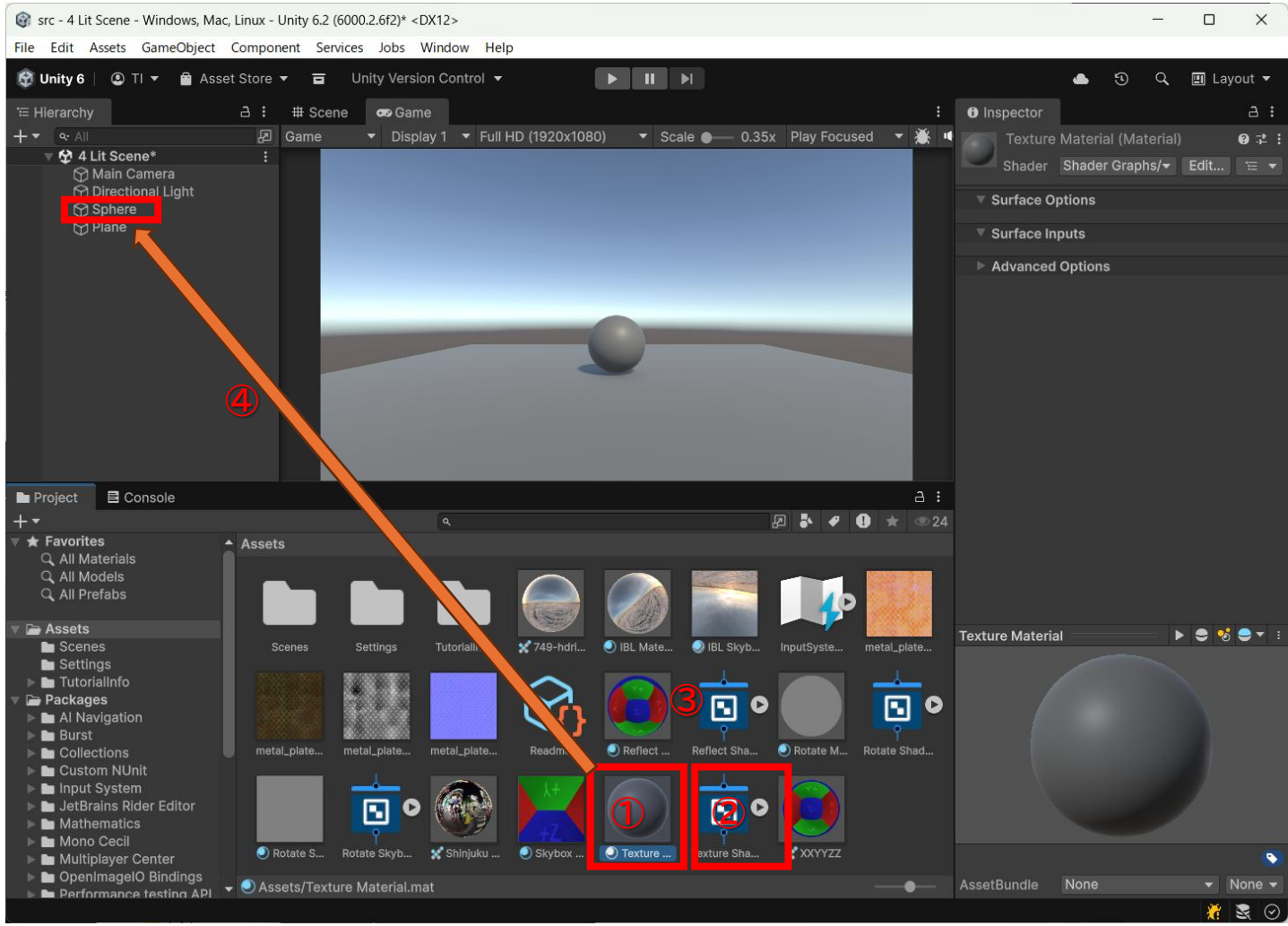


2.1ファイルをプロジェクトに追加



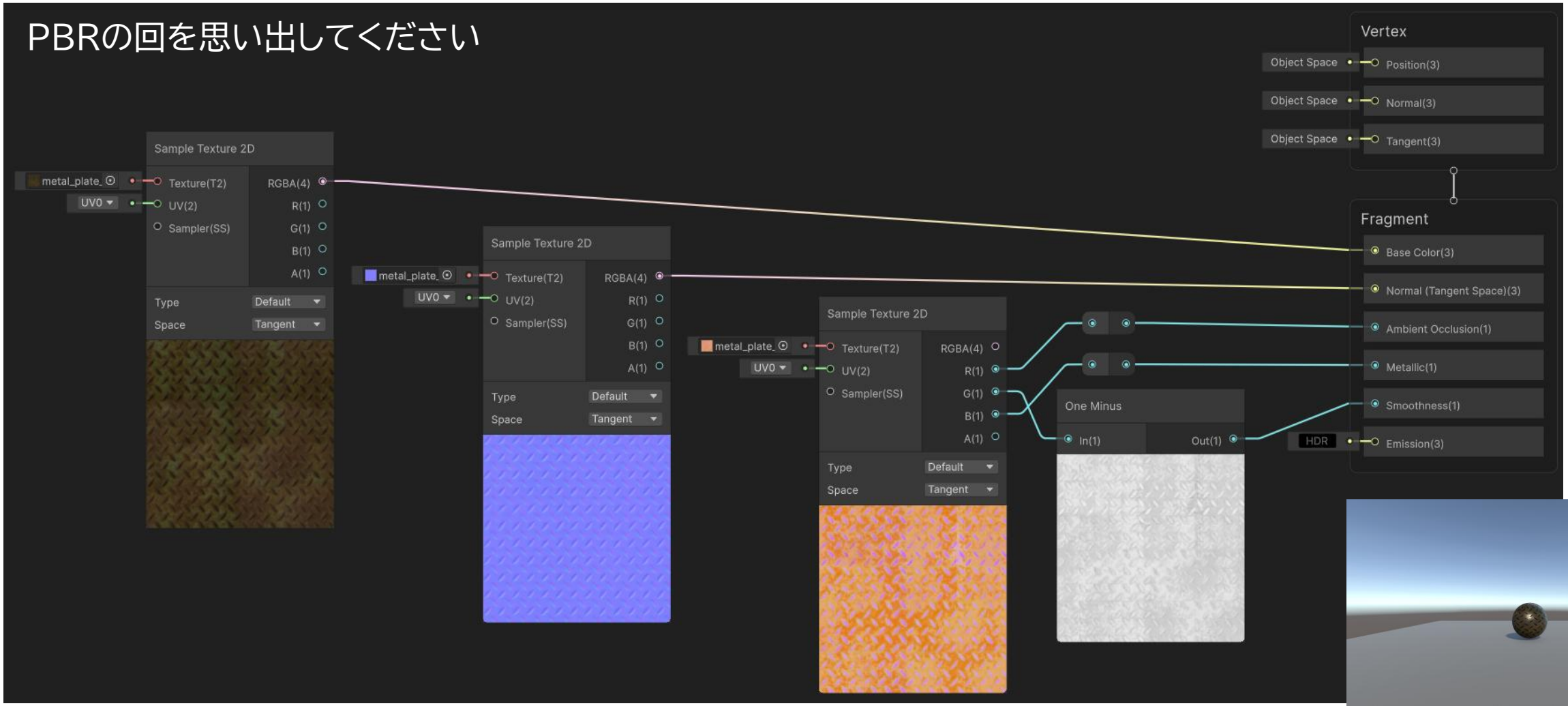
IBLテクスチャを反映

2. マテリアルを追加
   - 名前:「Texture material」等
3. Lit Shader Graphを追加
   - 名前:「Texture Shader Graph」等
4. マテリアルにShader Graphを設定
5. オブジェクトにマテリアルを設定



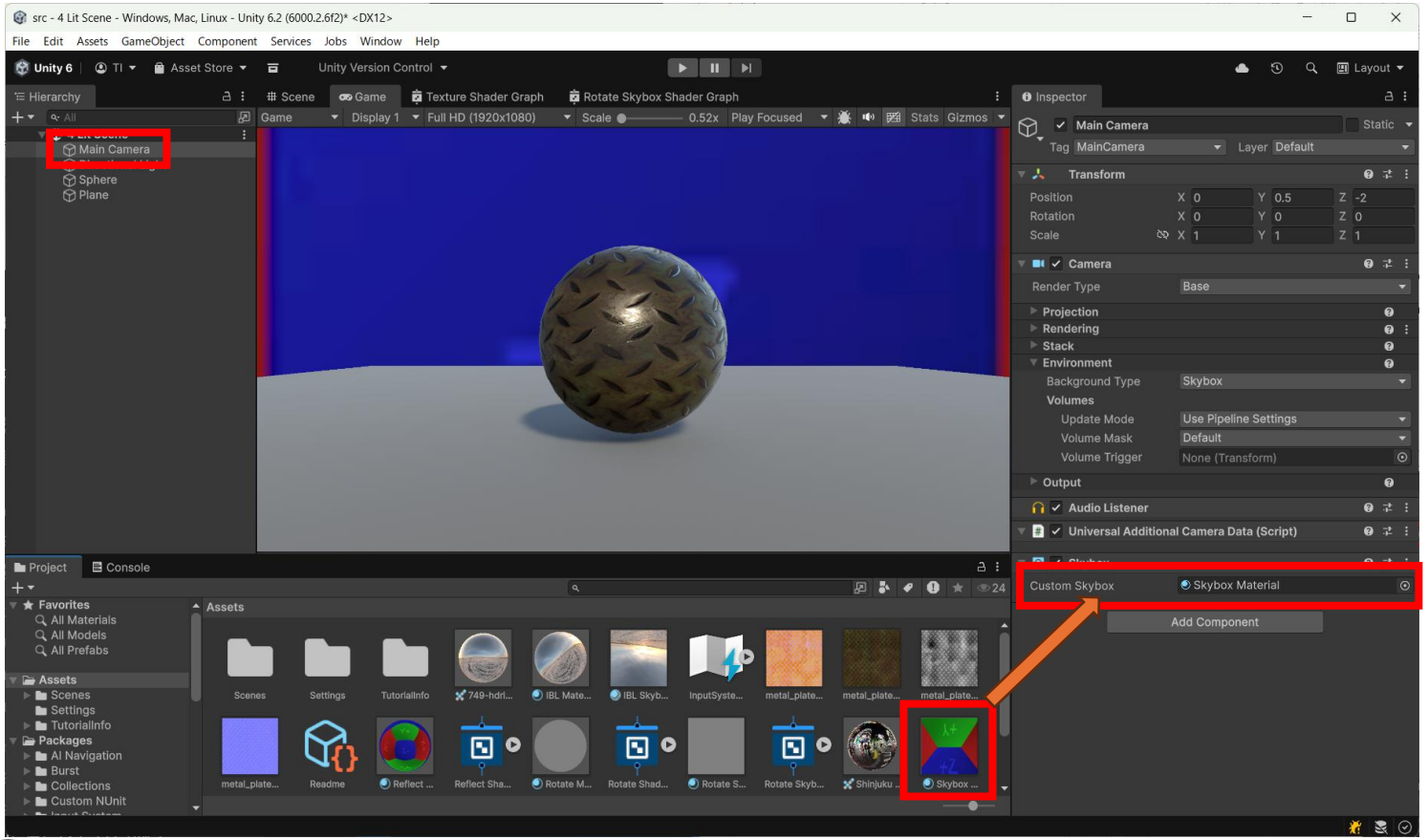
2.6 Shader Graphにテクスチャを反映

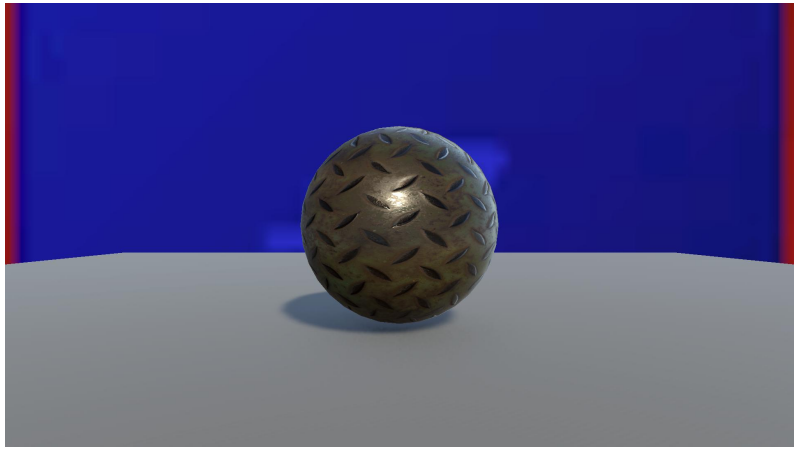
PBRの回を思い出してください



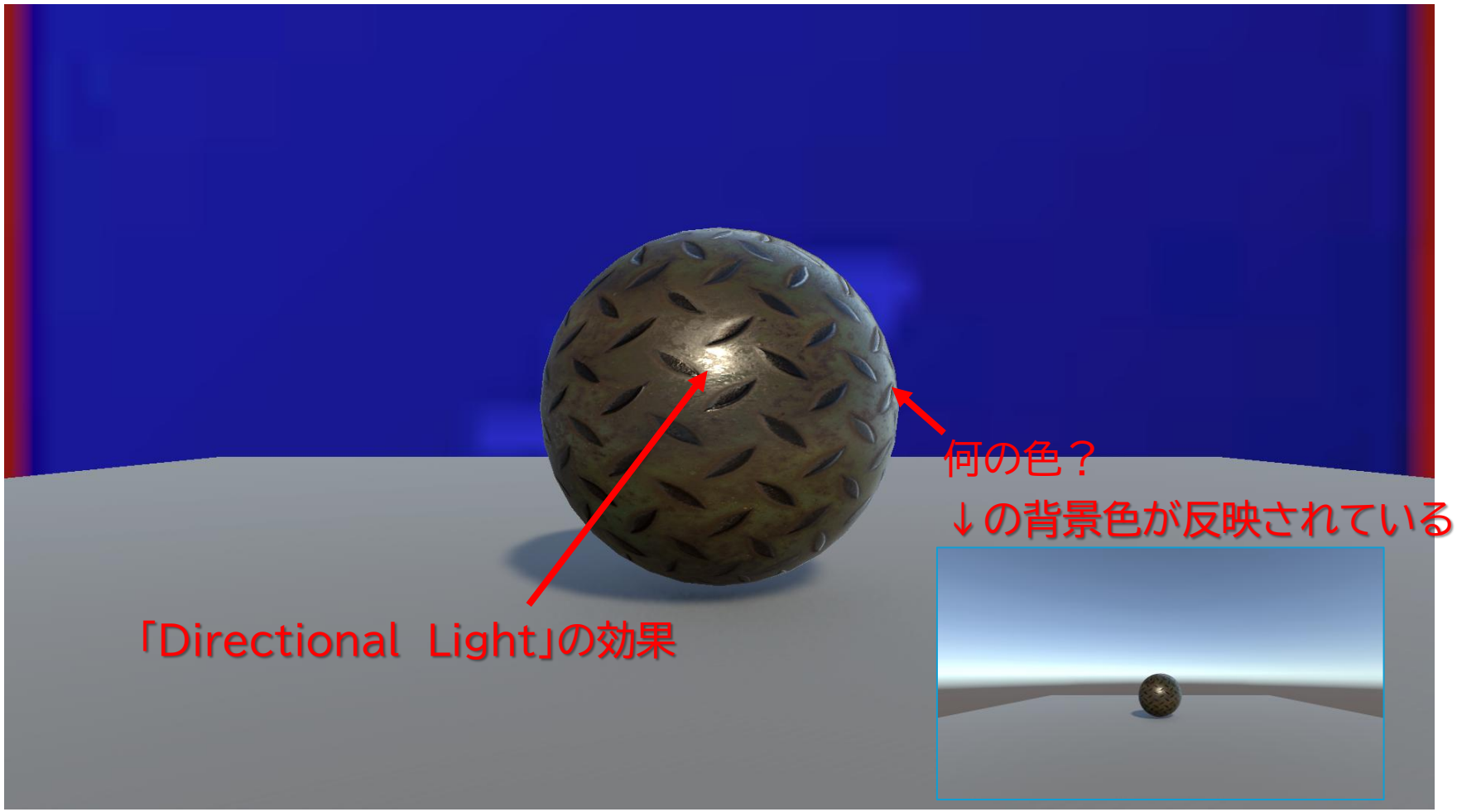
3.背景を設定

- Main CameraにSkymapを追加して反映



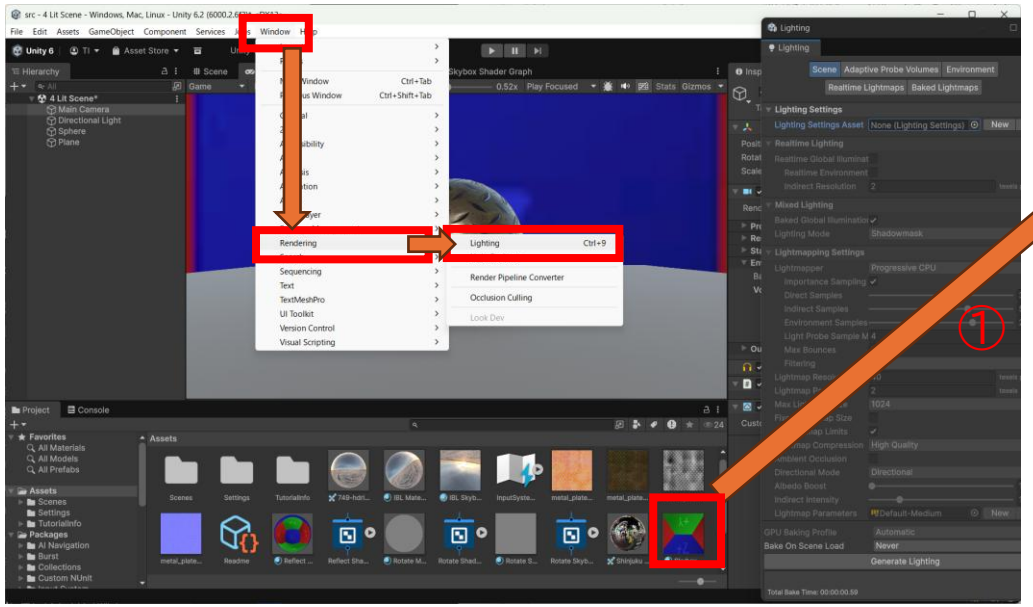


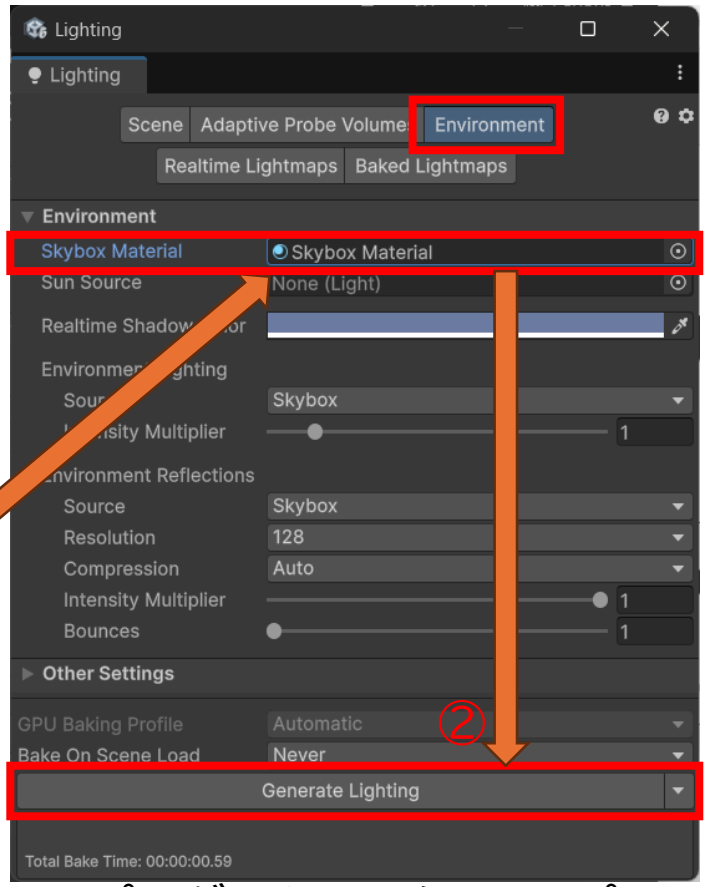
問題点：背景の色が反映されていない



背景の色の更新

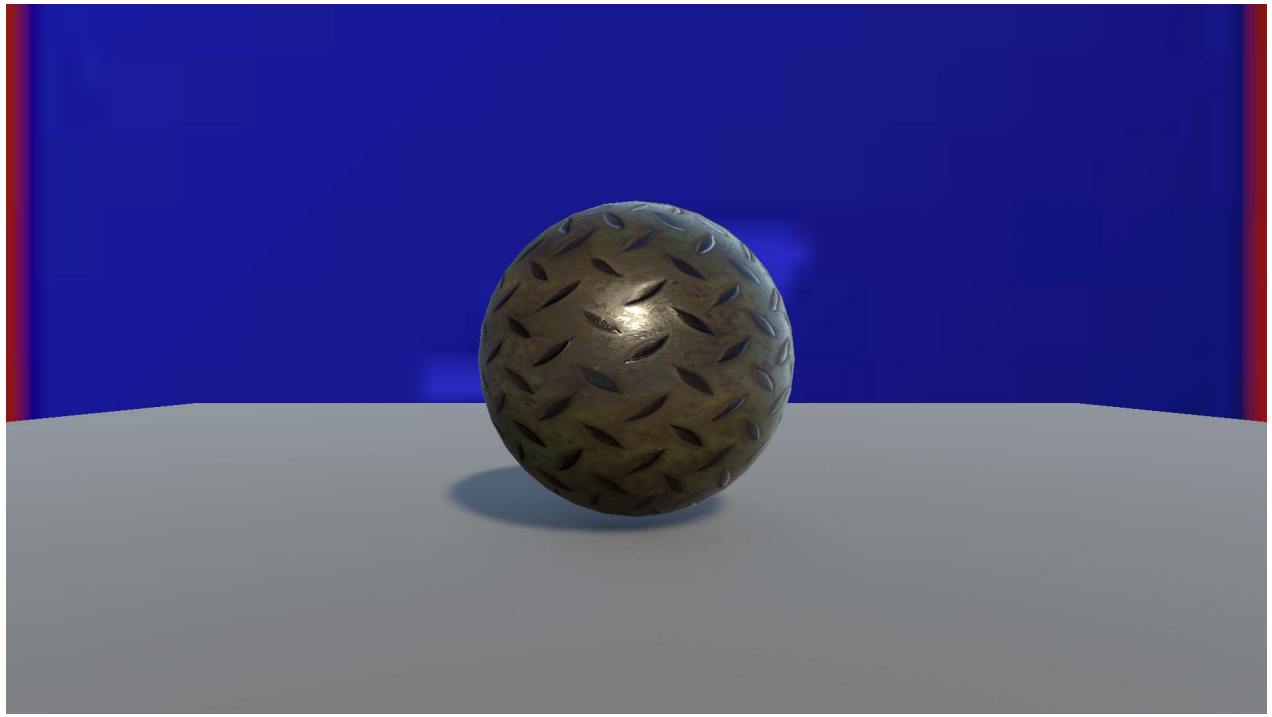
- 「Window」-「Rendering」-「Lighting」のダイアログで設定
- 「Environment」タブ
  1. 「Skybox Material」に天球のマテリアルを追加
  2. 「Generate Lighting」を押す





キューブマップの色が反映された

- キューブマップが動く場合は非対応（ベイクなので）
  - 興味が出てきたら自分で調べよう



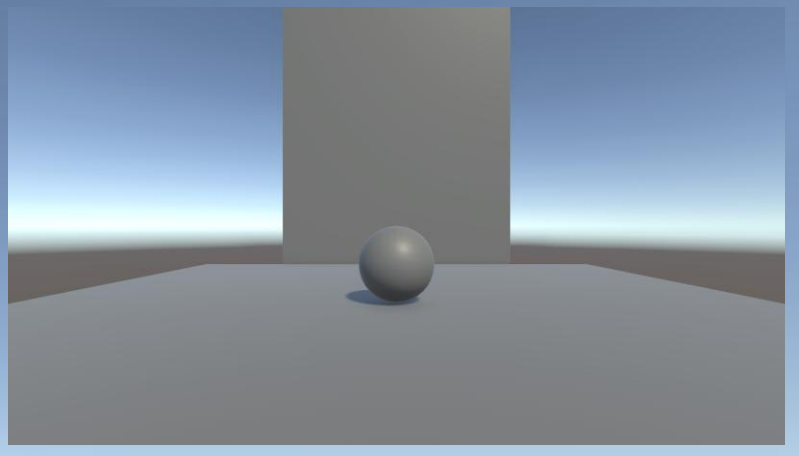
ゴール:4 Lit Sceneの球をPBRで表示

やってみよう

レジュメ

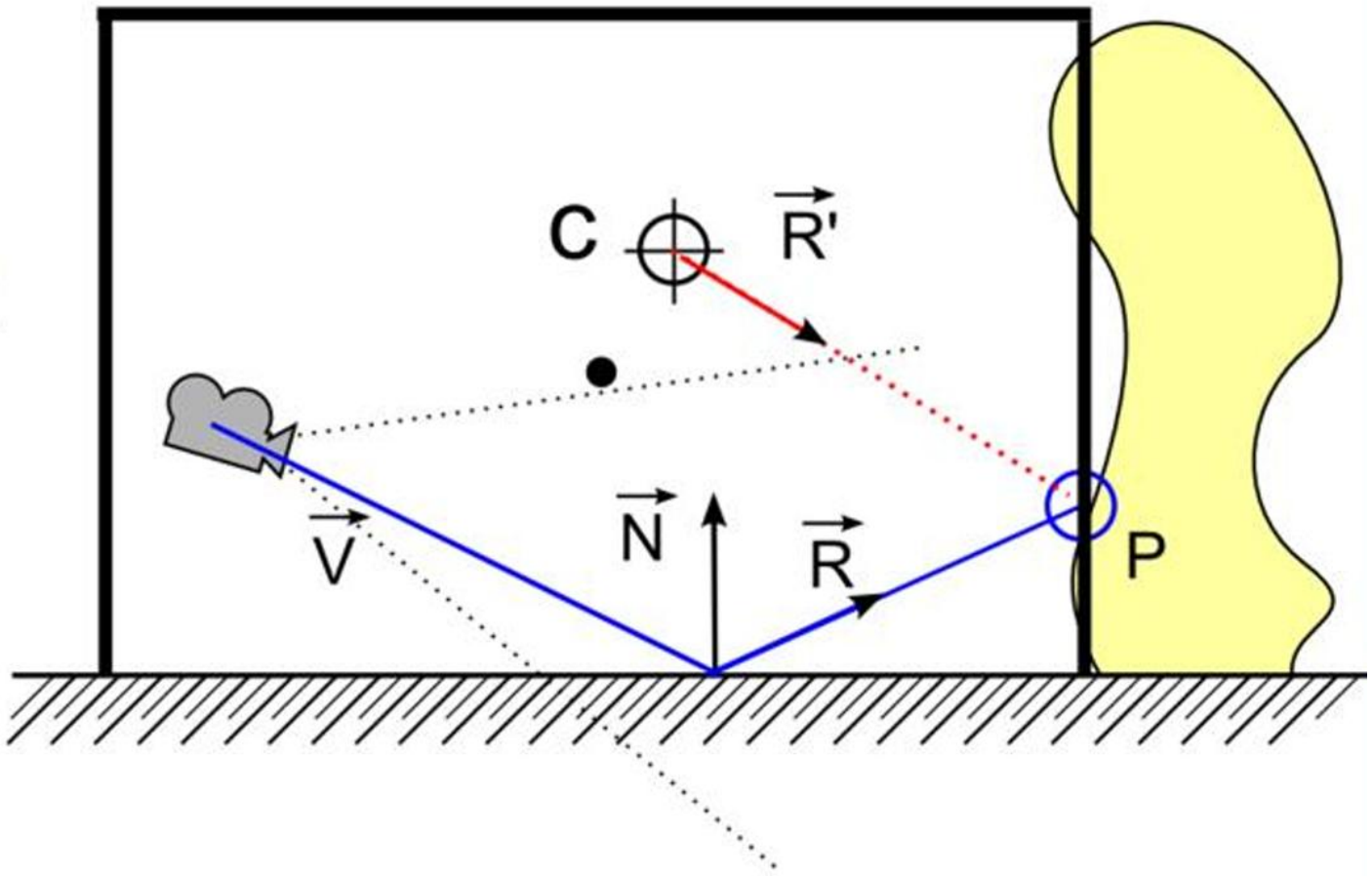
- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

ゴール:5 Reflection Probe Sceneをこの状態にしよう！



Parallax-corrected Cubemap
視差補正キューブマップ

- キューブマップは基本的に無限遠の景色を映している
  - 法線だけでみているので、カメラの位置が動いても見た目に影響しない
- キューブマップの箱のサイズを決めておくことで（直方体の形状で）厳密な反射を実現する

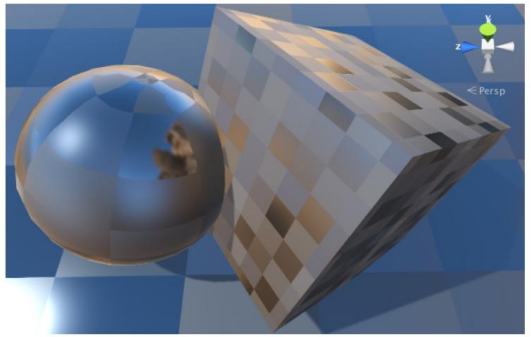


Unityの機能ではリフレクションプローブを使う

- ある点からの全方位の形式をキューブマップに記録

リフレクションプローブ

リフレクションプローブ (Reflection Probe) は、全方向に向かってその周囲の球状のビューをキャプチャするカメラのようなものです。キャプチャしたイメージはその後、反射マテリアルを持つオブジェクトに使用される キューブマップ として保存されます。リフレクションプローブの中には、指定したシーンで使用し、オブジェクトは最も近いプローブによって生成されたキューブマップを使うように設定されているものもあります。その結果、オブジェクトの反射を、その環境と調和した、説得力のある形で変化させる事ができます。



近接したオブジェクトを映し出す Reflection Probe

プロパティ

Unity は、プロジェクトに使用するレンダーパイプラインに応じて、リフレクションプローブ Inspector に異なるプロパティを表示します。

- プロジェクトで ユニバーサルレンダーパイプライン (URP) を使用する場合は、URP パッケージドキュメント を参照してください。
- プロジェクトで HD レンダーパイプライン (HDRP) を使用する場合は、HDRP パッケージドキュメント を参照してください。
- プロジェクトで ビルトインレンダーパイプライン を使用する場合、Unity は以下のプロパティを表示します。

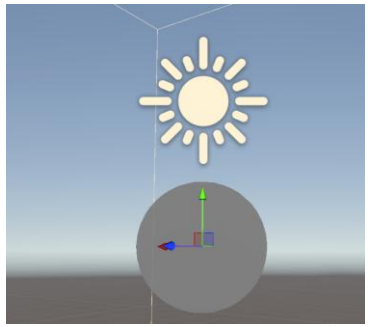
プロパティ	機能
Type	プローブの設定を **Baked**、**Custom**、**Realtime** のいずれにするかを選択します。**Baked** を選択すると、リフレクションプローブは、Reflection Probe Static フラグが無効であるゲームオブジェクトをランタイムに キャプチャしません。ベイクしたリフレクションプローブで動的ゲームオブジェクトをキャプチャたい場合は、**Custom** を選択して **Dynamic Objects** を有効にします。
Dynamic Objects	(**Custom** タイプのみ) **Static** とマークされていないオブジェクトを、強制的に映り込みにベイクする。
Cubemap	(**Custom** タイプのみ) カスタム キューブマップ を設定します。
Refresh Mode	(**Realtime** タイプのみ) 実行時にプローブをリフレッシュするか、どうリフレッシュするか、を選択します。 *On Awake* オプションは、プローブが有効になった最初の 1 回だけ描画します。 *Every Frame* は、毎フレーム更新時にプローブを描画します。オプションで **Time Slicing** (以下参照) を使う事ができます。**Via Scripting** オプションは、自動更新ではなくユーザースクリプトコマンドからプローブをリフレッシュします。
Time Slicing	(Realtime タイプのみ) プローブの更新を配布する時間を設定します。オプションとして All Faces At Once (9 フレーム以上で更新を配布する)、Individual Faces (14 フレーム以上で更新する)、そして No Time Slicing (1 フレームでしか更新が起らない)、があります。詳細に関しては以下を参照してください。

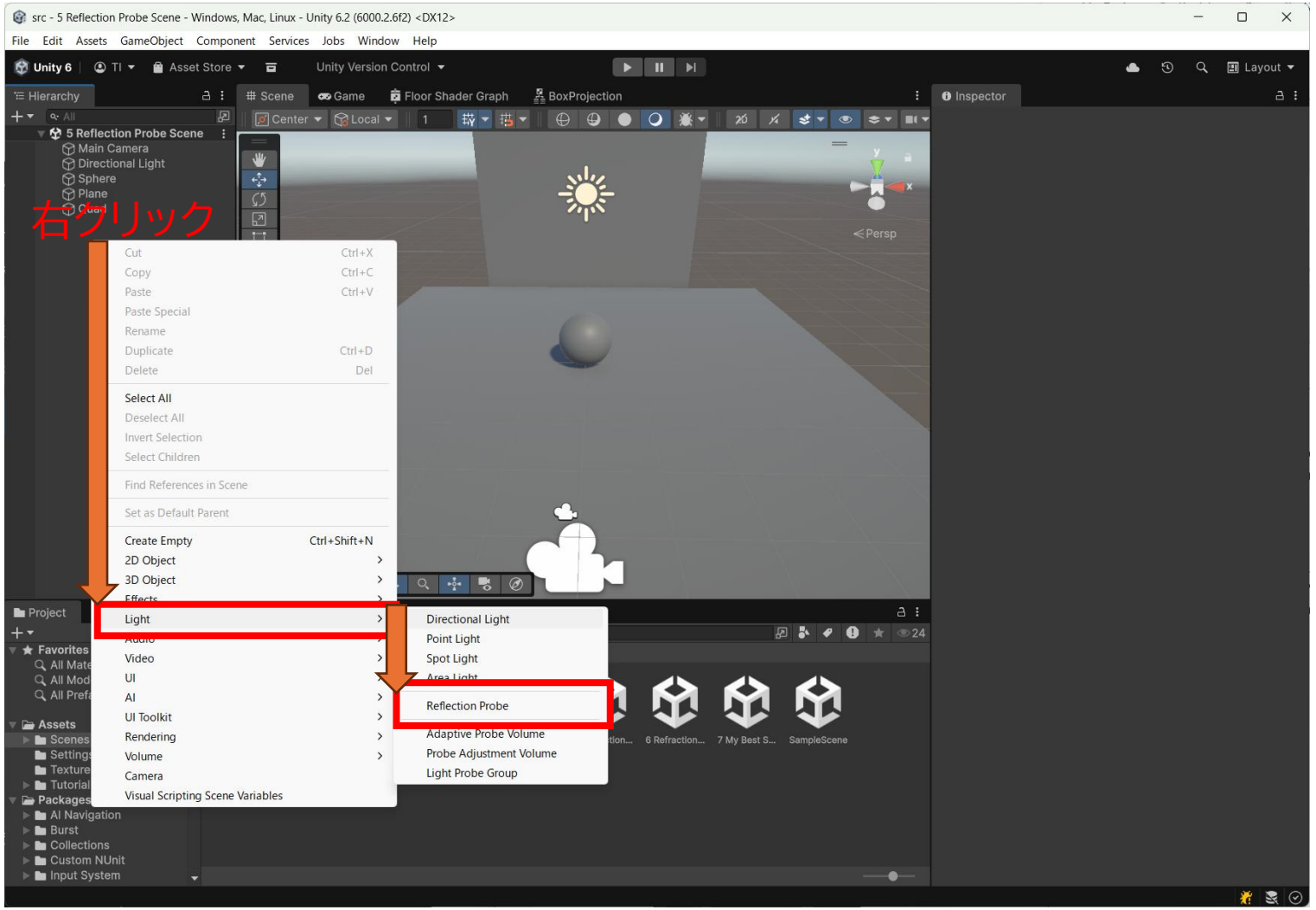
https://docs.unity3d.com/ja/2021.3/Manual/class-ReflectionProbe.html

リフレクションプローブの追加

- Hierarchyから、「Light」-「Reflection Probe」を追加

- 球で表示されるオブジェクトが追加される

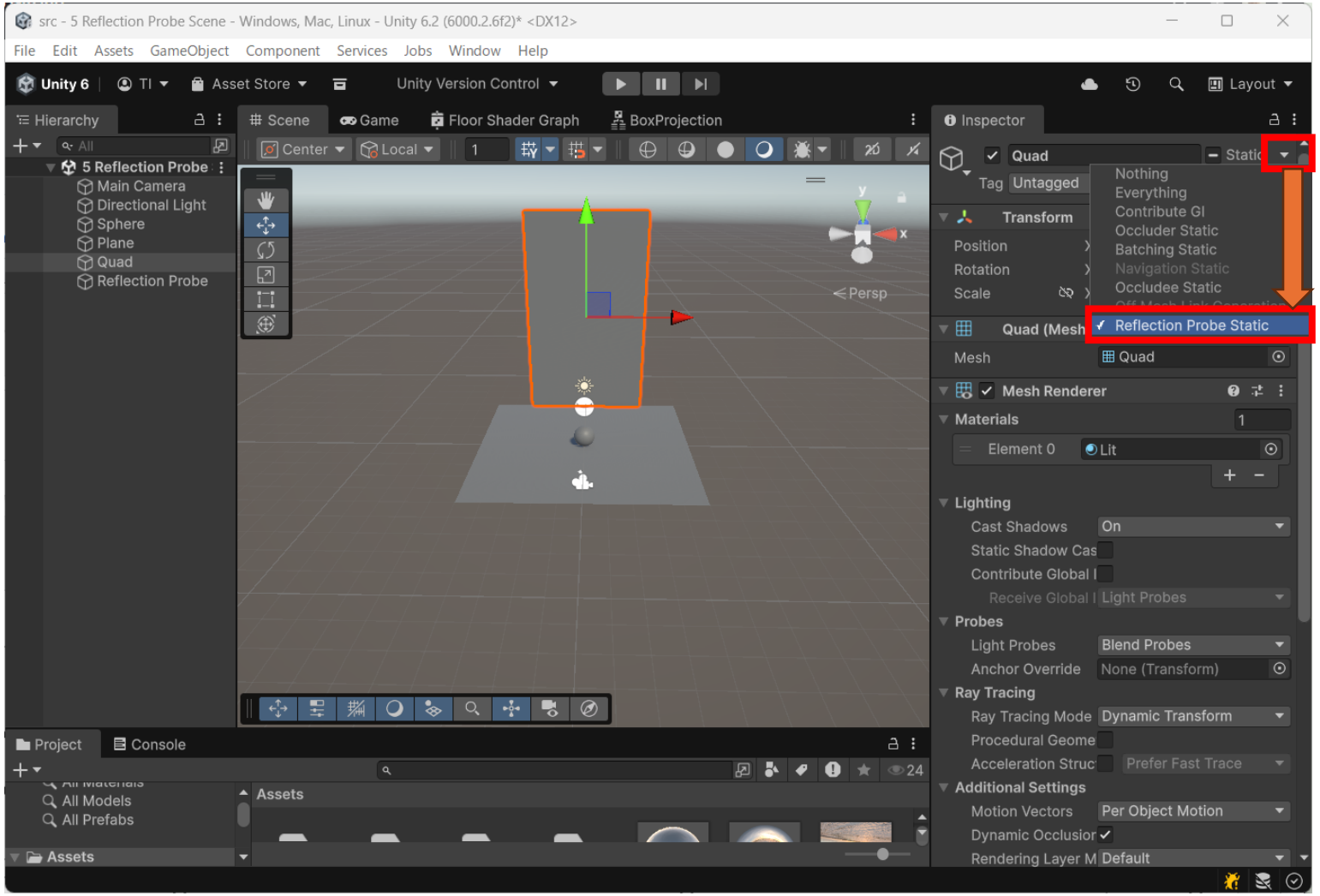




壁をキューブマップに描画するように設定

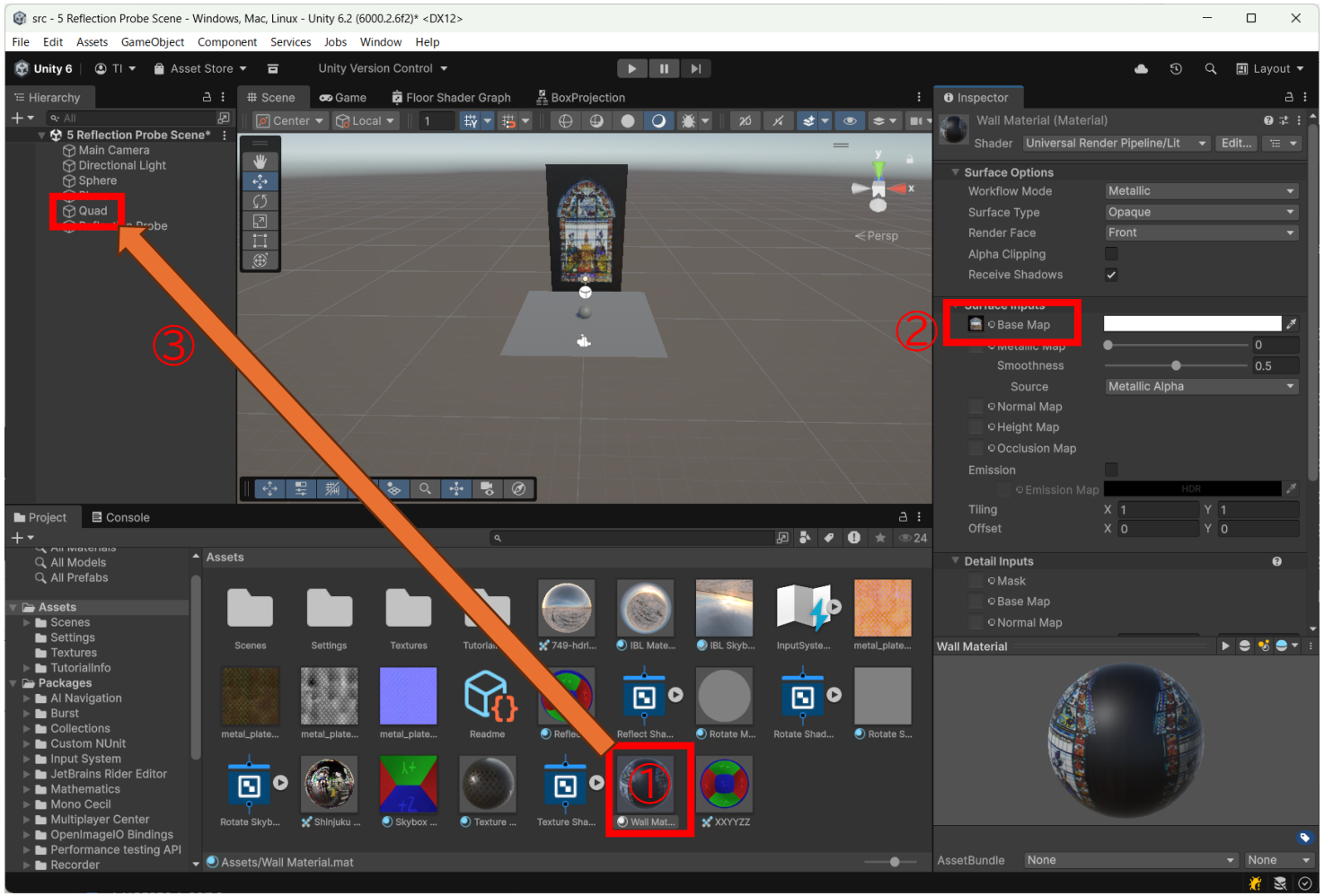
壁:「Quad」オブジェクト

- 「Static」の「Reflection Probe Static」を選択してチェックを入れる
  - キューブマップにキャプチャーさせる静的設定



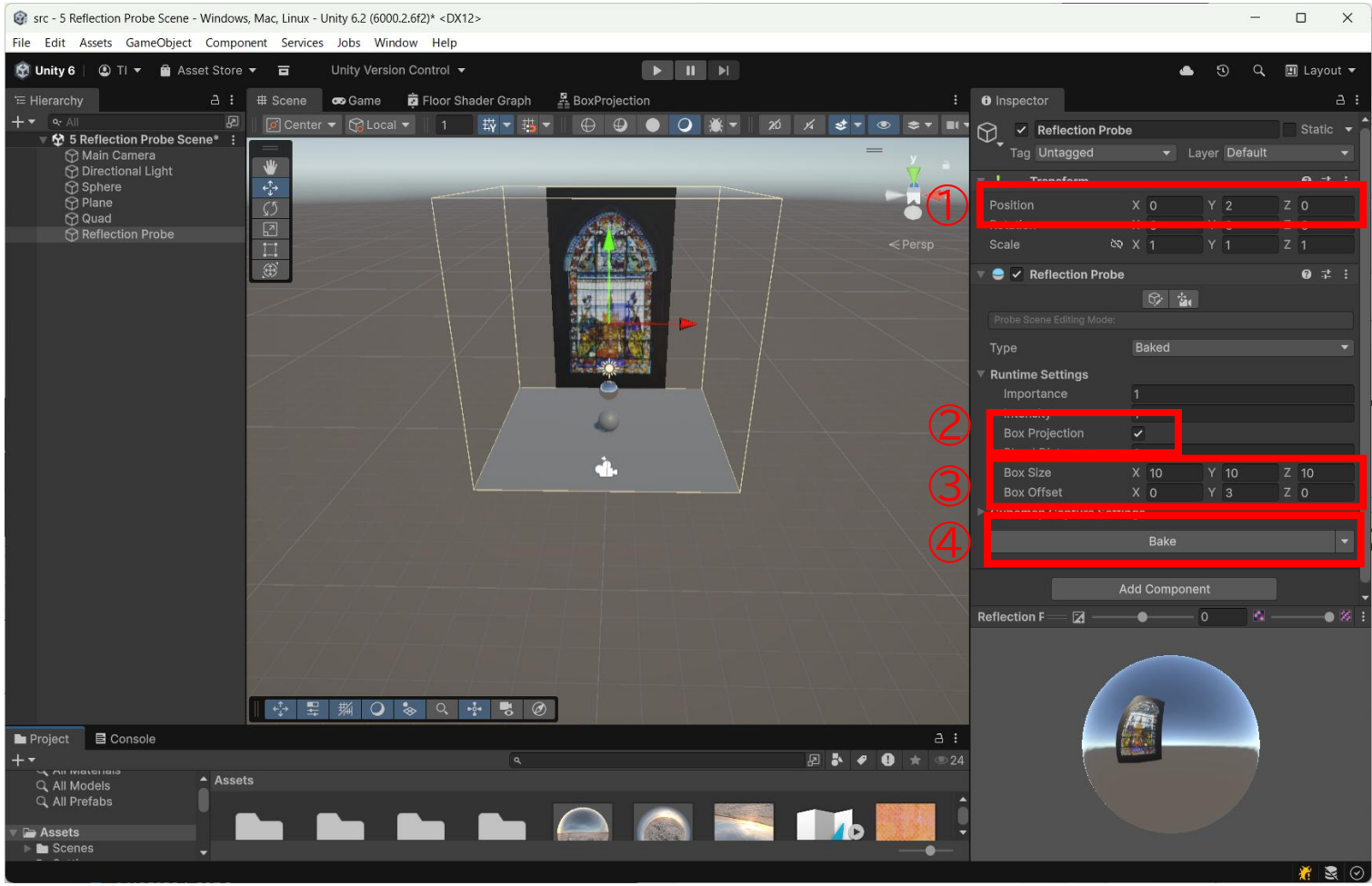
壁にテクスチャを張る

1. マテリアルを追加
   - 名前「Wall Material」等
2. マテリアルの「Base Map」にテクスチャを設定
   - 「Textures」フォルダに「Stained-glass_window,_Cathedral_in_Seville,_1685.jpg」ファイルがあります
3. マテリアルをオブジェクトに追加



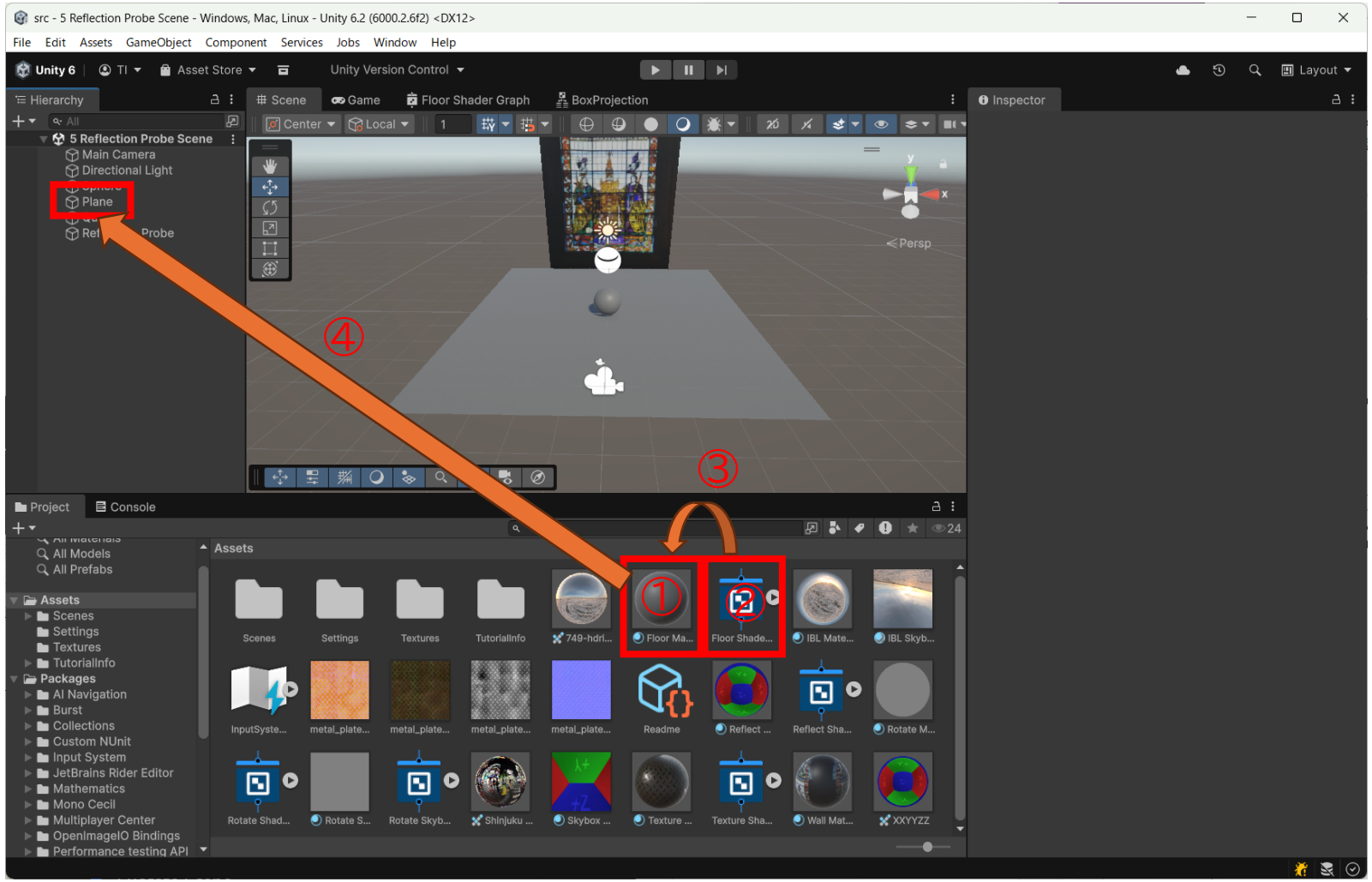
Parallax-corrected Cubemap用の調整

1. 位置(0,2,0)は、キューブマップを作成する仮想カメラの位置
2. 「Box Projection」を有効にする
3. 「Box Size」と「Box Offset」で想定する部屋の大きさを決める
4. 「Bake」ボタンを押す



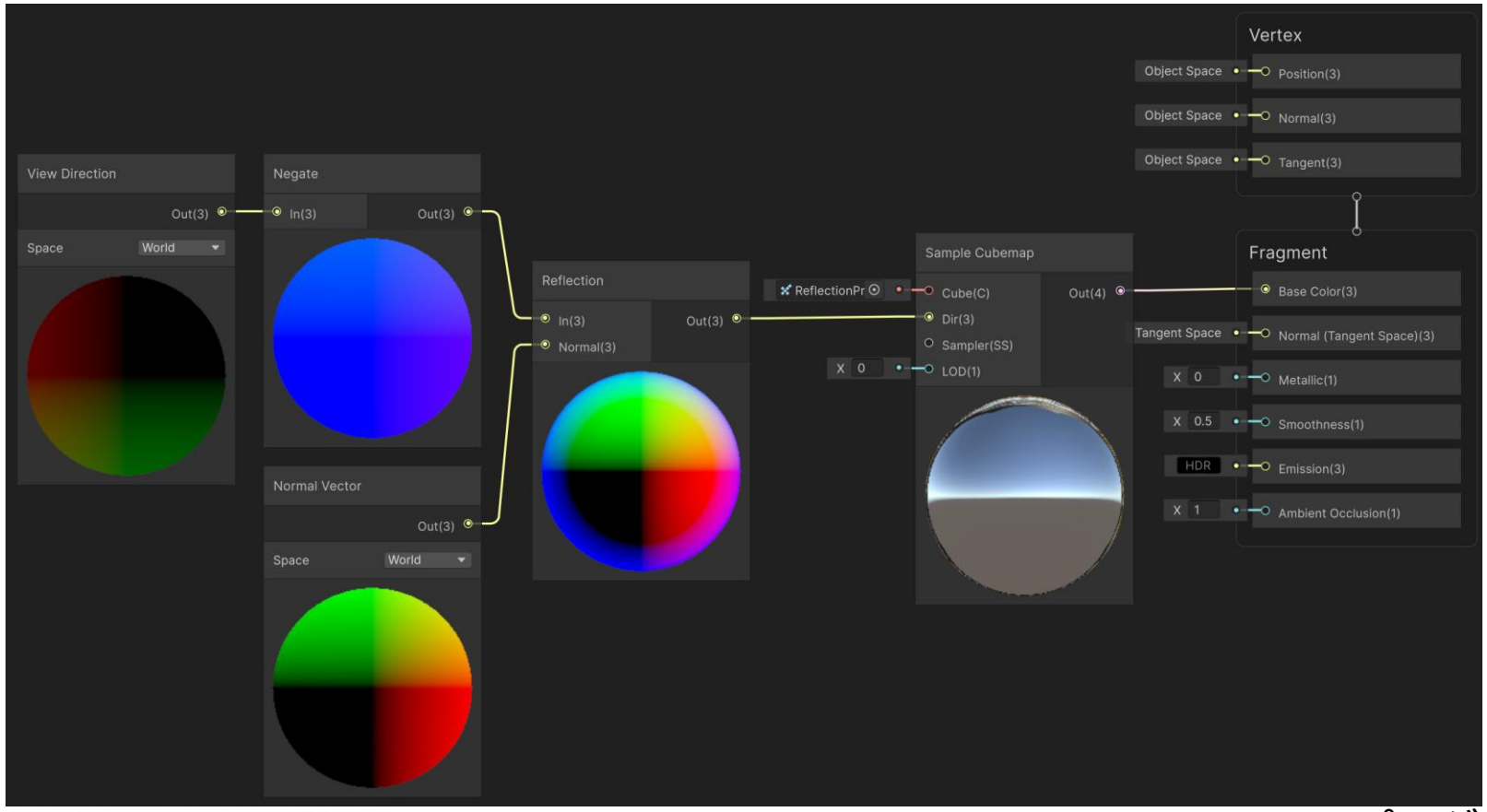
床用のマテリアルを準備する

1. マテリアルを作成
   - 名前は、「Floor Material」等
2. Shader Graphを作成
   1. 名前は、「Floor Shader Graph」等
3. Floor Shader GraphをFloor Materialに設定
4. Floor Materialを床のモデル（Plane）に設定

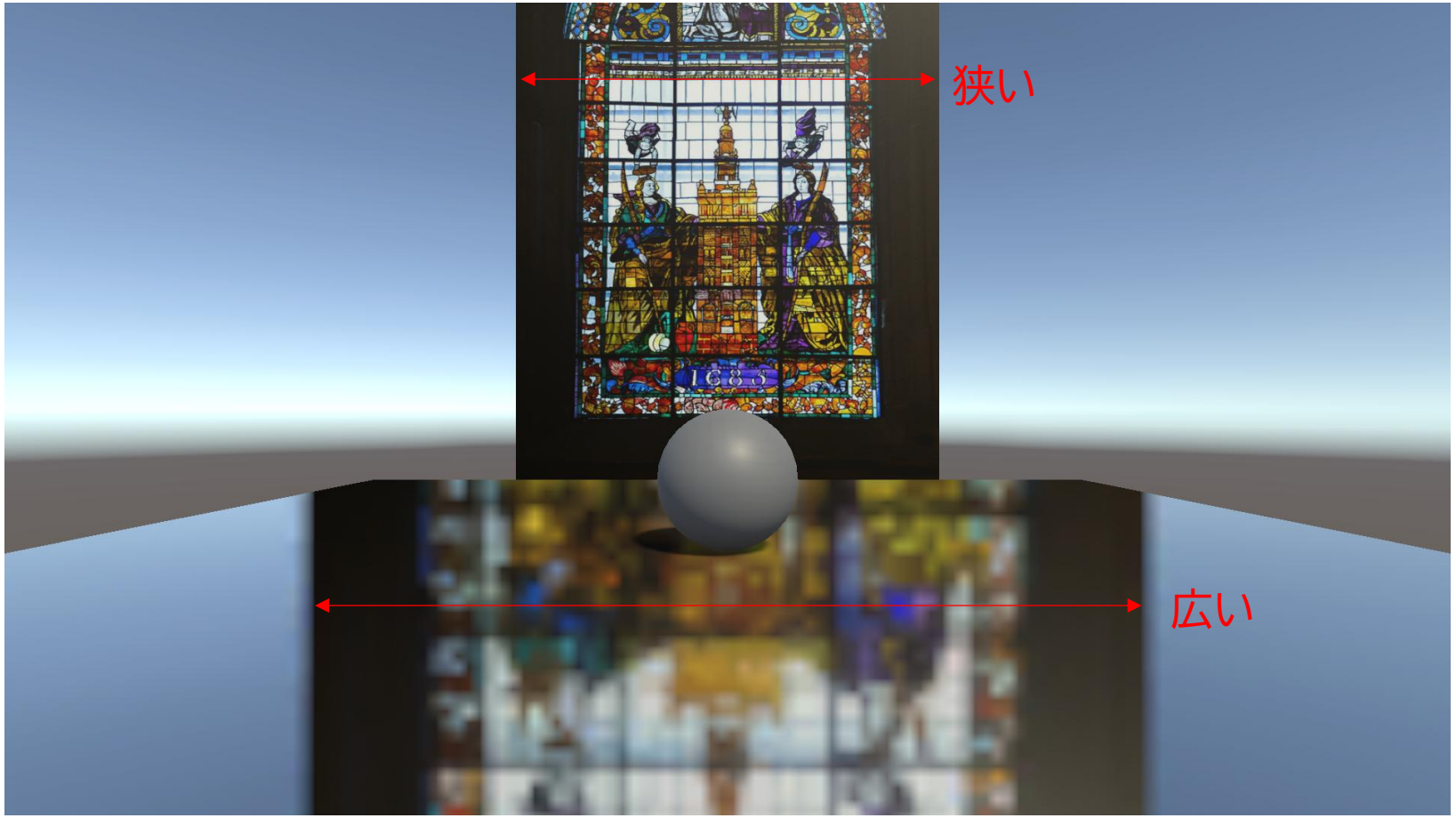


Floor Shader Graphの実装

- 反射ベクトルでのキューブマップの参照(まだ視差補正はない)



問題：反射した画像があっていない



シェーダサブグラフを追加して補正する

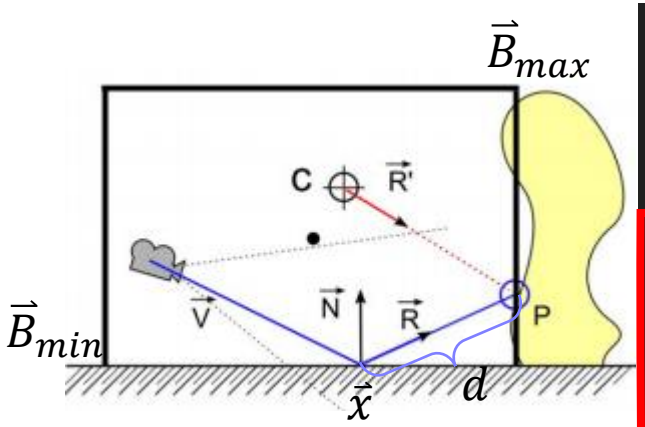
追加

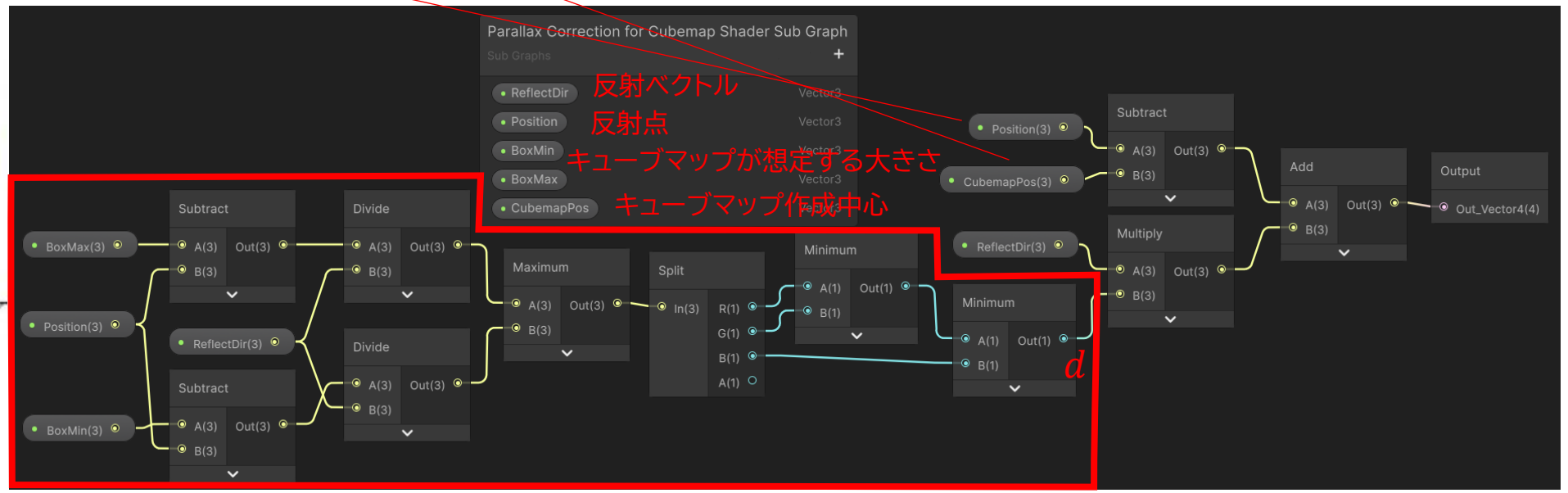
- $\vec{R'} = \vec{P} - \vec{C}$
- $\vec{P} = \vec{x} + d\vec{R}$

正の値(正面を向いている箱面)を選択

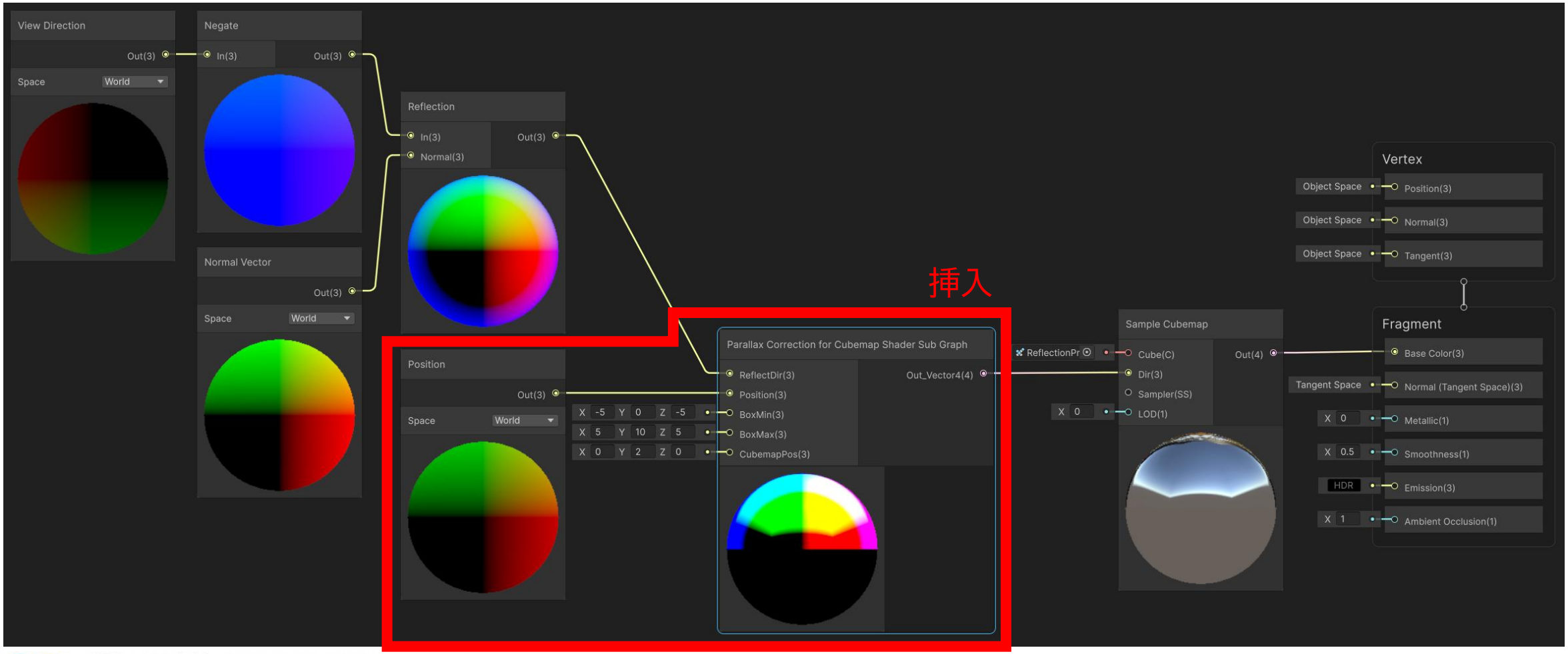
$$d = \min\left(\max\left(\frac{\vec{B}^x_{max} - \vec{x}^x}{\vec{R}^x}, \frac{\vec{B}^x_{min} - \vec{x}^x}{\vec{R}^x}\right), \max\left(\frac{\vec{B}^y_{max} - \vec{x}^y}{\vec{R}^y}, \frac{\vec{B}^y_{min} - \vec{x}^y}{\vec{R}^y}\right), \max\left(\frac{\vec{B}^z_{max} - \vec{x}^z}{\vec{R}^z}, \frac{\vec{B}^z_{min} - \vec{x}^z}{\vec{R}^z}\right)\right)$$

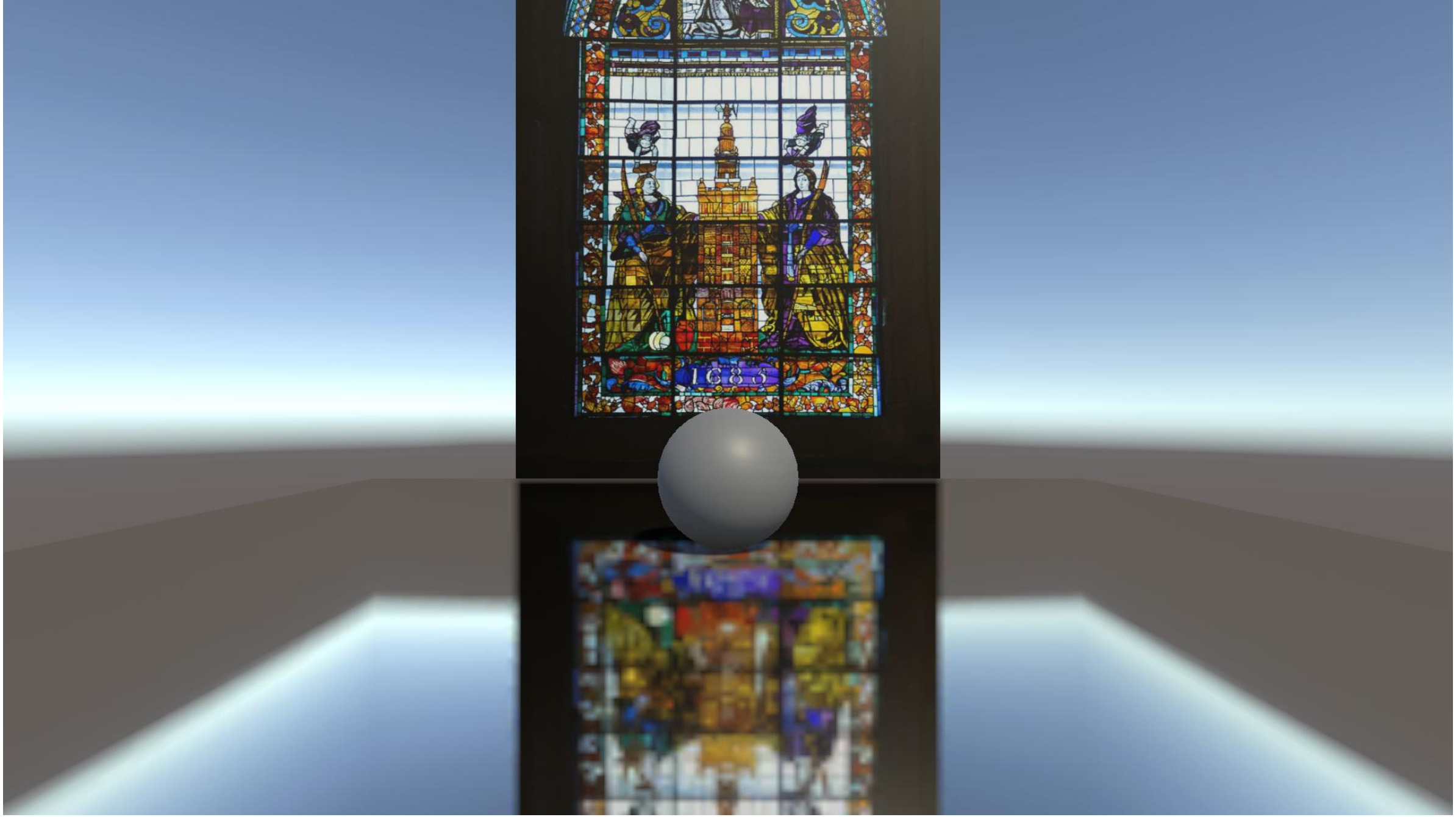
:一番近い壁までの距離



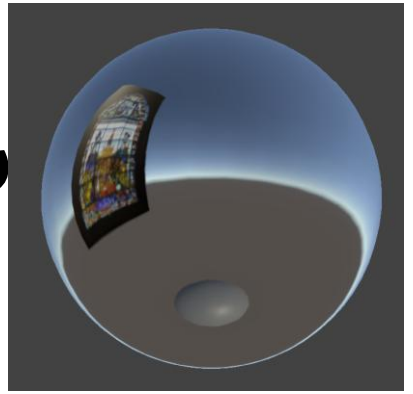


シェーダサブグラフを使う（Floor Shader Graph）

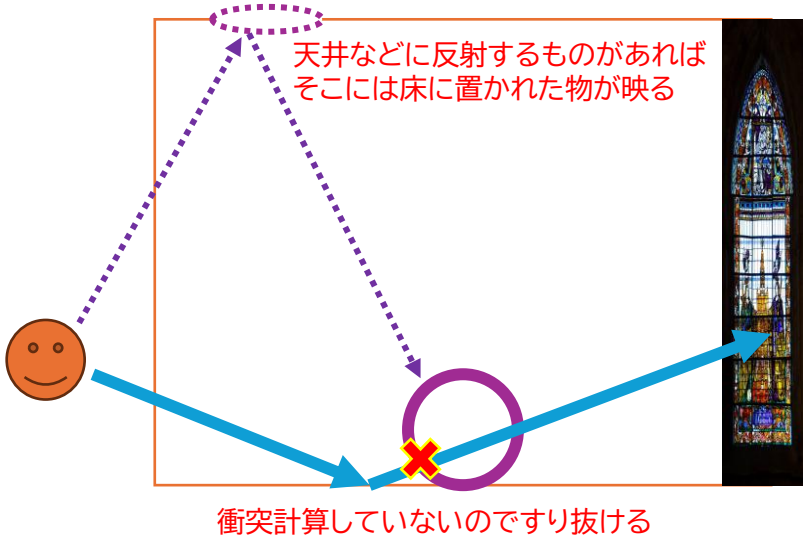


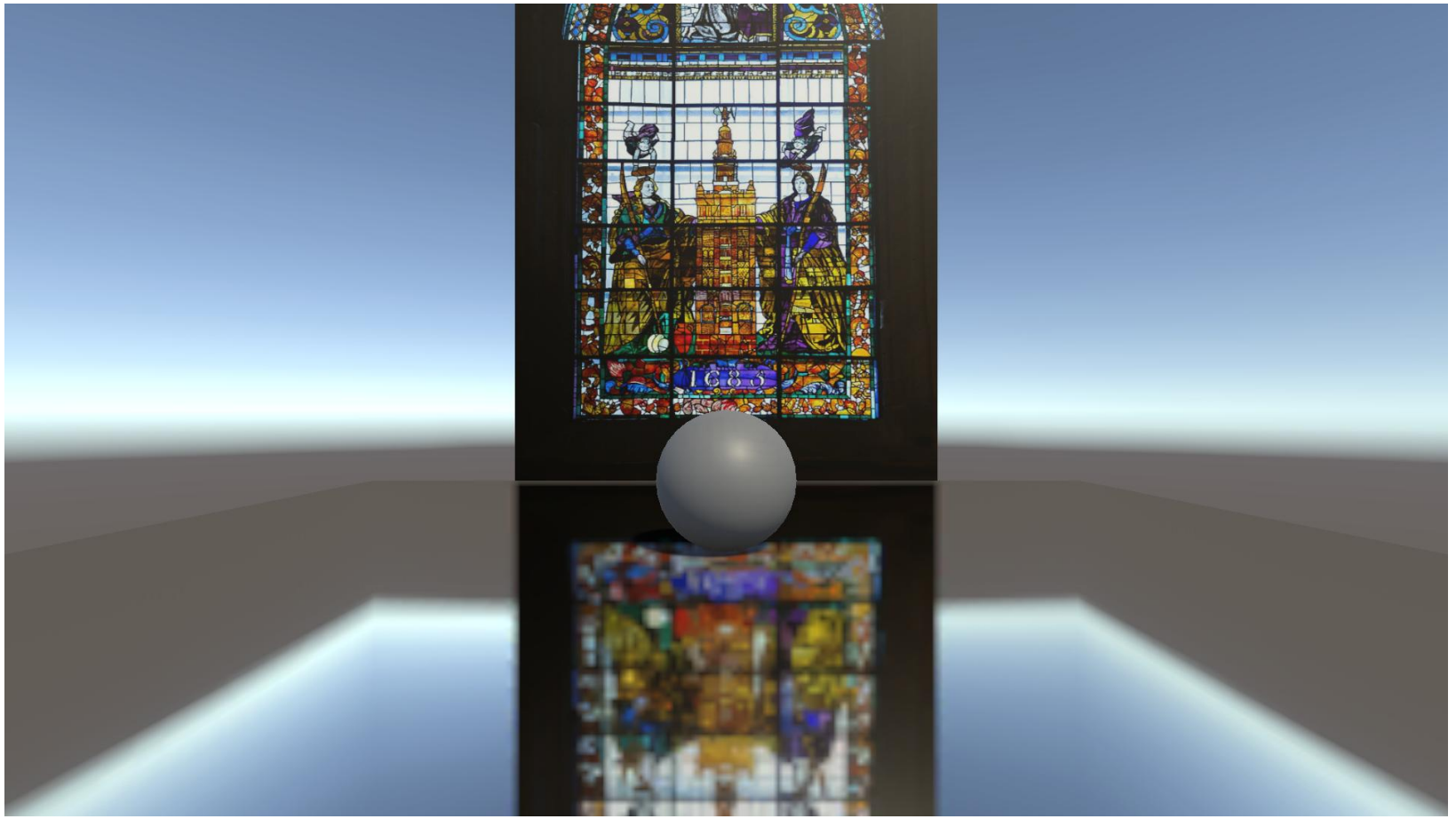

1685

なお、床をキューブマップに描画しても映らない



- 床が描画しようとするのは物体ではなく、キューブマップに映された絵



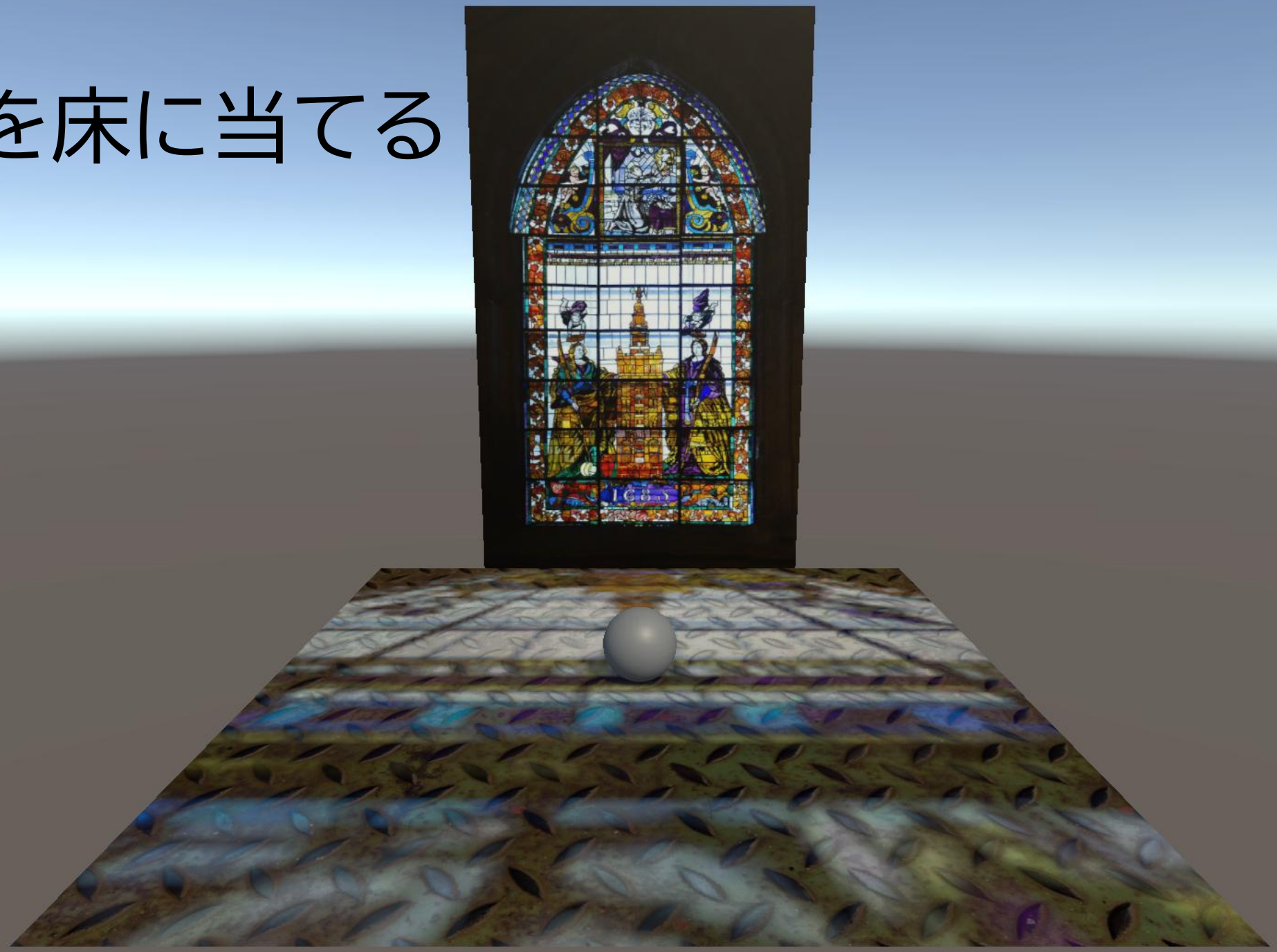



ゴール:床を反射させよう
やってみよう

レジュメ

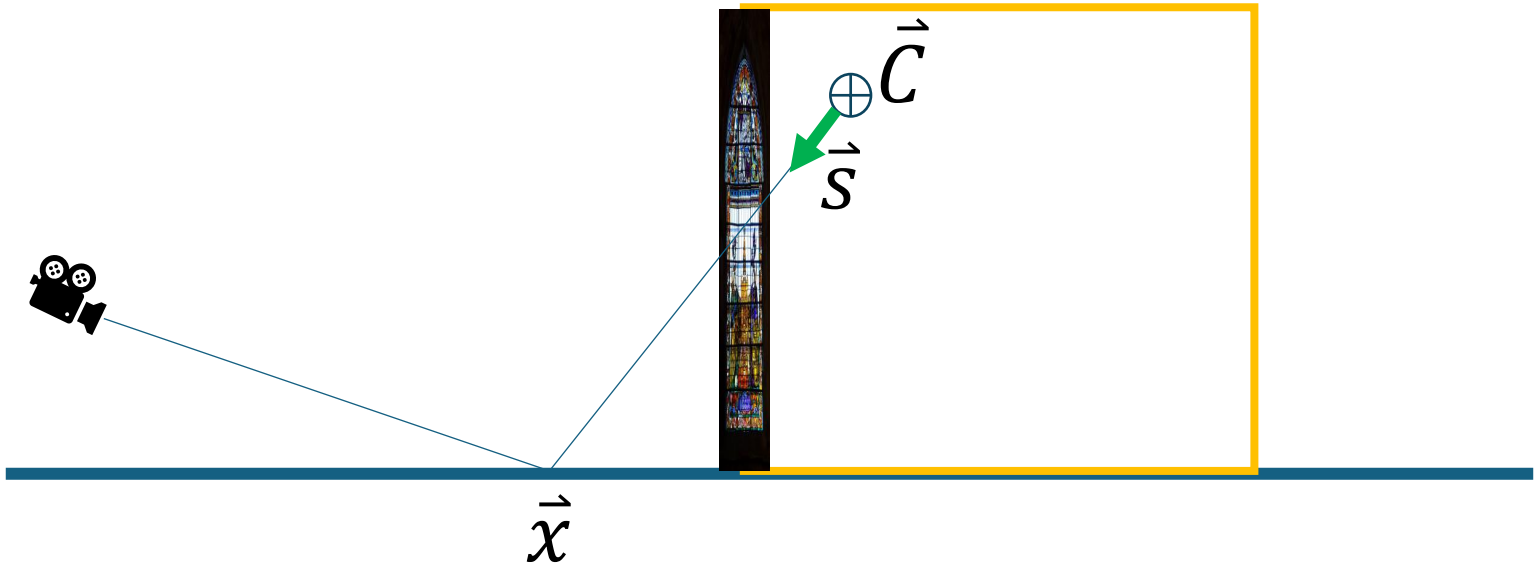
- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
    - 床反射
    - 投影
    - 投影を動かす
  - 屈折

光を床に当てる



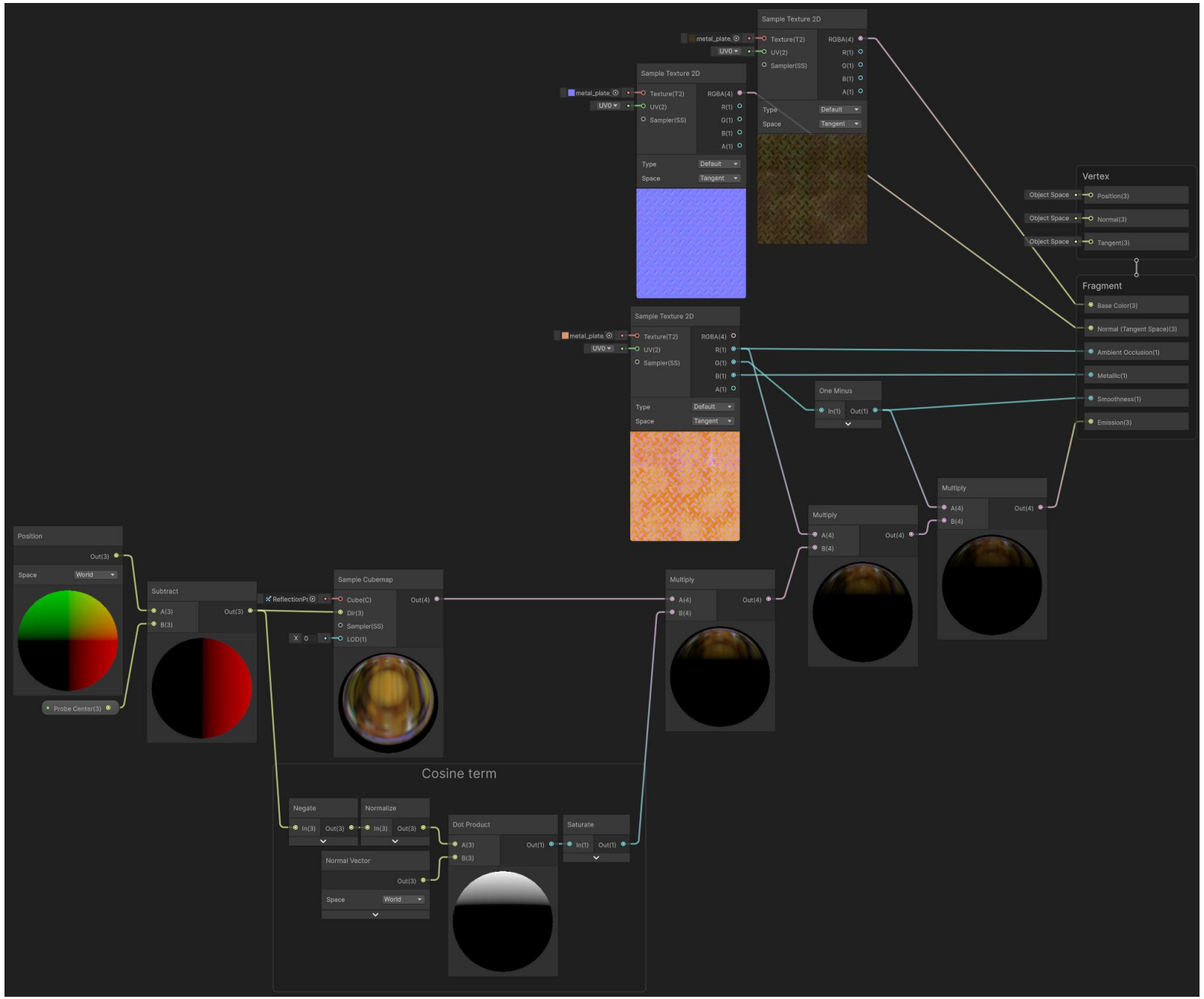
サンプリング方向

- プローブの中心からレンダリングする位置への向き
  - $\vec{s} = \frac{\vec{x}-\vec{C}}{\|\vec{x}-\vec{C}\|}$
    - サンプリング時は正規化は不要

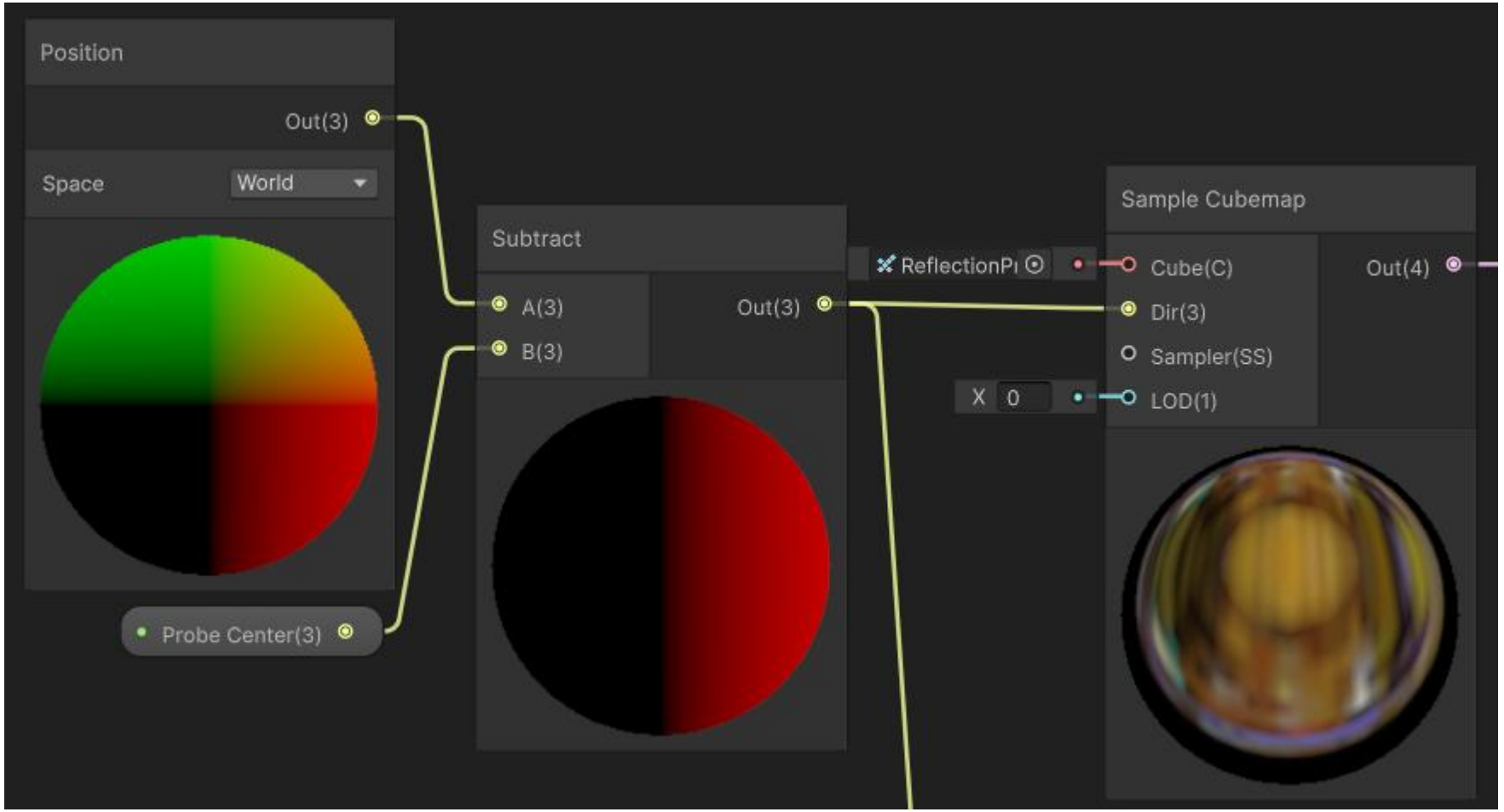


Shader Graph

- Lit Shader GraphのEmissionとして光を追加

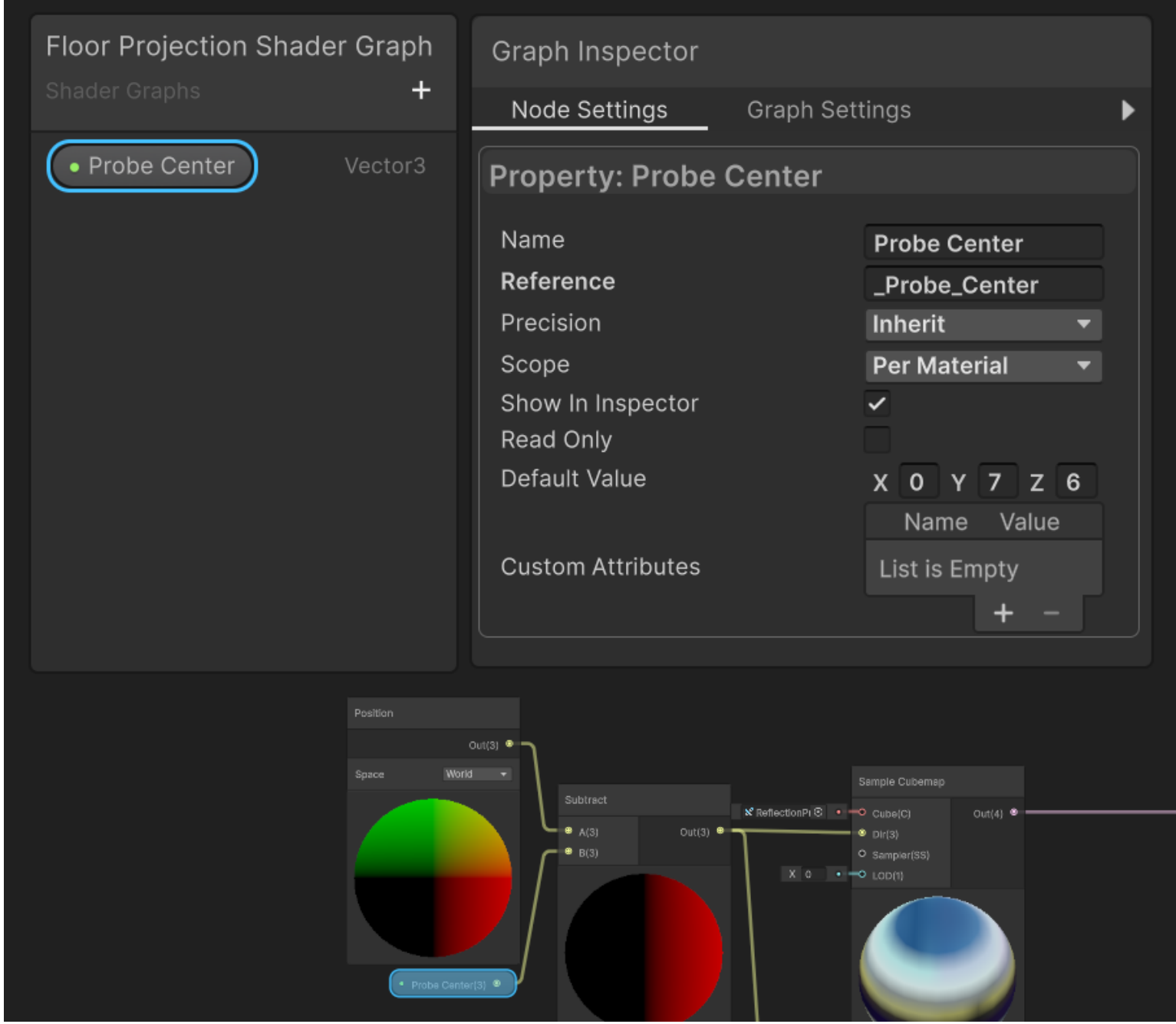


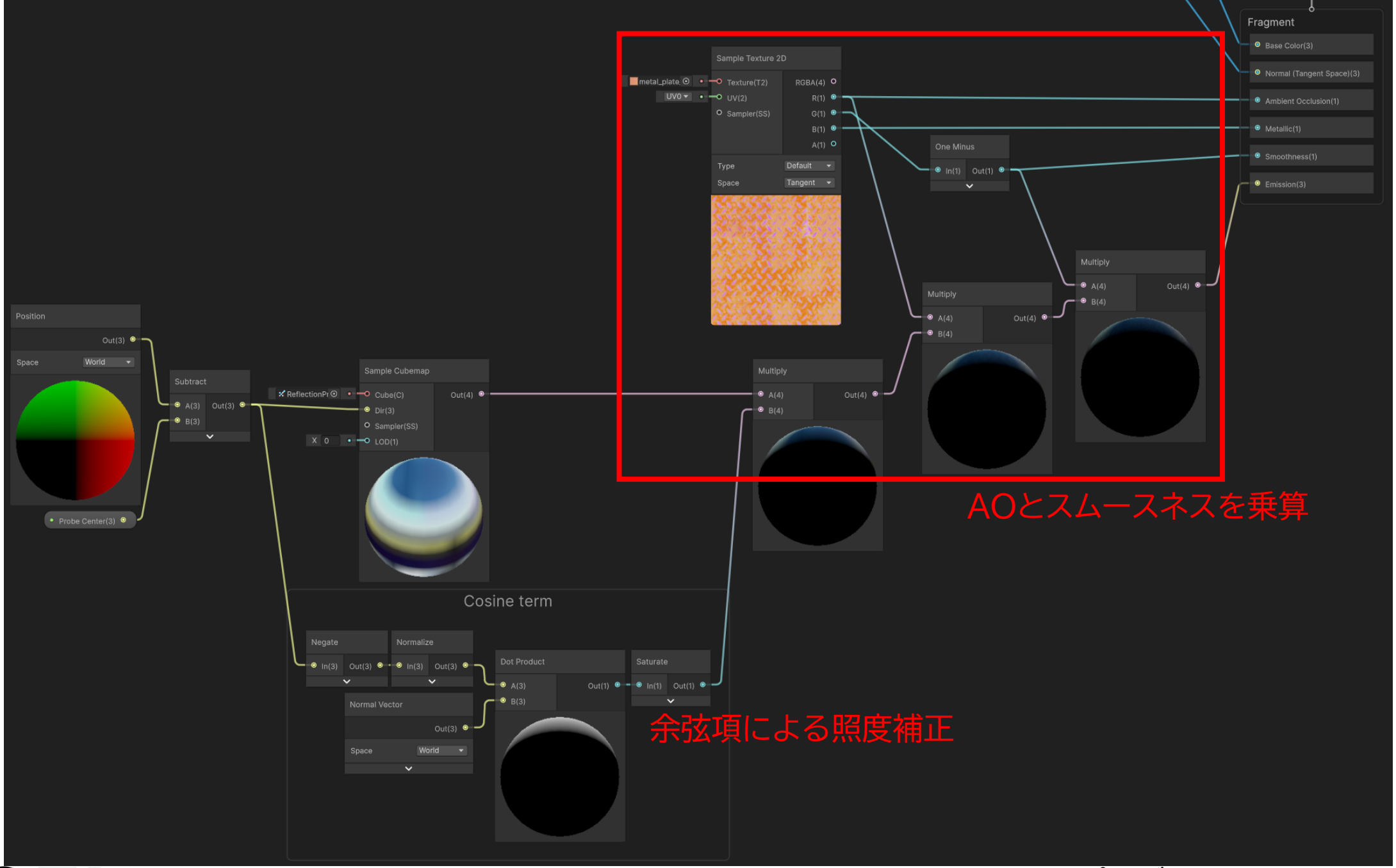
光の投影


Position
Out(3)
Space
World
Probe Center(3)
Subtract
A(3)
B(3)
Out(3)
ReflectionPr
X 0
Sample Cubemap
Cube(C)
Dir(3)
Sampler(SS)
LOD(1)
Out(4)

プロパティ

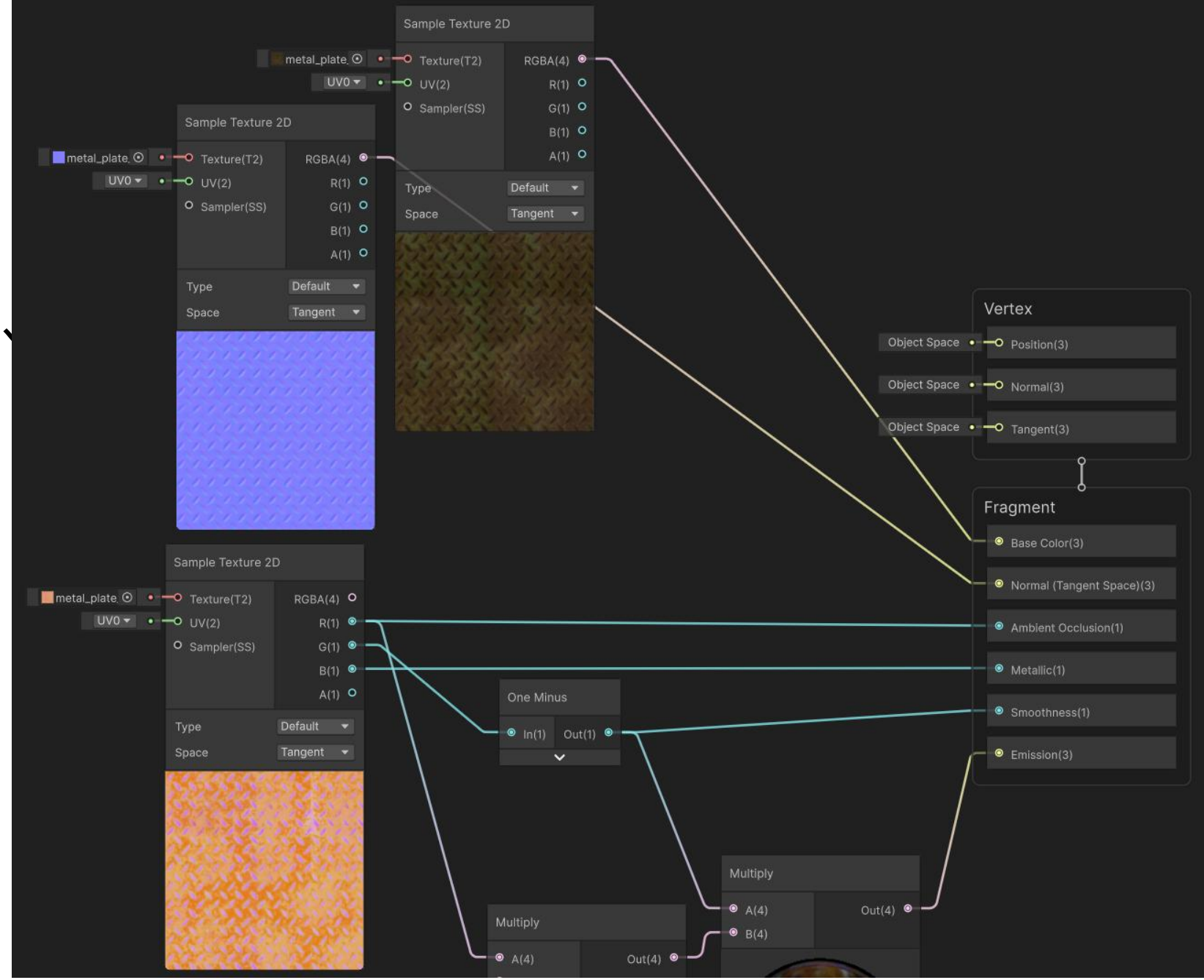
- Probe Center
  - Vector3
  - プローブ作成時のカメラ位置




Fragment
Base Color(3)
Normal (Tangent Space)(3)
Ambient Occlusion(1)
Metallic(1)
Smoothness(1)
Emission(3)
Sample Texture 2D
metal_plate
UV0
Texture(T2)
UV(2)
Sampler(SS)
RGBA(4)
R(1)
G(1)
B(1)
A(1)
Type
Default
Space
Tangent
One Minus
In(1)
Out(1)
Multiply
A(4)
B(4)
Out(4)
Position
Out(3)
Space
World
Probe Center(3)
Subtract
A(3)
B(3)
Out(3)
ReflectionPr
Sample Cubemap
Cube(C)
Dir(3)
Sampler(SS)
LOD(1)
X 0
Out(4)
AOとスムースネスを乗算
Cosine term
Negate
In(3)
Out(3)
Normalize
Dot Product
Saturate
Normal Vector
Space
World
余弦項による照度補正

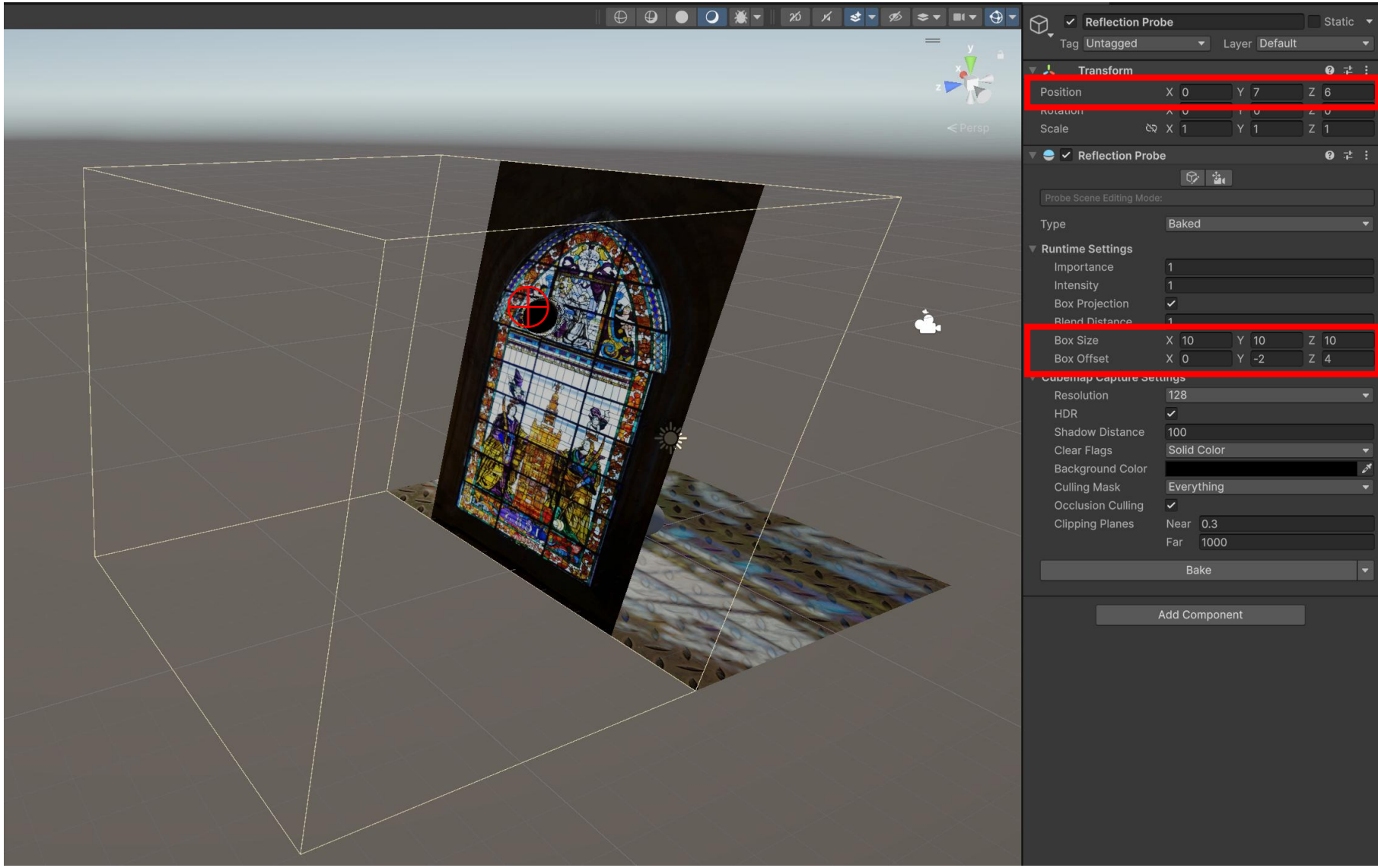
PBR
テクスチャ

- アルベド、法線マップ、AO,スムースネス、メタリックを適応
  - 通常の表面計算



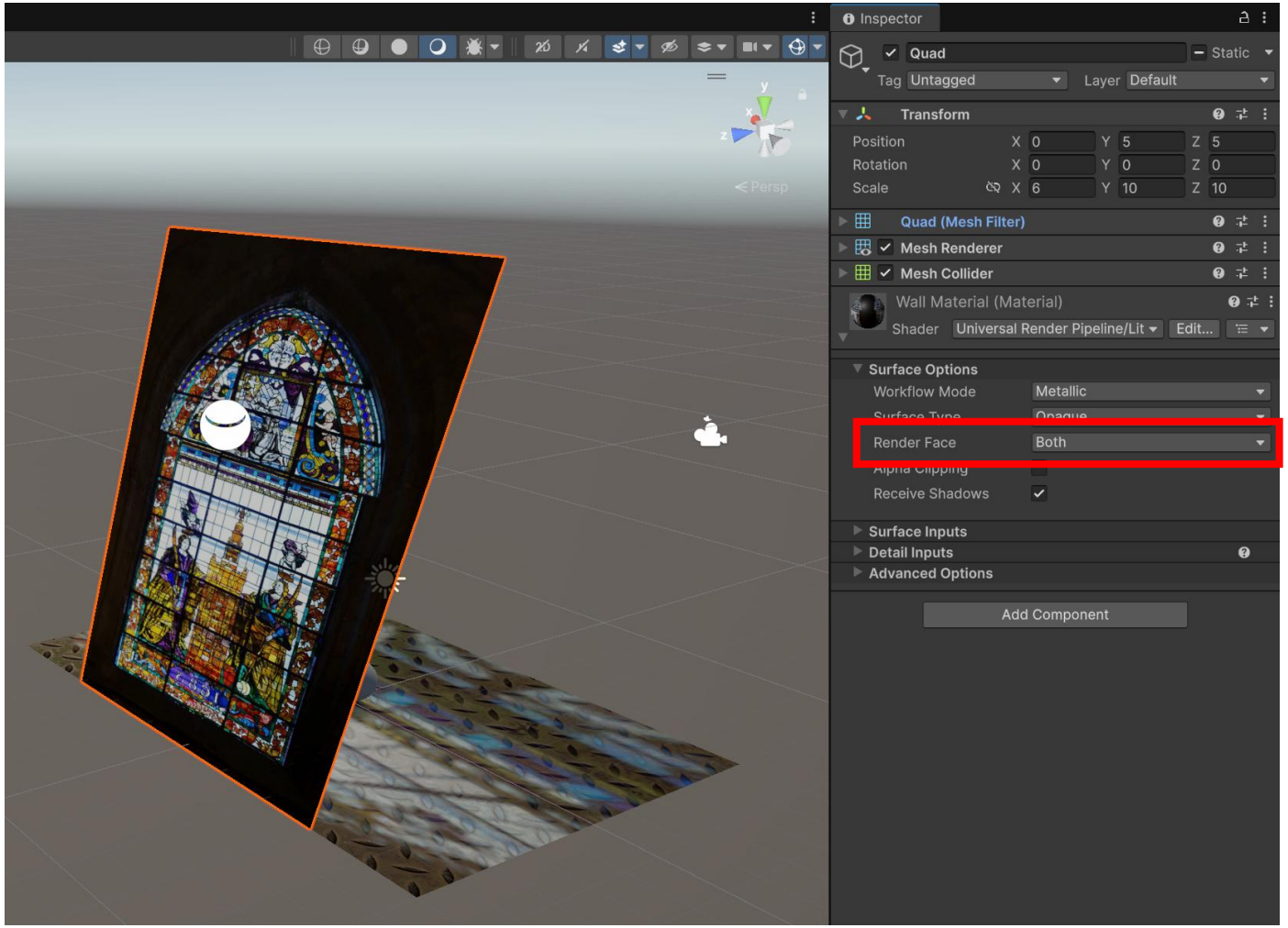
プローブ

- 窓の裏に設置
- テクスチャに
  なるべく大きく
  写るように、
  プローブ中心を
  窓に近づける



窓

- 両面描画
  - 窓の裏からレンダリングした際も、描画されるようにする

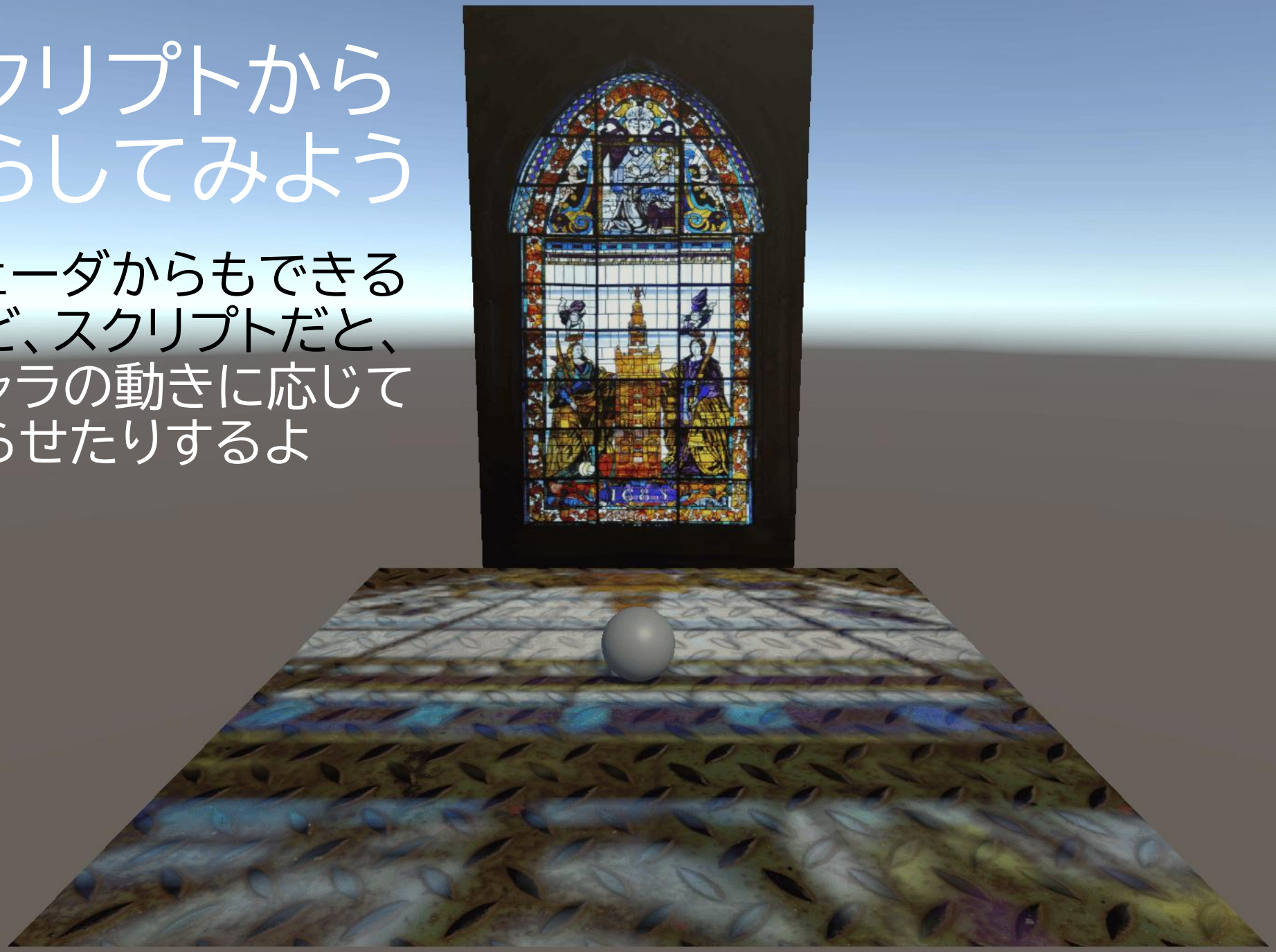


レジュメ

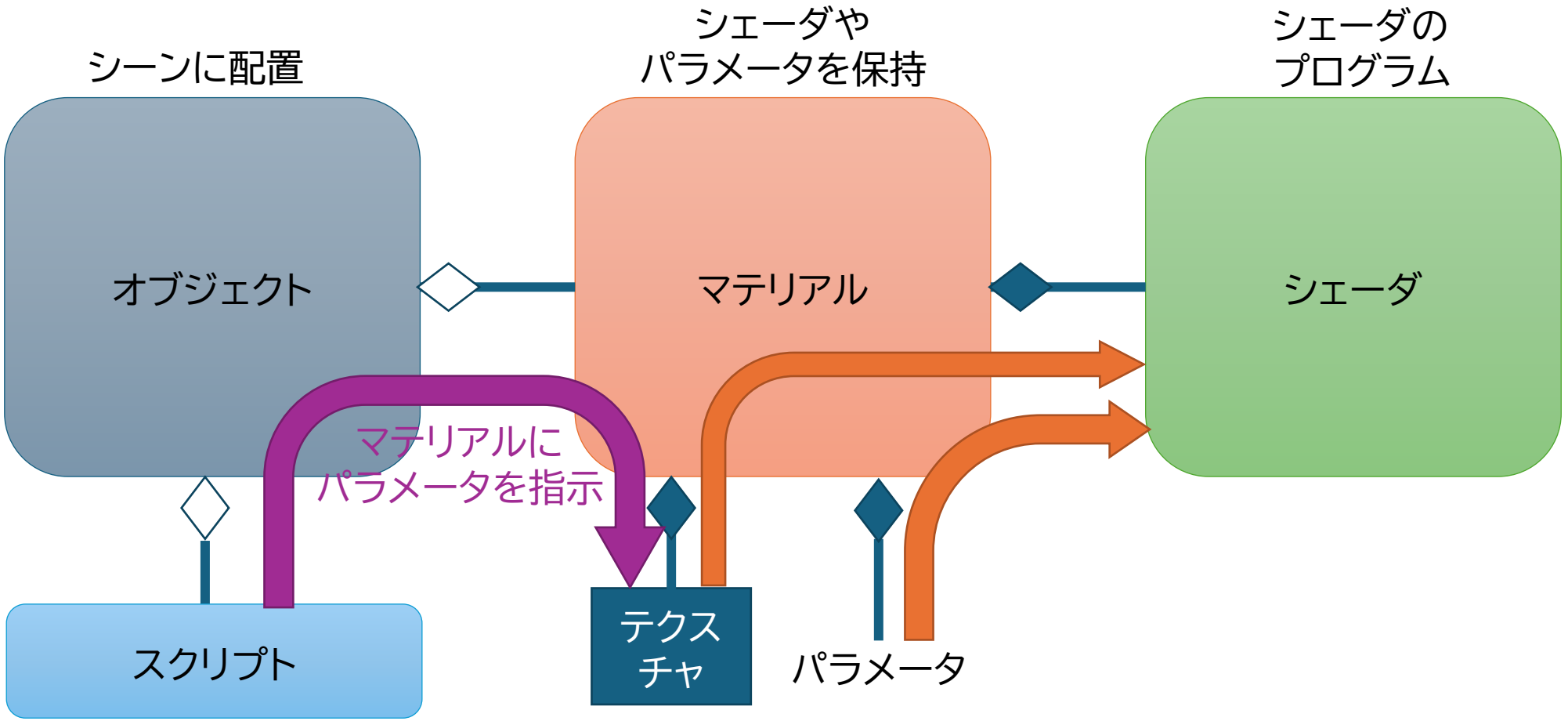
- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
    - 床反射
    - 投影
    - 投影を動かす：スクリプトからのシェーダパラメータの操作
  - 屈折

スクリプトからゆらしてみよう

- シェーダからもできるけど、スクリプトだと、キャラの動きに応じてゆらせたりするよ

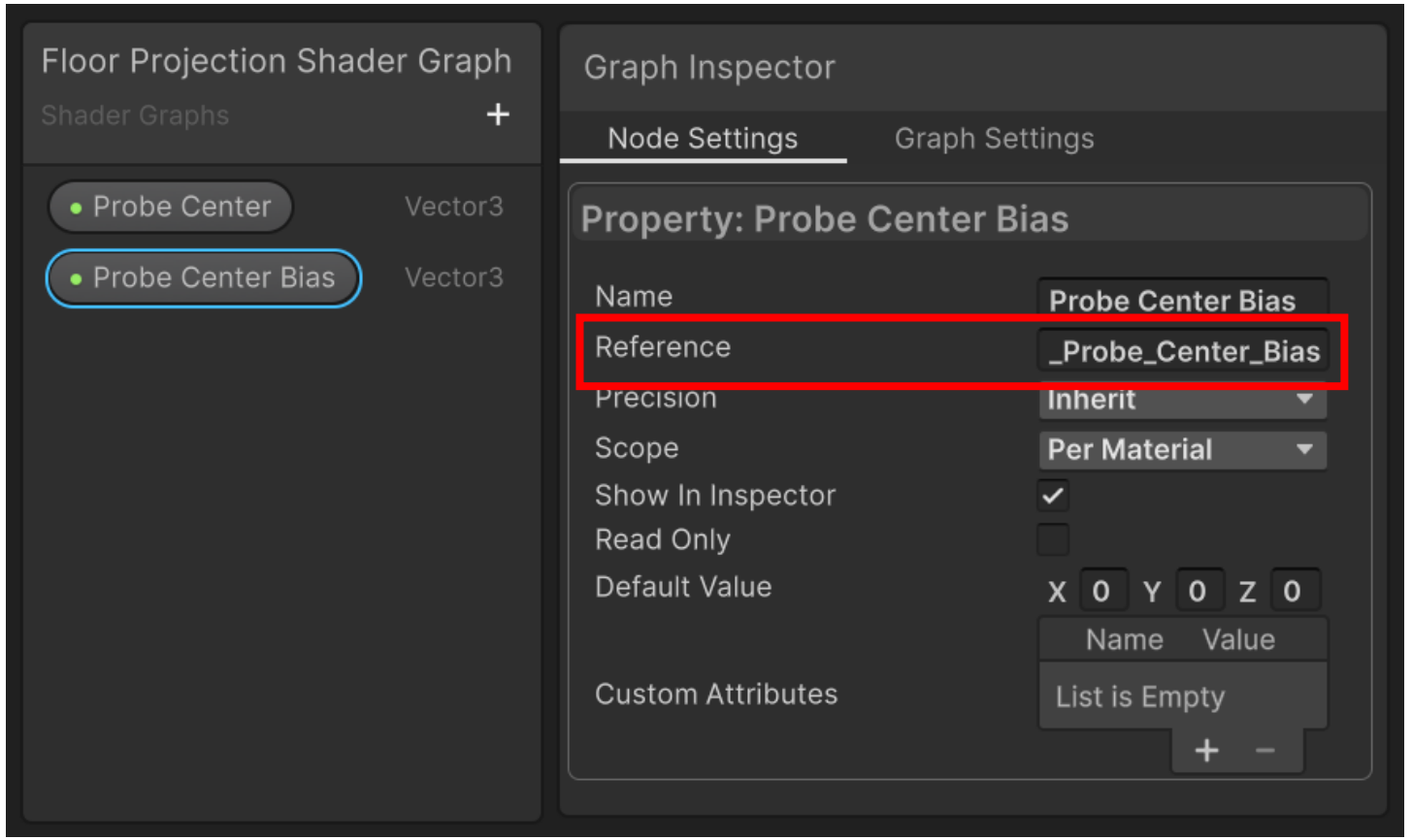


Unityでのシェーダの制御



シェーダグラフのプロパティ

- 「Reference」に書かれている名前でアクセスする

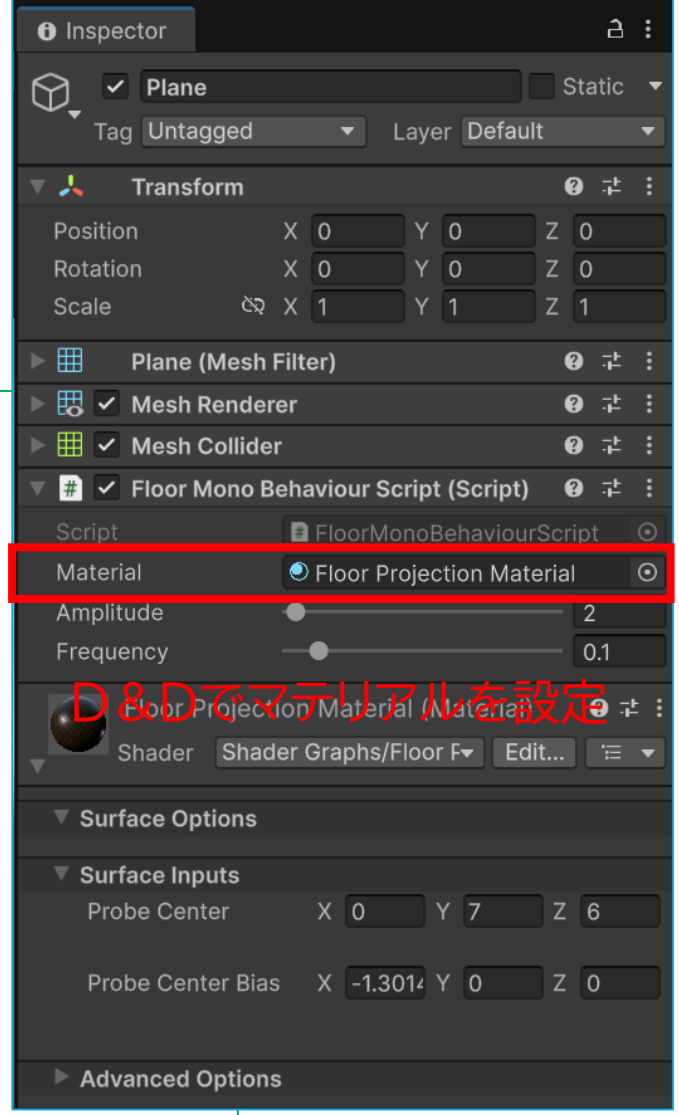


スクリプトからの設定

```
1   using UnityEngine;
2
    Unity スクリプト (1 件のアセット参照)|0 個の参照
3   public class FloorMonoBehaviourScript : MonoBehaviour
4   {
5       [SerializeField] Material material = default!;
6       [SerializeField, Range(0.0f, 100.0f)] float Amplitude = 2.0f;
7       [SerializeField, Range(0.0f, 1.0f)] float Frequency = 0.1f;
8       float Angle = 0.0f;
9
        Unity メッセージ|0 個の参照
10      void Update()
11      {
12          Angle += Time.deltaTime;
13
14          float x = Amplitude * Mathf.Sin(2.0f * Mathf.PI * Frequency * Angle);
15          material.SetVector("_Probe_Center_Bias", new Vector3(x, 0, 0));
16      }
17  }
```

マテリアルの保持

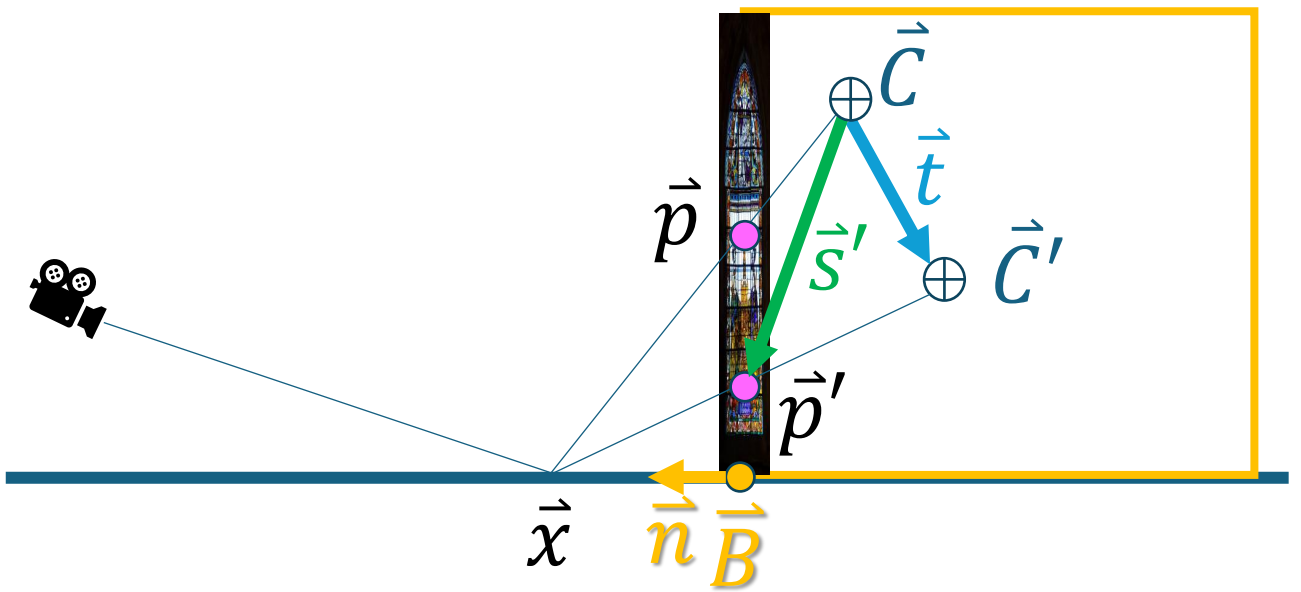
型に応じたメソッドでプロパティを設定

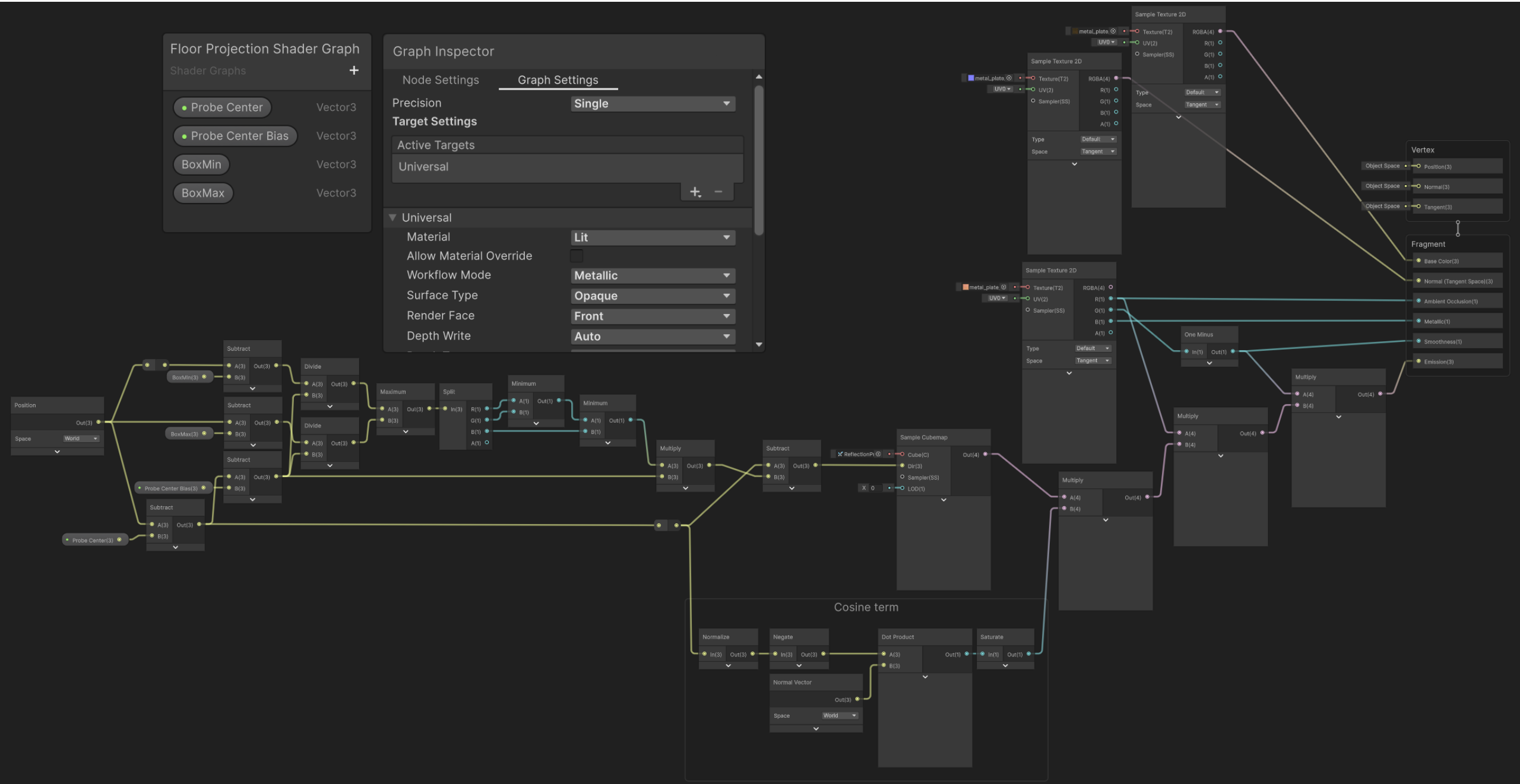


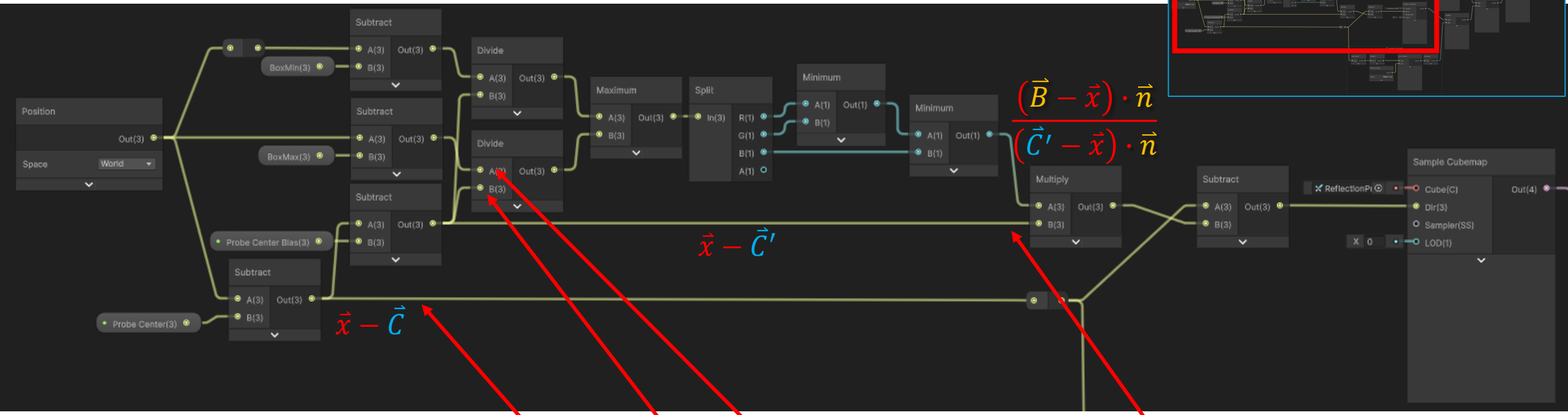
プローブ中心を移動した際の向きを使用

- $\vec{C}$中心が、$\vec{t}$だけ移動した際の描画される場所$\vec{p}'$を求め、その位置がサンプリングできる方向$\vec{s}'$を導出する
  - $\vec{s}' = \vec{p}' - \vec{C}$
  - $\vec{p}' = \vec{x} + \frac{(\vec{B}-\vec{x})\cdot\vec{n}}{(\vec{C}'-\vec{x})\cdot\vec{n}}(\vec{C}' - \vec{x})$

➡ $\vec{s}' = \vec{x} - \vec{C} + \frac{(\vec{B}-\vec{x})\cdot\vec{n}}{(\vec{C}'-\vec{x})\cdot\vec{n}}(\vec{C}' - \vec{x})$



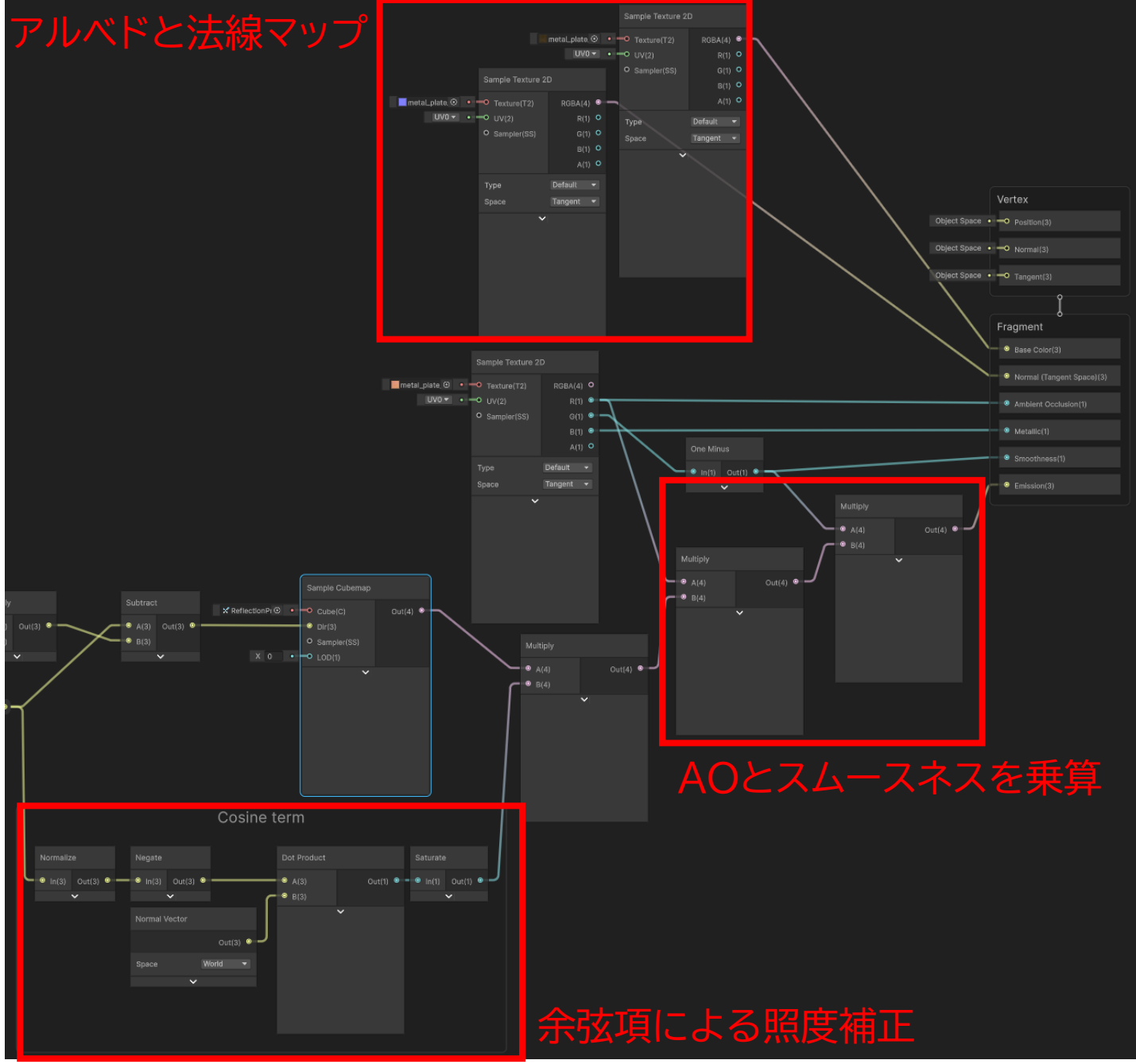

Floor Projection Shader Graph
Shader Graphs
Probe Center
Vector3
Probe Center Bias
Vector3
BoxMin
Vector3
BoxMax
Vector3
Graph Inspector
Node Settings
Graph Settings
Precision
Single
Target Settings
Active Targets
Universal
Universal
Material
Lit
Allow Material Override
Workflow Mode
Metallic
Surface Type
Opaque
Render Face
Front
Depth Write
Auto
Sample Texture 2D
Vertex
Fragment
Position
Subtract
Divide
Maximum
Split
Minimum
Multiply
Sample Cubemap
One Minus
Cosine term
Normalize
Negate
Dot Product
Saturate
Normal Vector



$$\vec{s}' = \vec{x} - \vec{C} + \frac{(\vec{B} - \vec{x}) \cdot \vec{n}}{(\vec{C}' - \vec{x}) \cdot \vec{n}} (\vec{C}' - \vec{x})$$

その他

- 静的な投影と同じ



レジュメ

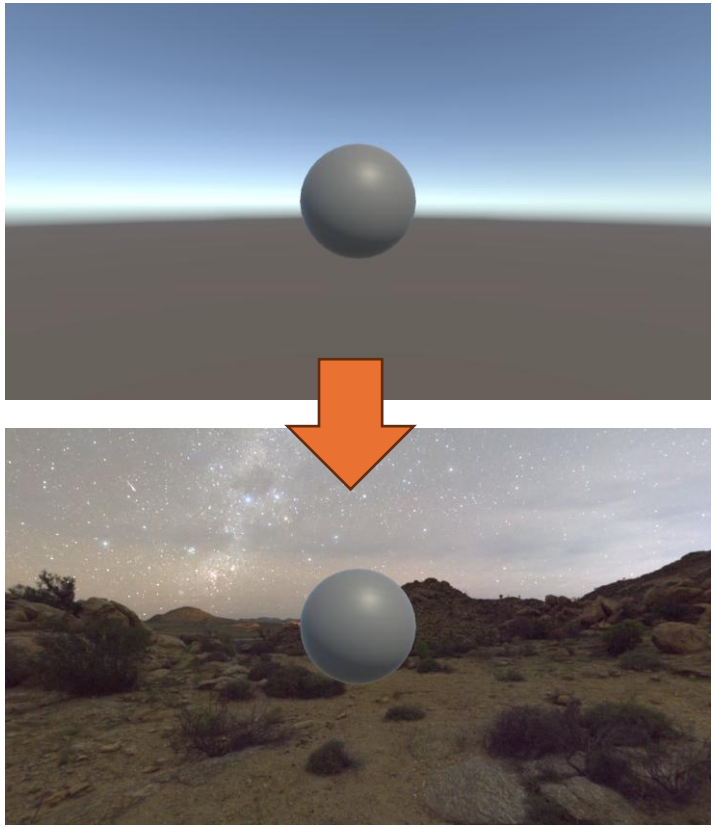
- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
  - キューブマップを動かす
  - 自分のキューブマップに差し替えてみよう
  - PBRでのキューブマップ
  - 反射プローブ
  - 屈折

屈折:アジェンダ

- 背景にキューブマップを設定
- オブジェクトに鏡面反射を付ける(フレネル効果付き)

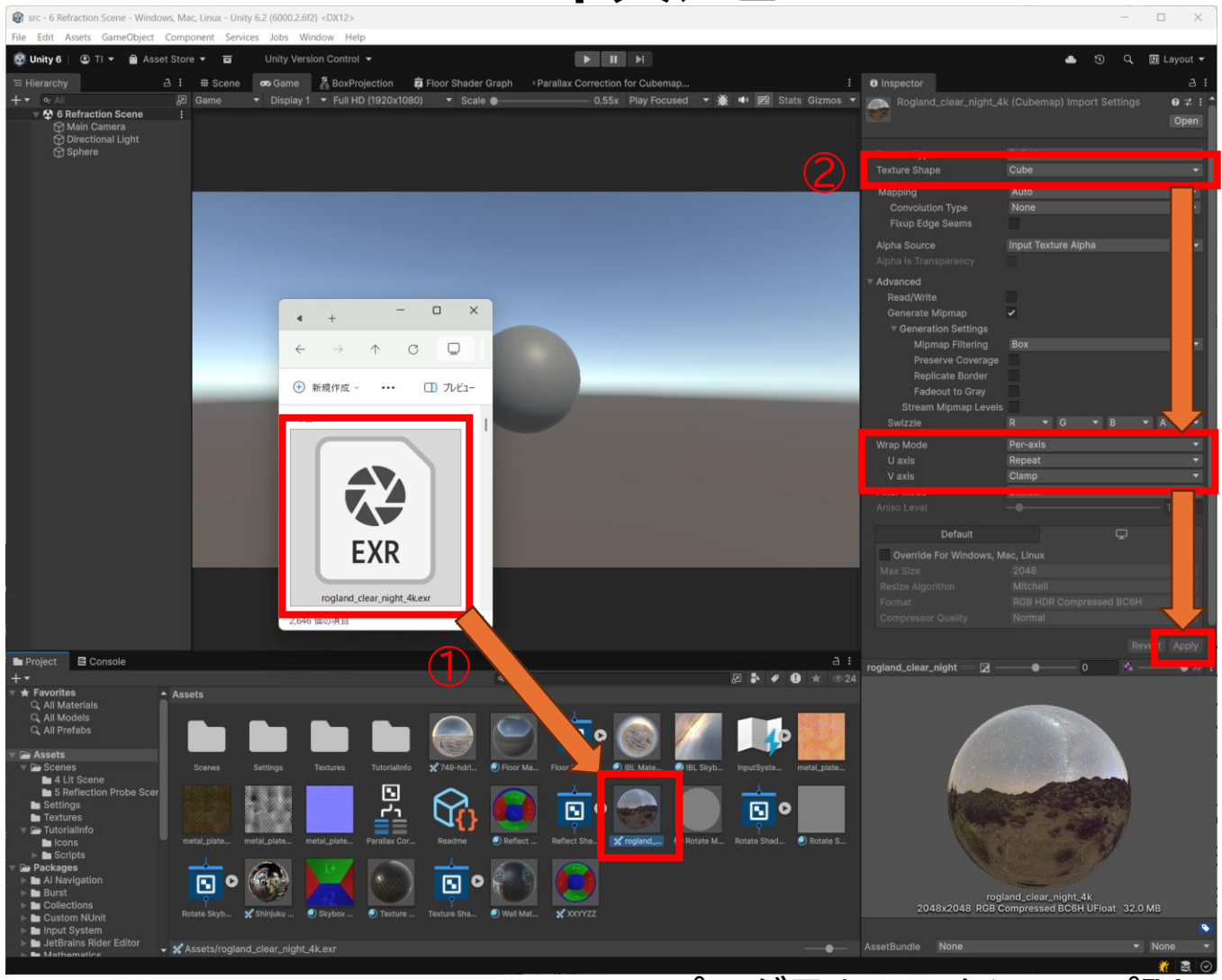
背景にキューブマップを設定

1. 適当に画像を拾ってD＆D
2. キューブマップに変換
3. マテリアルを生成
   - ここでは、名前を「IBL Skybox 2 Material」
   - Shaderを「Skybox/Cubemap」に設定
   - 「Cubemap(HDR)」に読み込んだ画像を設定
4. Lightingの「Environment」-「Skybox Material」にマテリアル(IBL Skybox 2 Material)を設定



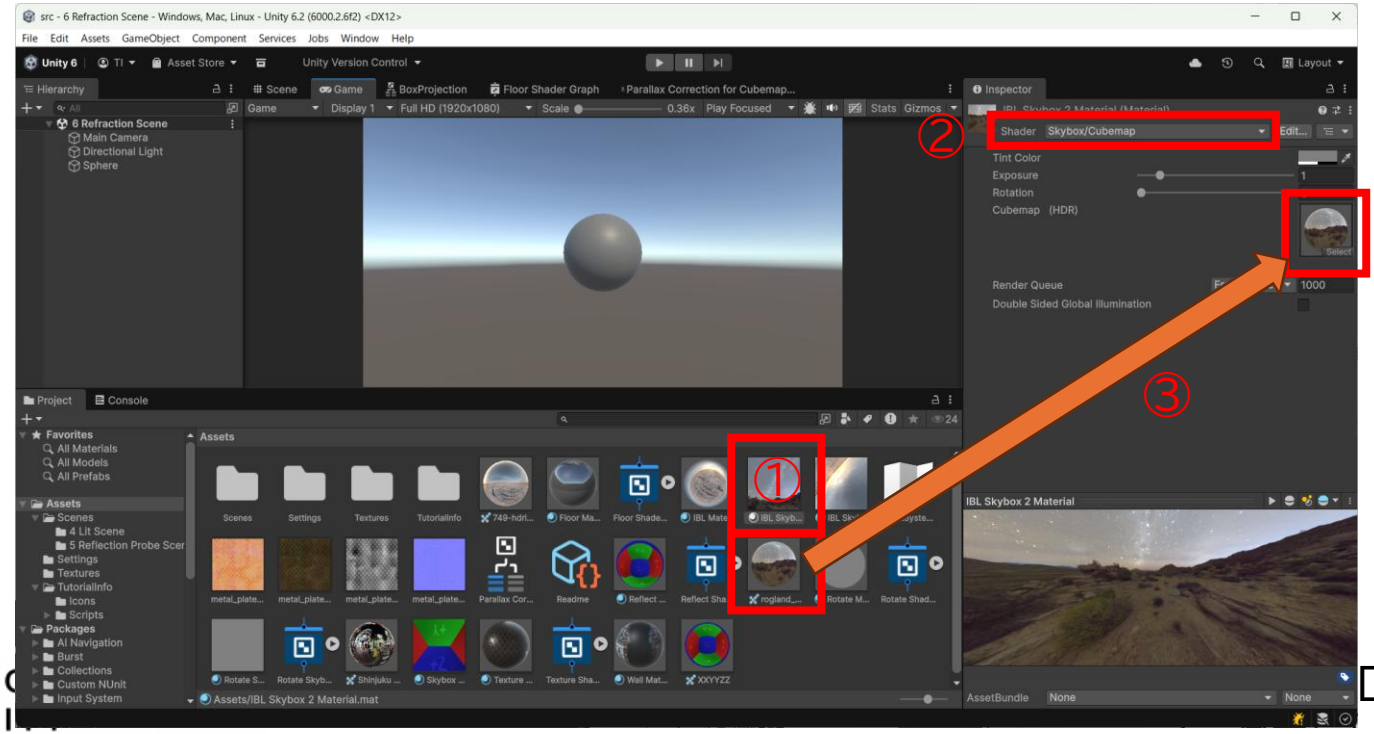
まずは、背景にキューブマップを設定 1-2

1. 適当に画像を拾ってD＆D
2. キューブマップに変換



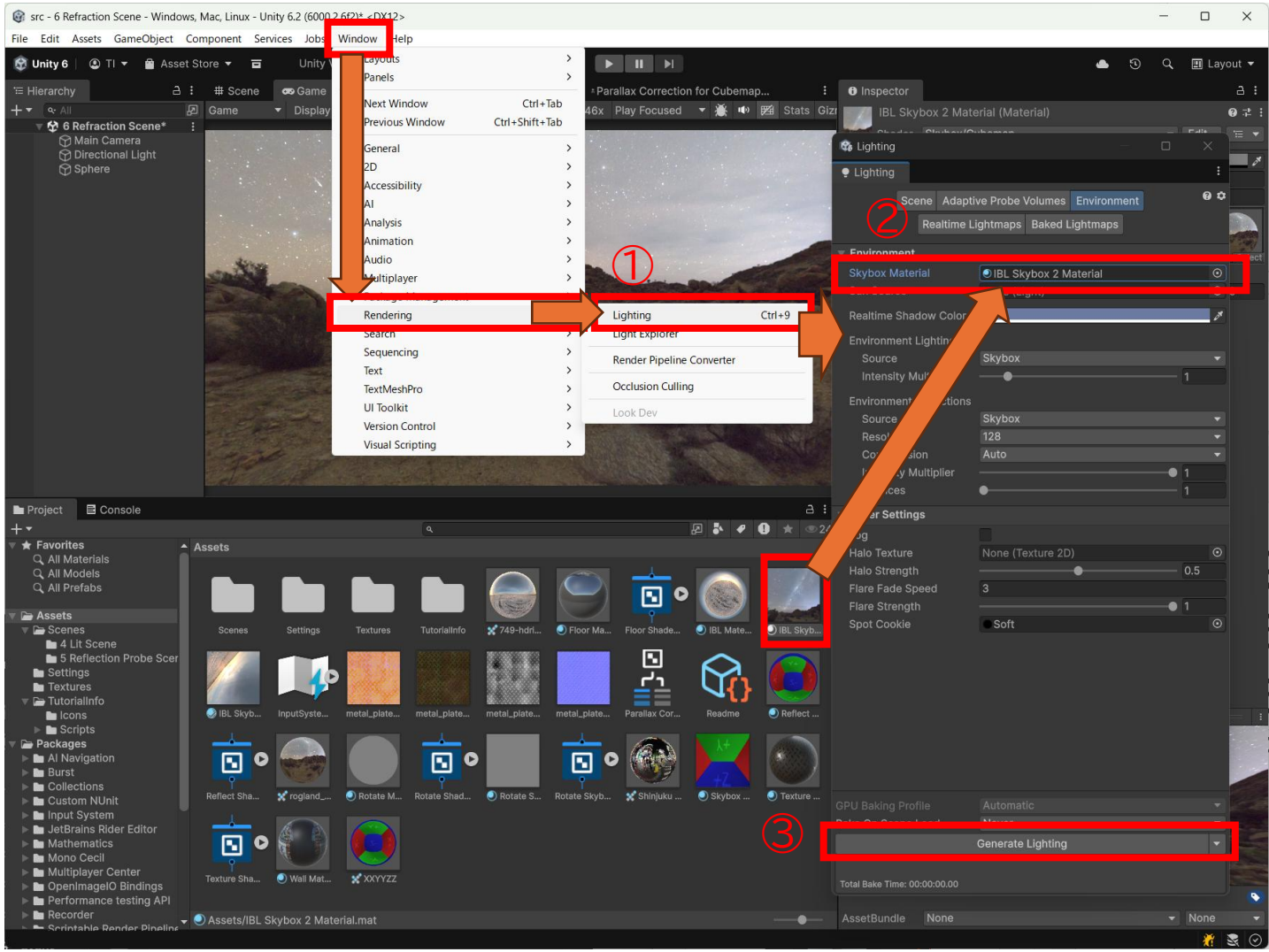
マテリアルを生成

- ここでは、名前を「IBL Skybox 2 Material」
- Shaderを「Skybox/Cubemap」に設定
- 「Cubemap(HDR)」に読み込んだ画像を設定



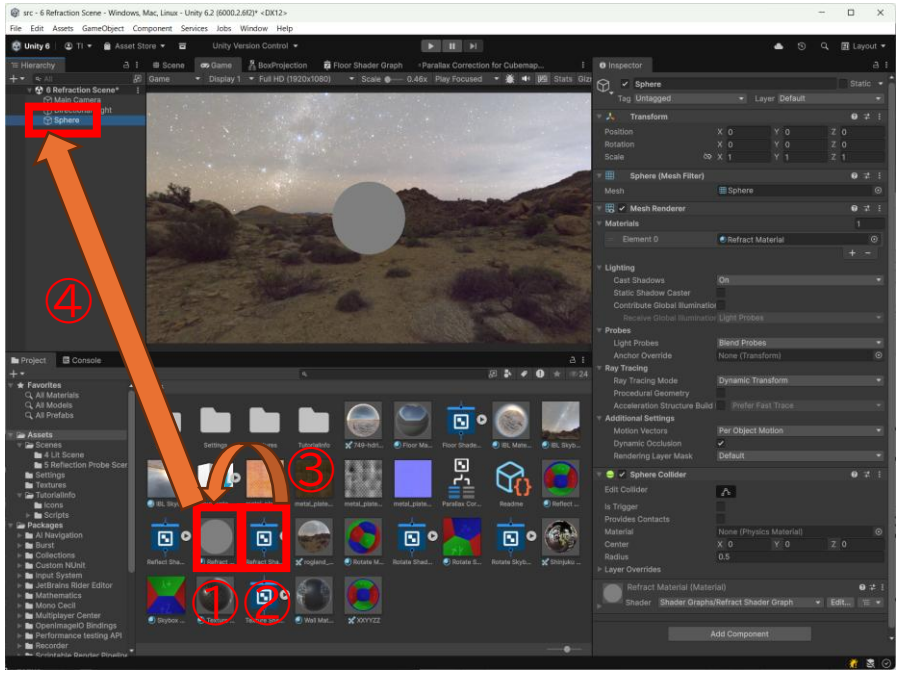
Lighting設定にマテリアルを設定

- メニューの「Window」-「Rendering」-「Lighting」
- 「Environment」タブの「Skybox Material」に「 IBL Skybox 2 Material」を設定
- 「Generate Lighting」を押す

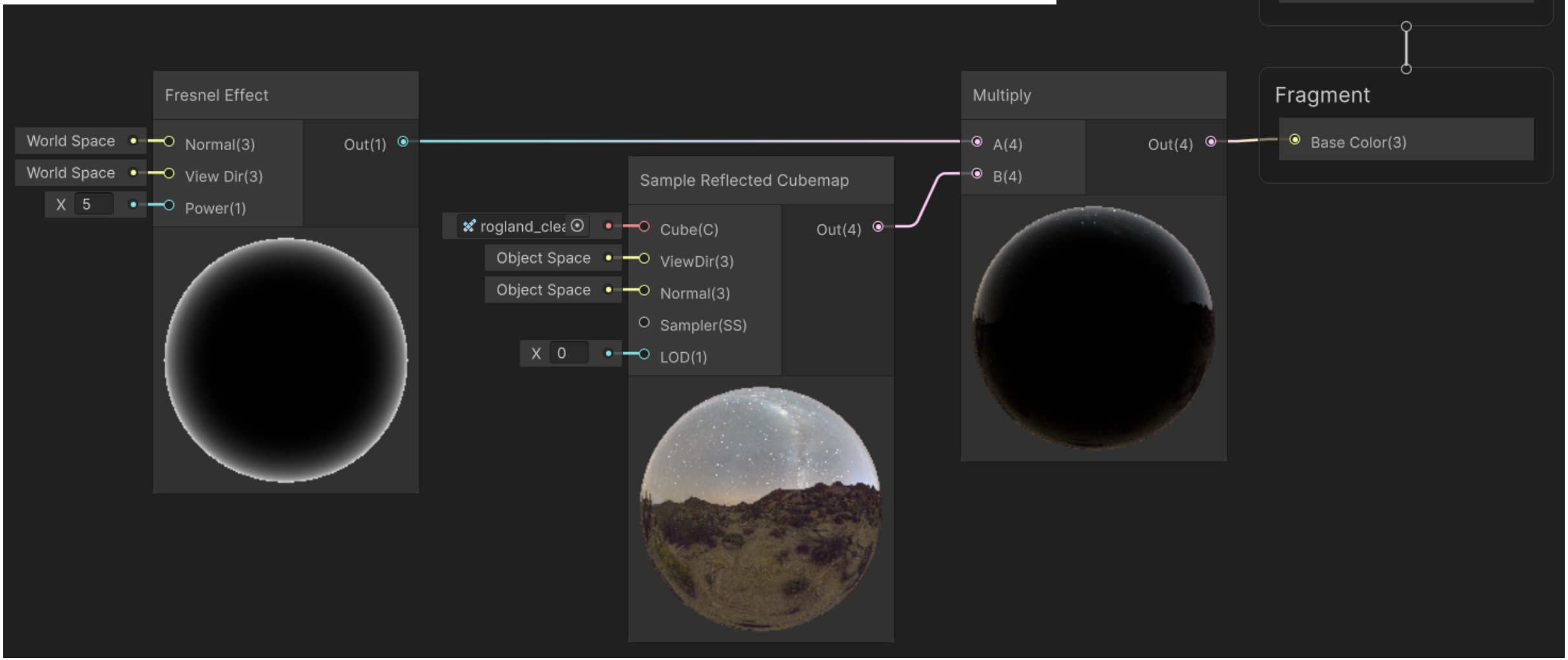


オブジェクトにシェーダーを設定する

1. マテリアルを作成
   - ここでの名前は「Refract material」
2. Unlit Shader Graphを作成
   - ここでの名前は「Refract Shader Graph」
3. マテリアルにシェーダグラフを設定
4. オブジェクトにマテリアルをバインド



フレネル効果付きの鏡面反射




中間ゴール：鏡面反射させよう
やってみよう

屈折

- 透明なガラス等では反射した以外の光は屈折して、物質内部に向かう
  - R(反射率) + T(透過率) = 1
    - T(透過率) = 1 − R(反射率)

$$= 1 - \left\{F_0 + (1 - F_0)\left(1 - \vec{V} \cdot \vec{H}\right)^5\right\}$$

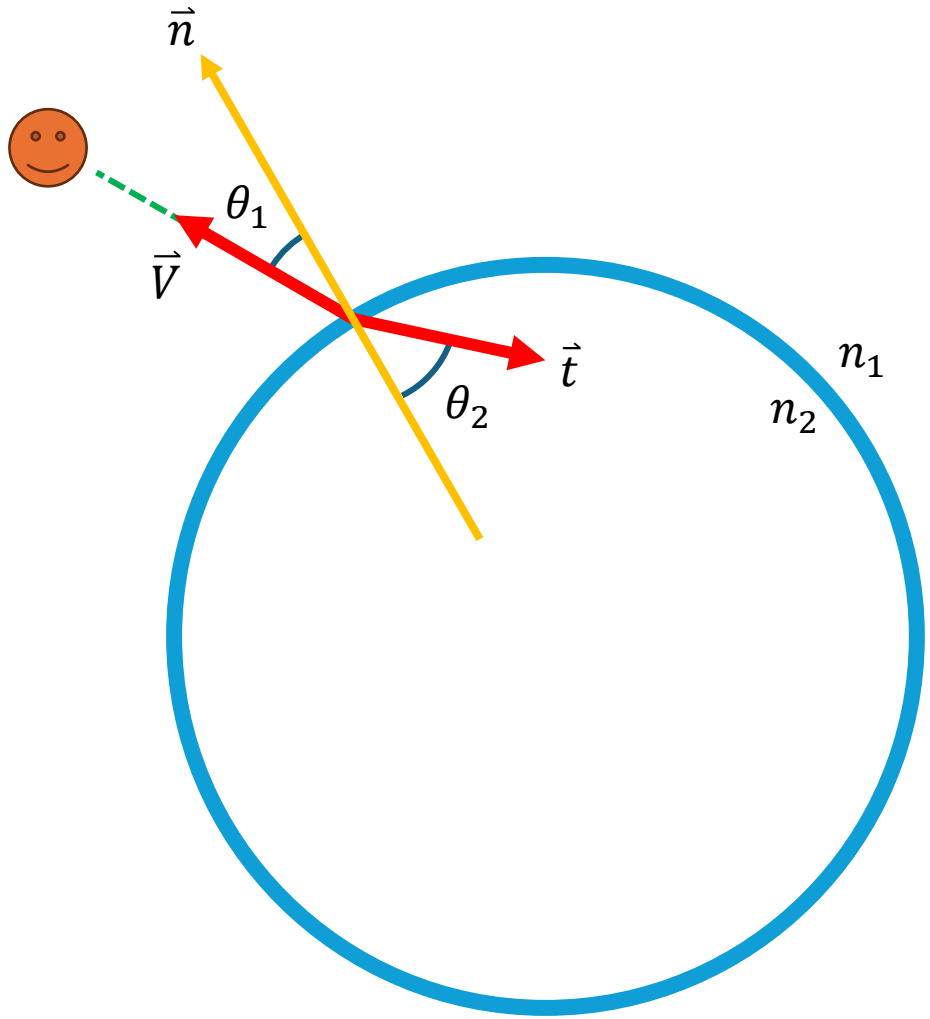
- 屈折する角度: スネルの法則

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$

$$\sin\theta_1 = n_{12} \sin\theta_2$$

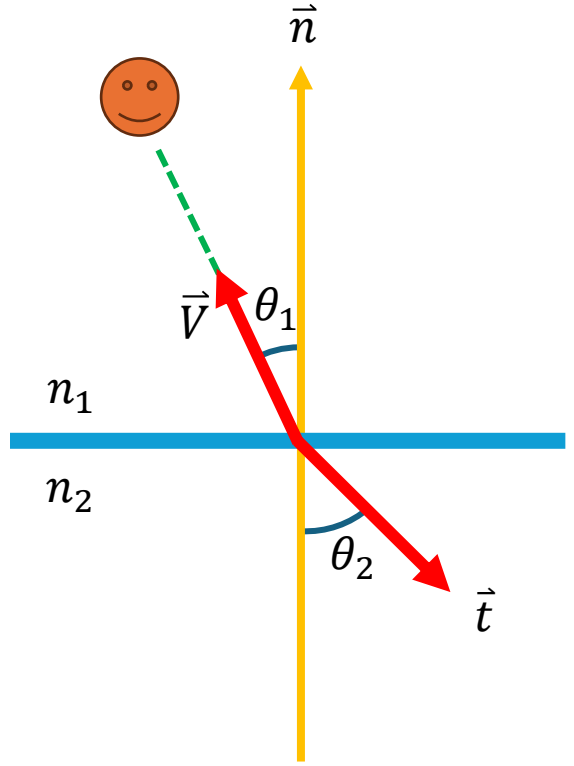
$$n_{12} = \frac{n_2}{n_1}$$：相対屈折率

空気に対するガラス：1.5程度



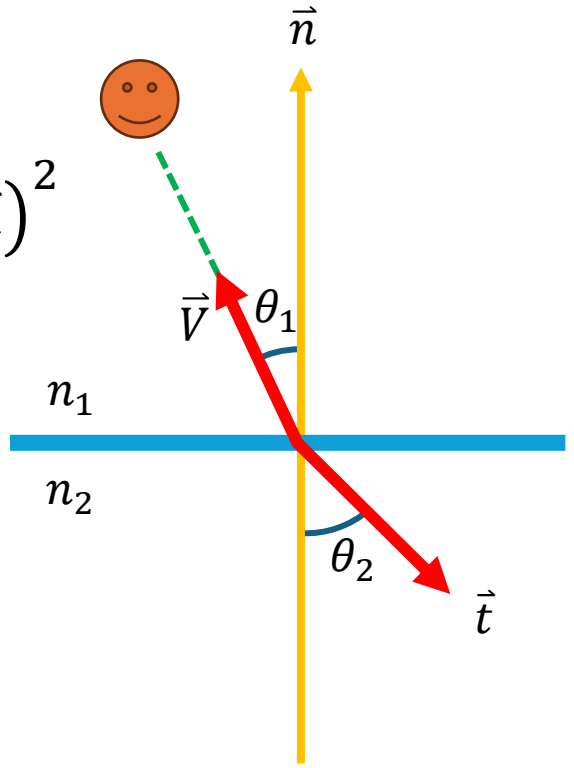
透過方向 $\vec{t}$

- スネルの法則
  - $\sin\theta_2 = \frac{1}{n_{12}}\sin\theta_1$
- 入射角度
  - $\cos\theta_1 = \vec{V}\cdot\vec{n}$
  - $\sin\theta_1 = \sqrt{1-\cos^2\theta_1} = \sqrt{1-\left(\vec{V}\cdot\vec{n}\right)^2}$
- 上の式から
  - $\sin\theta_2 = \frac{1}{n_{12}}\sin\theta_1 = \frac{1}{n_{12}}\sqrt{1-\left(\vec{V}\cdot\vec{n}\right)^2}$
  - $\cos\theta_2 = \sqrt{1-\sin^2\theta_2} = \frac{1}{n_{12}}\sqrt{n_{12}^2-\left\{1-\left(\vec{V}\cdot\vec{n}\right)^2\right\}}$
  - $\vec{t} = -\cos\theta_2\,\vec{n} - \sin\theta_2\,\frac{\vec{V}-(\vec{V}\cdot\vec{n})\vec{n}}{\|\vec{V}-(\vec{V}\cdot\vec{n})\vec{n}\|}$

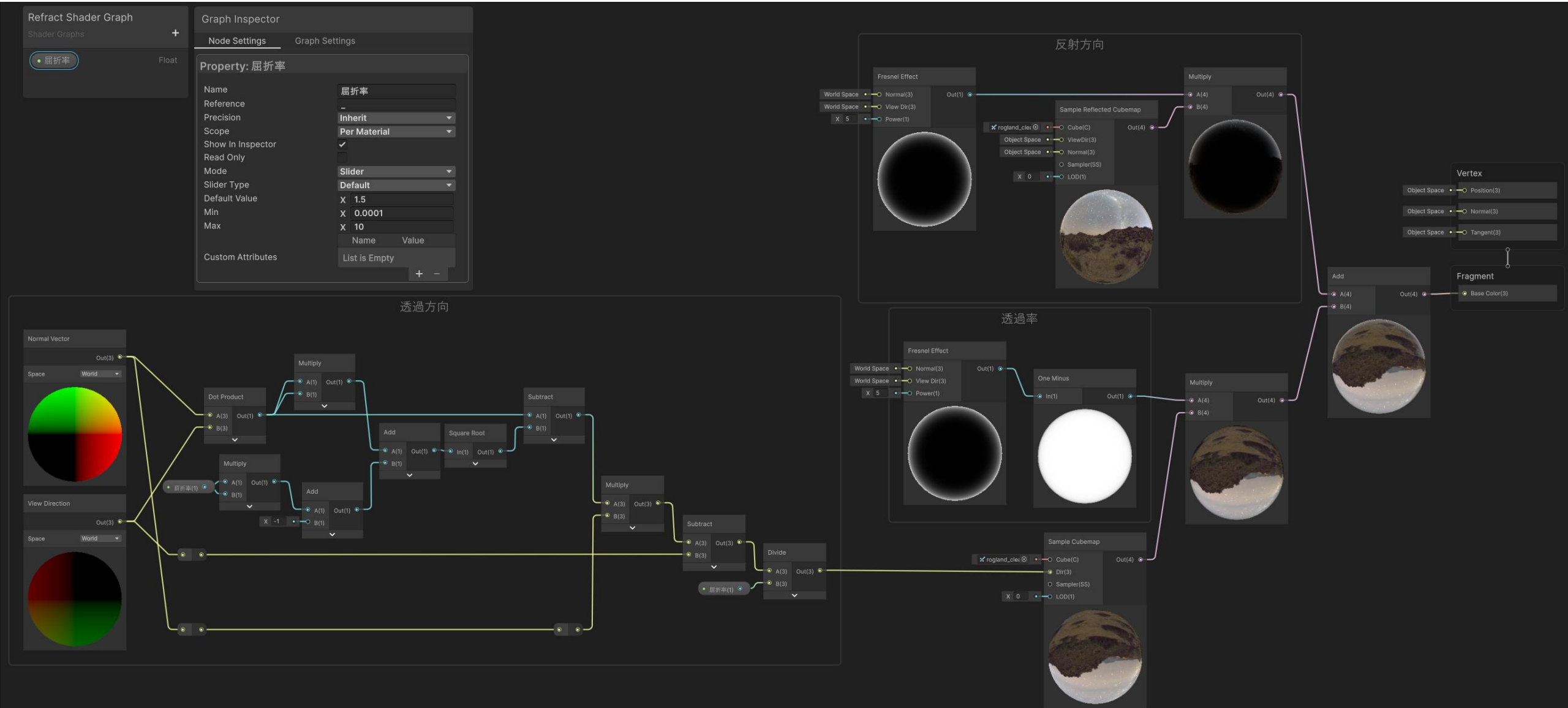


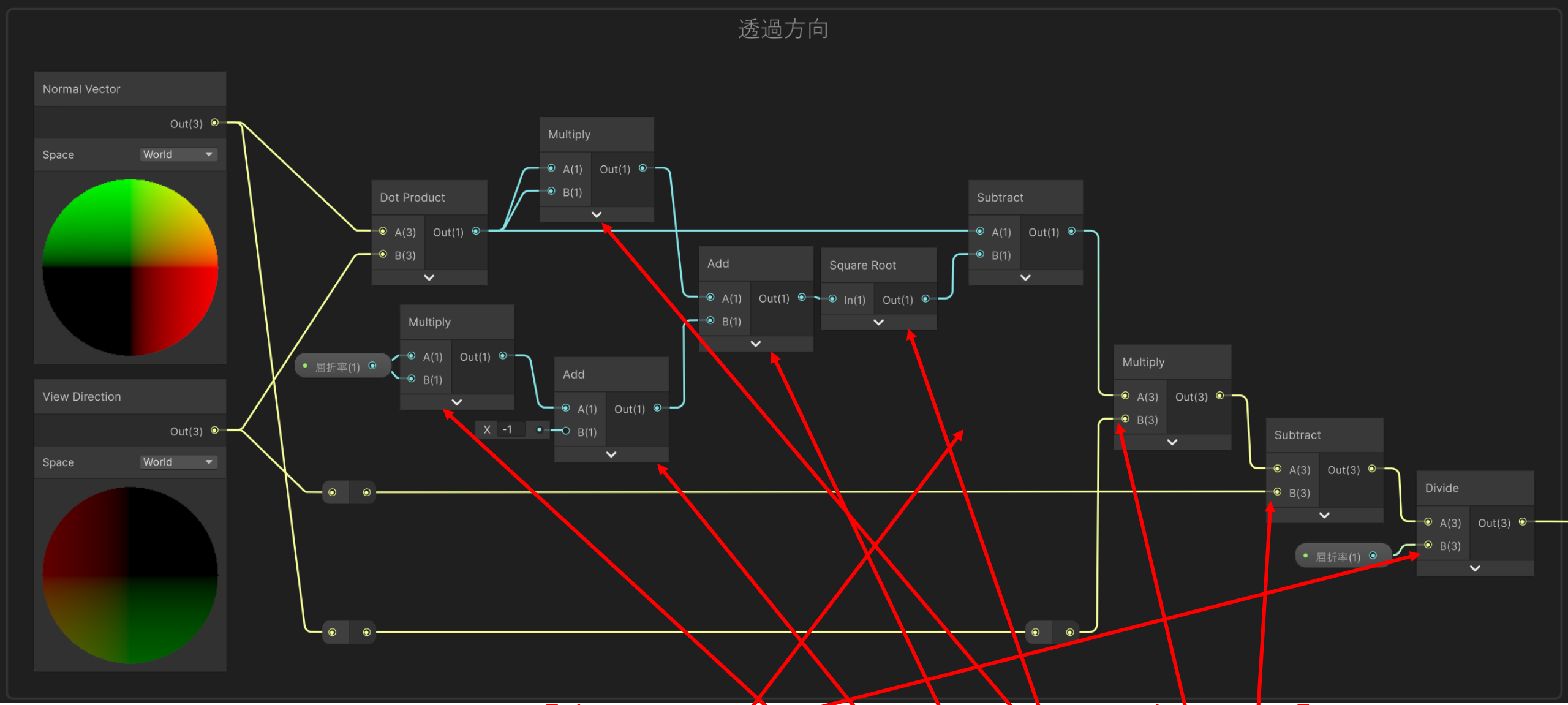
透過方向 $\vec{t}$

- $\left\|\vec{V}-(\vec{V}\cdot\vec{n})\vec{n}\right\|^2=(\vec{V}-(\vec{V}\cdot\vec{n})\vec{n})\cdot(\vec{V}-(\vec{V}\cdot\vec{n})\vec{n})$
  $=\vec{V}\cdot\vec{V}-2(\vec{V}\cdot\vec{n})(\vec{V}\cdot\vec{n})+(\vec{V}\cdot\vec{n})^2\vec{n}\cdot\vec{n}=1-(\vec{V}\cdot\vec{n})^2$
  $=\sin^2\theta_1$
- $\dfrac{\sin\theta_2}{\left\|\vec{V}-(\vec{V}\cdot\vec{n})\vec{n}\right\|}=\dfrac{\sin\theta_2}{\sin\theta_1}=\dfrac{1}{n_{12}}$ （∵スネルの法則より）
- $\vec{t}=-\cos\theta_2\,\vec{n}-\sin\theta_2\dfrac{\vec{V}-(\vec{V}\cdot\vec{n})\vec{n}}{\left\|\vec{V}-(\vec{V}\cdot\vec{n})\vec{n}\right\|}=-\cos\theta_2\,\vec{n}-\dfrac{\vec{V}-(\vec{V}\cdot\vec{n})\vec{n}}{n_{12}}$
  $=-\dfrac{1}{n_{12}}\sqrt{n_{12}^2-\left\{1-(\vec{V}\cdot\vec{n})^2\right\}}\,\vec{n}-\dfrac{\vec{V}-(\vec{V}\cdot\vec{n})\vec{n}}{n_{12}}$
  $=\dfrac{1}{n_{12}}\left[\left\{(\vec{V}\cdot\vec{n})-\sqrt{n_{12}^2-1+(\vec{V}\cdot\vec{n})^2}\right\}\vec{n}-\vec{V}\right]$



シェーダの全貌


Refract Shader Graph
Shader Graphs
屈折率
Float
Graph Inspector
Node Settings
Graph Settings
Property: 屈折率
Name 屈折率
Reference _
Precision Inherit
Scope Per Material
Show In Inspector
Read Only
Mode Slider
Slider Type Default
Default Value X 1.5
Min X 0.0001
Max X 10
Name Value
Custom Attributes
List is Empty
反射方向
Fresnel Effect
World Space
Normal(3)
View Dir(3)
Power(1)
Out(1)
Sample Reflected Cubemap
Cube(C)
ViewDir(3)
Normal(3)
Sampler(SS)
LOD(1)
Out(4)
Multiply
A(4)
B(4)
Vertex
Object Space
Position(3)
Normal(3)
Tangent(3)
Add
Fragment
Base Color(3)
透過方向
Normal Vector
Out(3)
Space
World
View Direction
Dot Product
A(3)
B(3)
Out(1)
Multiply
A(1)
B(1)
Add
Square Root
In(1)
Subtract
Divide
屈折率(1)
透過率
One Minus
In(1)
Out(1)
Sample Cubemap
Dir(3)

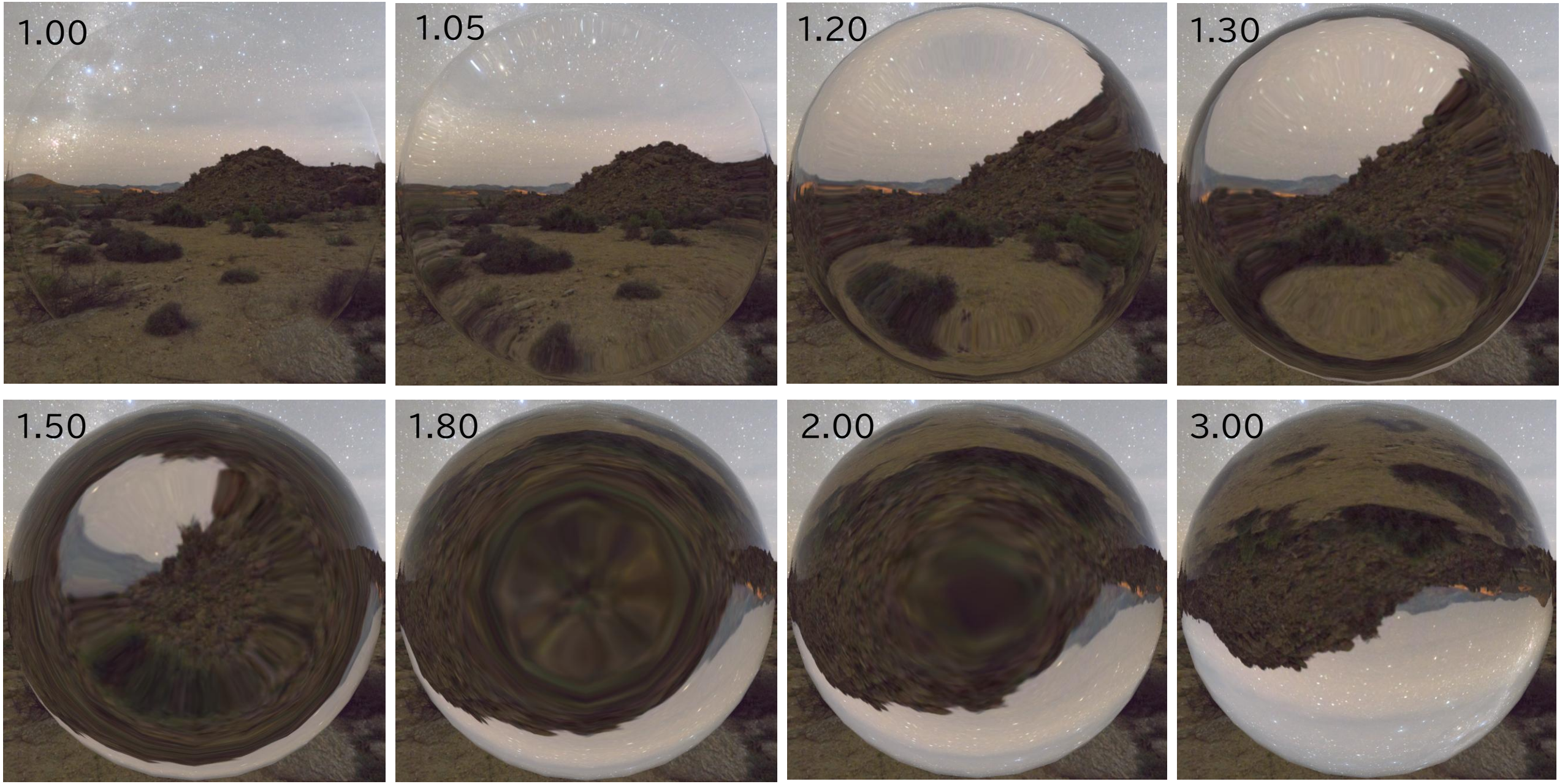


$$\vec{t} = \frac{1}{n_{12}}\left[\left\{(\vec{V}\cdot\vec{n}) - \sqrt{n_{12}^2 - 1 + (\vec{V}\cdot\vec{n})^2}\right\}\vec{n} - \vec{V}\right]$$


反射方向
Fresnel Effect
World Space
World Space
X 5
Normal(3)
View Dir(3)
Power(1)
Out(1)
Sample Reflected Cubemap
rogland_cle
Object Space
Object Space
X 0
Cube(C)
ViewDir(3)
Normal(3)
Sampler(SS)
LOD(1)
Out(4)
Multiply
A(4)
B(4)
Out(4)
反射(フレネル付き)
Vertex
Object Space
Object Space
Object Space
Position(3)
Normal(3)
Tangent(3)
Add
A(4)
B(4)
Out(4)
Fragment
Base Color(3)
透過率
Fresnel Effect
World Space
World Space
X 5
Normal(3)
View Dir(3)
Power(1)
Out(1)
屈折のフレネル項(1-F)
One Minus
In(1)
Out(1)
Multiply
A(4)
B(4)
Out(4)
Sample Cubemap
rogland_cle
X 0
Cube(C)
Dir(3)
Sampler(SS)
LOD(1)
Out(4)
屈折でのサンプリング

中間ゴール：屈折の効果を入れてみよう

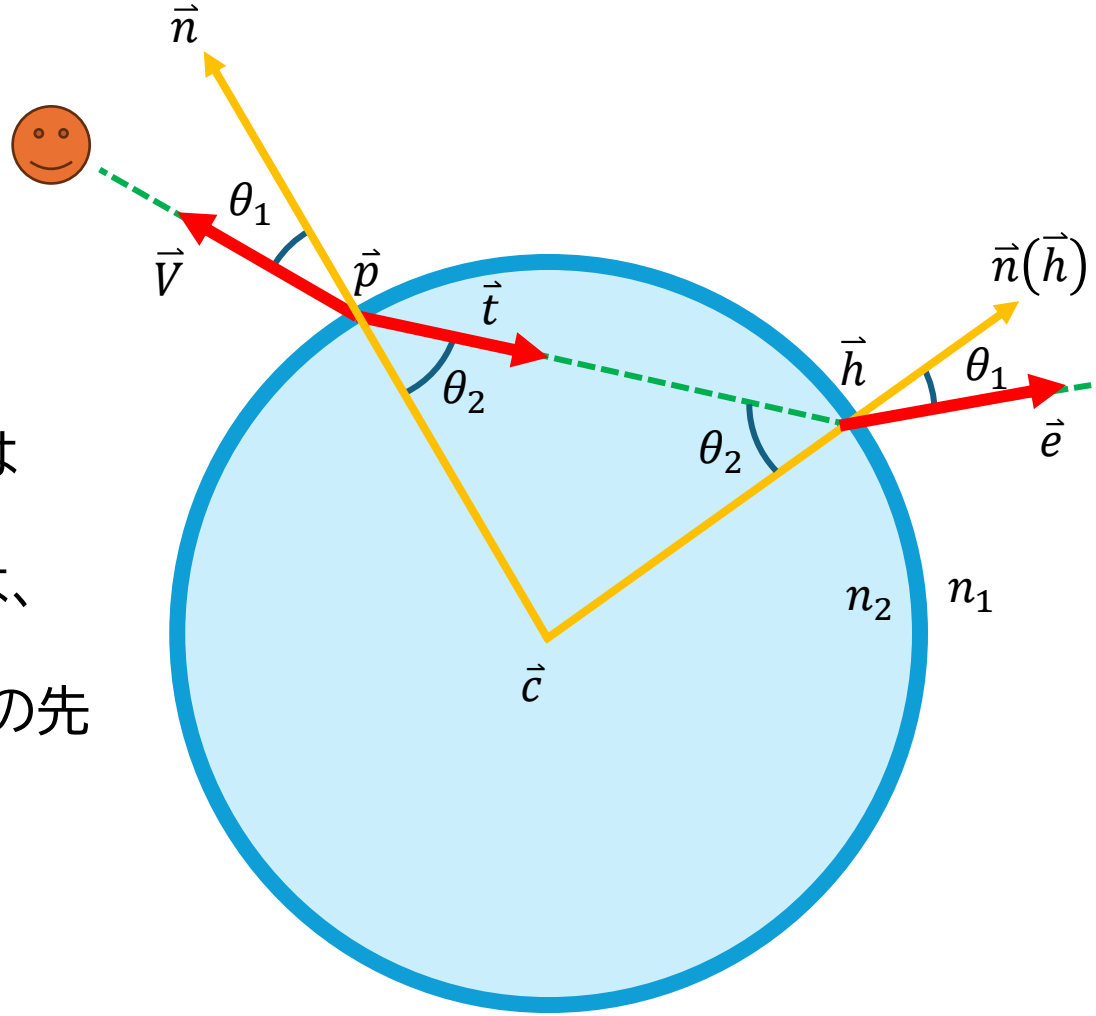
やってみよう



カメラ位置(z):-1.0

入った光は出ていく

- 一般に計算するのは困難
  - 球形状と仮定すれば計算できる
    - 形状の対称性により出ていく際の入射角度は入った際の透過時の角度と同じ
    - スネルの法則により出ていく際の透過角度は、入ってくる際の入射角度と同じ
    - 出ていく際の交点は球の方程式を満たし、$\vec{t}$の先
      - $\left(\vec{h}-\vec{c}\right)^2=r^2$
      - $\vec{h}=\vec{p}+d\vec{t}$

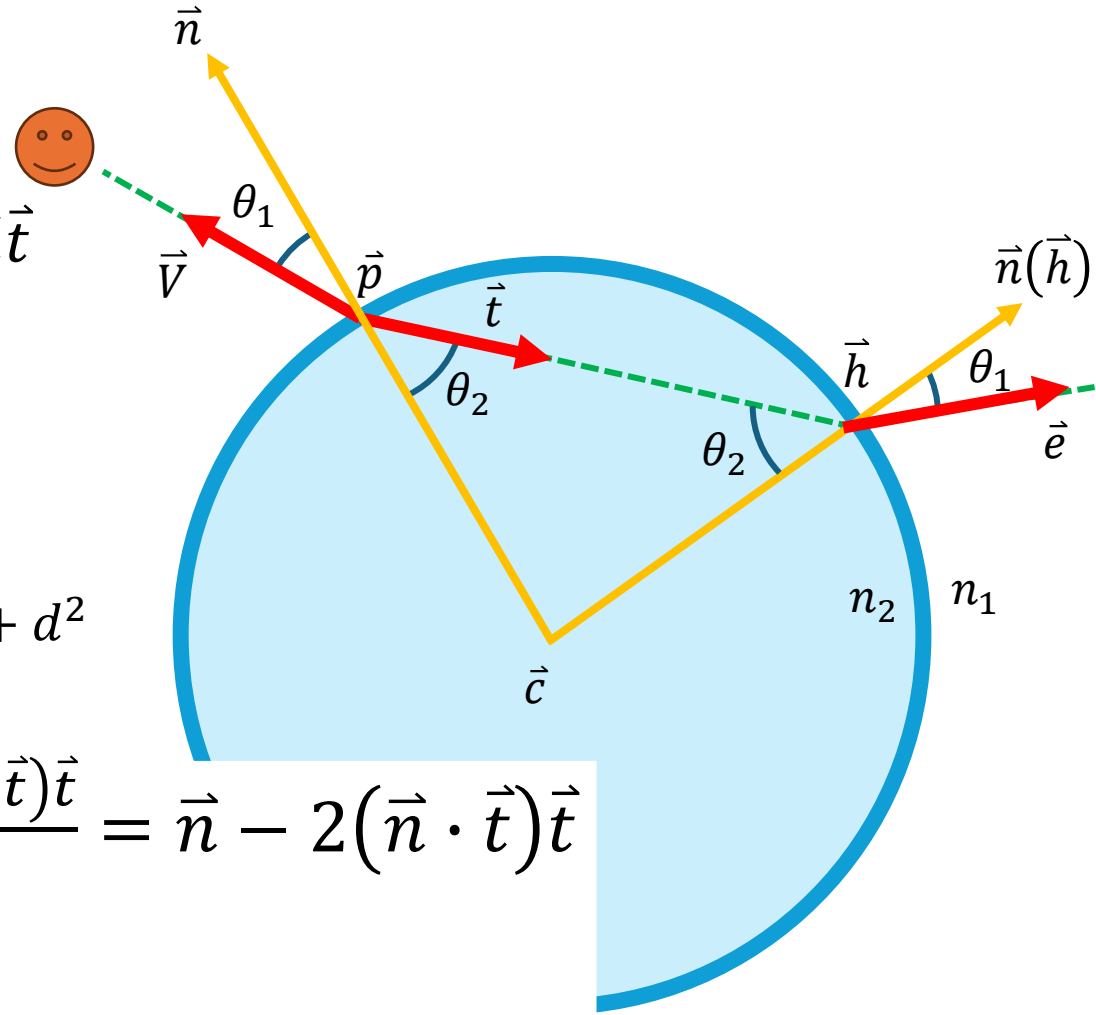


出ていく点での法線ベクトル

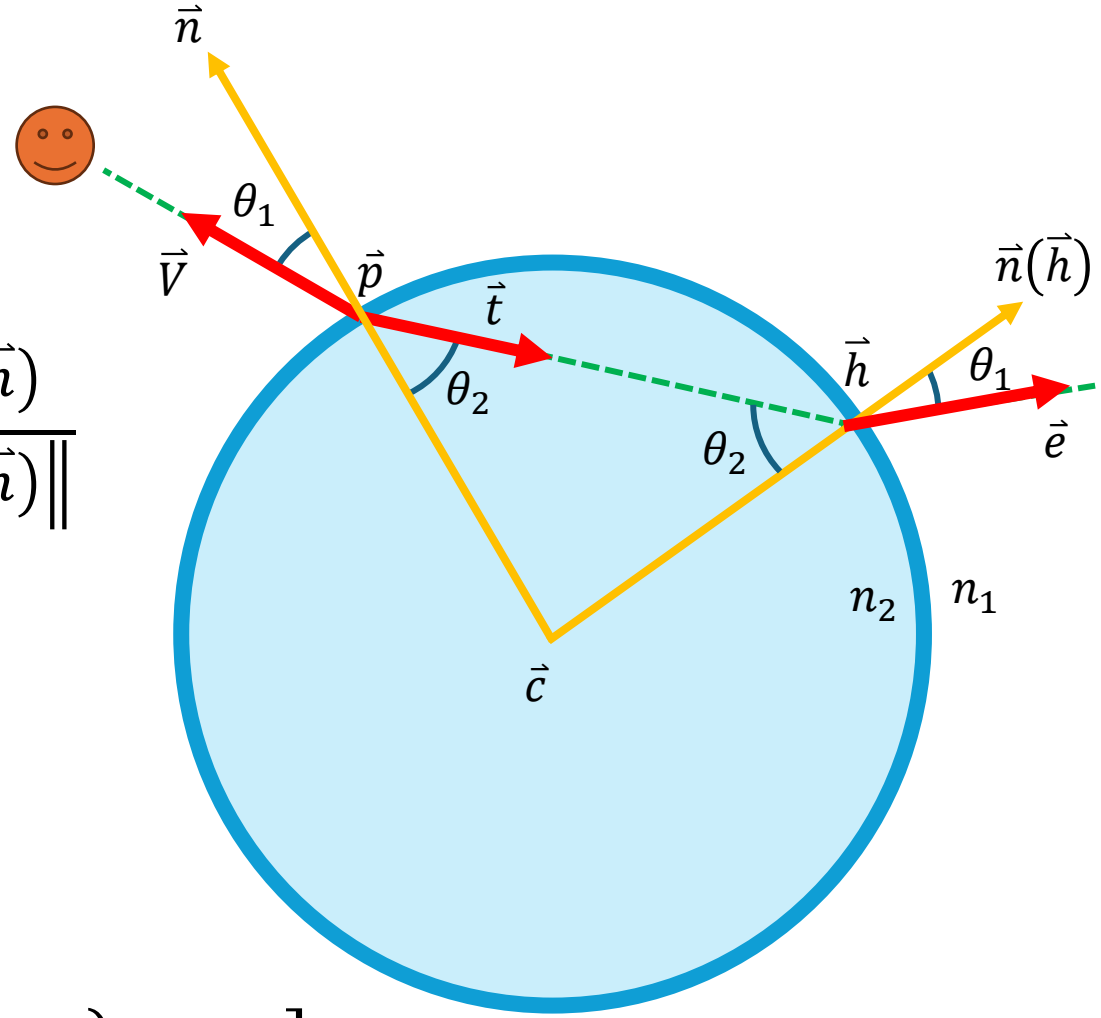
- ここで、
  - $\vec{h} = \vec{p} + d\vec{t} = (\vec{c} + r\vec{n}) + d\vec{t} = \vec{c} + r\vec{n} + d\vec{t}$
  - $\vec{h} - \vec{c} = r\vec{n} + d\vec{t}$
- 球の方程式から
  - $r^2 = \left(\vec{h} - \vec{c}\right)^2 = \left(r\vec{n} + d\vec{t}\right)^2$

    $= r^2\vec{n} \cdot \vec{n} + 2rd\vec{n} \cdot \vec{t} + d^2\vec{t} \cdot \vec{t} = r^2 + 2rd\vec{n} \cdot \vec{t} + d^2$

  $\Rightarrow \mathrm{d} = -2r\vec{n} \cdot \vec{t}$
- $\vec{n}(\vec{h}) = \dfrac{\vec{h}-\vec{c}}{\|\vec{h}-\vec{c}\|} = \dfrac{\vec{h}-\vec{c}}{r} = \dfrac{r\vec{n}+d\vec{t}}{r} = \dfrac{r\vec{n}-2r(\vec{n}\cdot\vec{t})\vec{t}}{r} = \vec{n} - 2(\vec{n} \cdot \vec{t})\vec{t}$



出ていく方向



- $\vec{e} = \cos\theta_1 \vec{n}(\vec{h}) + \sin\theta_1 \dfrac{\vec{t} - (\vec{t}\cdot\vec{n}(\vec{h}))\vec{n}(\vec{h})}{\left\|\vec{t} - (\vec{t}\cdot\vec{n}(\vec{h}))\vec{n}(\vec{h})\right\|}$

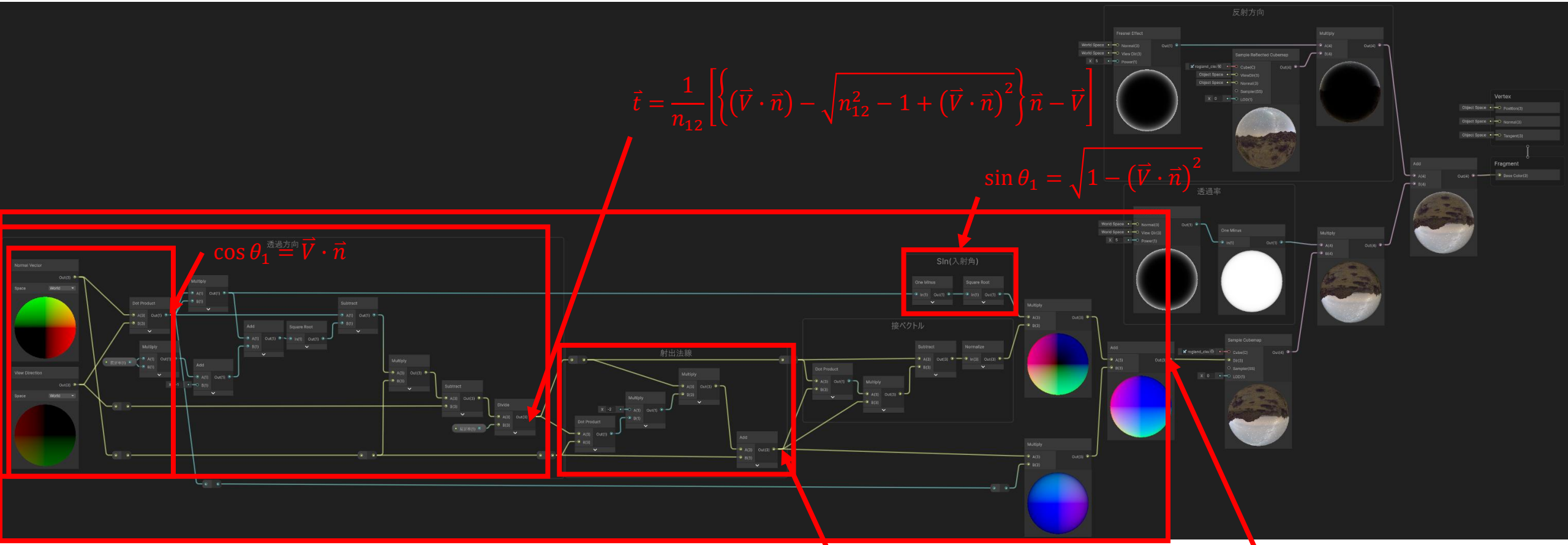
$$\vec{n}(\vec{h}) = \vec{n} - 2(\vec{n}\cdot\vec{t})\vec{t}$$

$$\cos\theta_1 = \vec{V}\cdot\vec{n}$$

$$\sin\theta_1 = \sqrt{1 - (\vec{V}\cdot\vec{n})^2}$$

$$\vec{t} = \frac{1}{n_{12}}\left[\left\{(\vec{V}\cdot\vec{n}) - \sqrt{n_{12}^2 - 1 + (\vec{V}\cdot\vec{n})^2}\right\}\vec{n} - \vec{V}\right]$$

Shader Graph



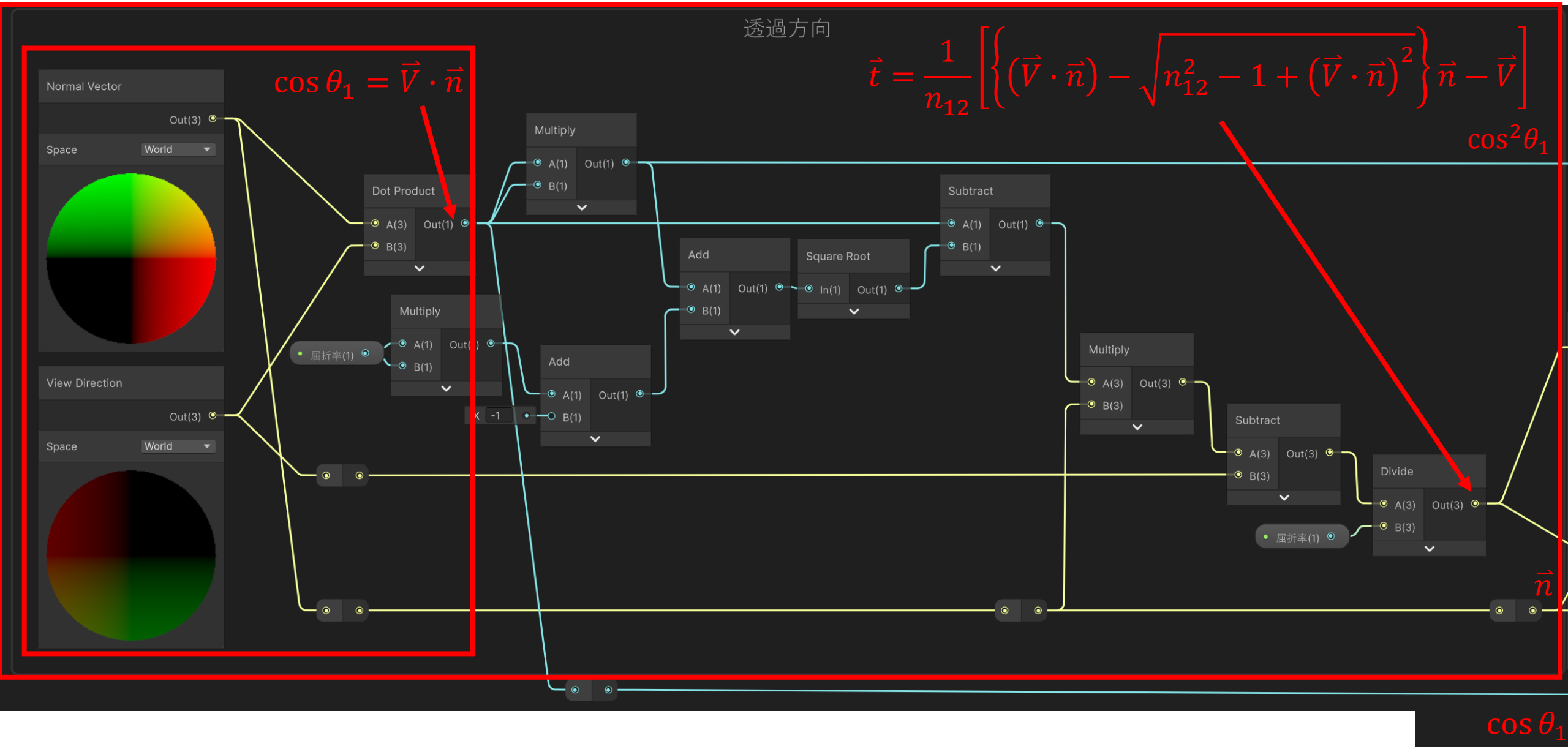
$$\vec{t} = \frac{1}{n_{12}}\left[\left\{(\vec{V}\cdot\vec{n}) - \sqrt{n_{12}^2 - 1 + (\vec{V}\cdot\vec{n})^2}\right\}\vec{n} - \vec{V}\right]$$

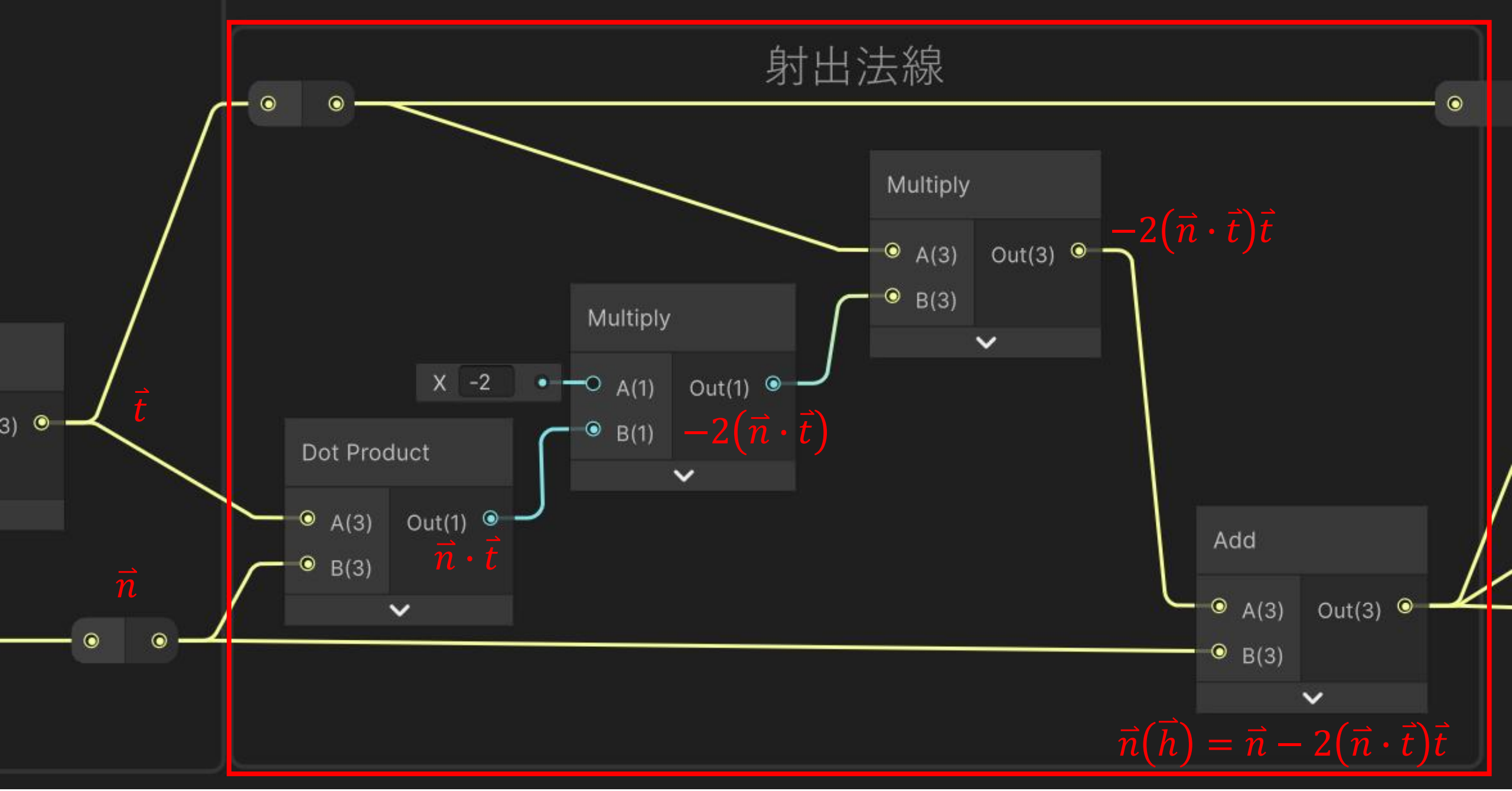
$$\sin\theta_1 = \sqrt{1 - (\vec{V}\cdot\vec{n})^2}$$

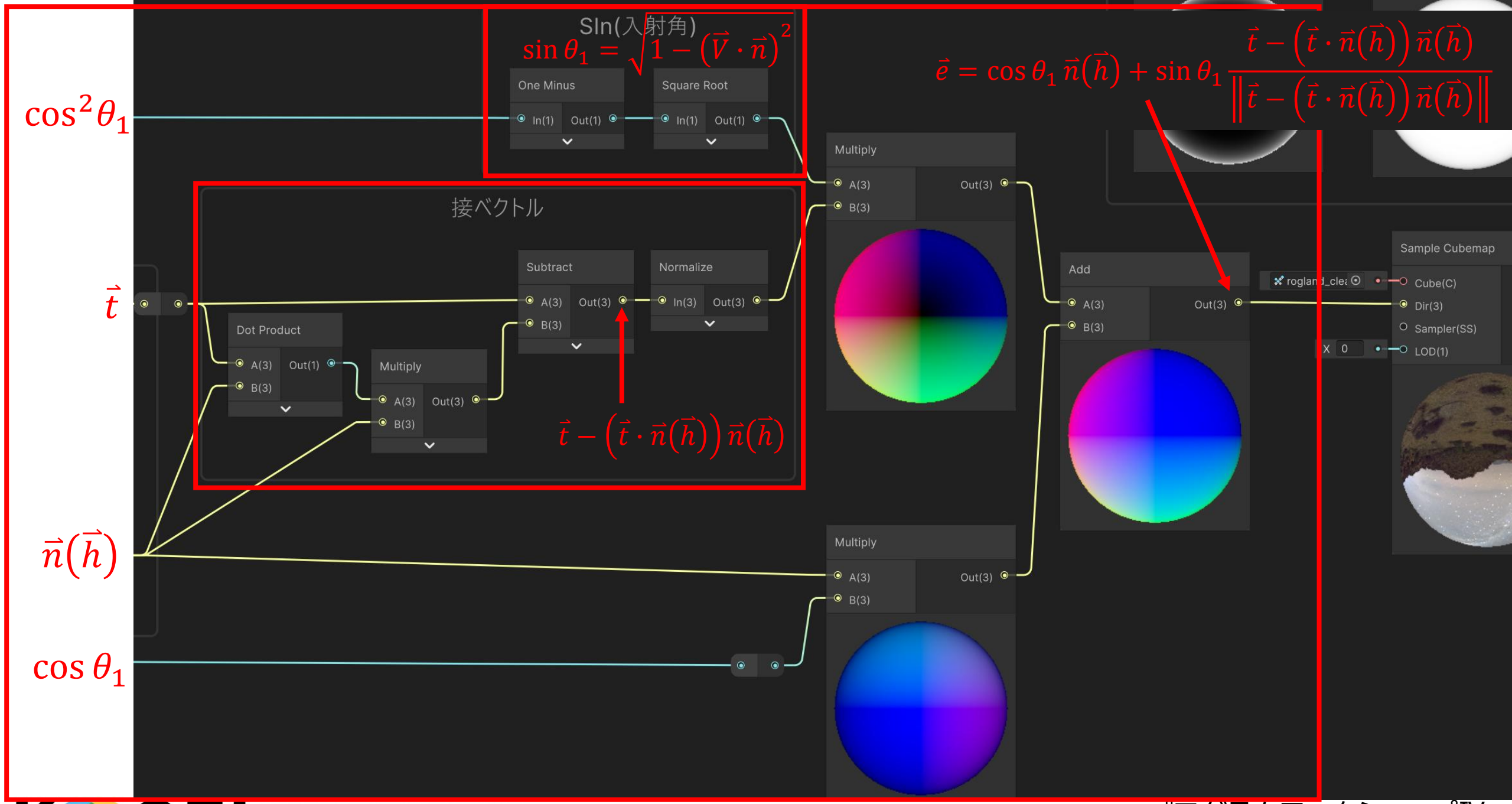
$$\cos\theta_1 = \vec{V}\cdot\vec{n}$$

$$\vec{n}(\vec{h}) = \vec{n} - 2(\vec{n}\cdot\vec{t})\vec{t}$$

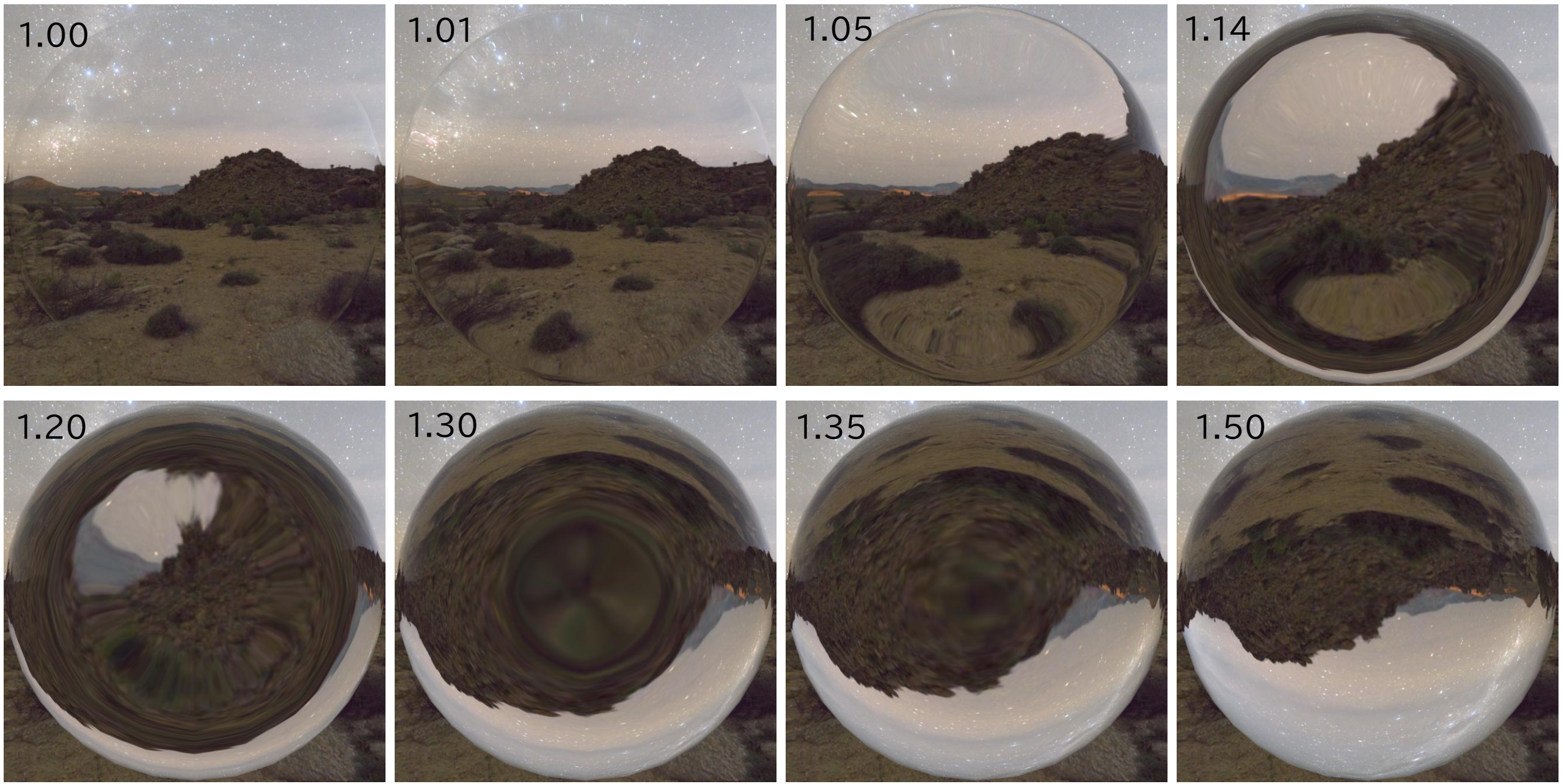
$$\vec{e} = \cos\theta_1\,\vec{n}(\vec{h}) + \sin\theta_1 \frac{\vec{t} - (\vec{t}\cdot\vec{n}(\vec{h}))\vec{n}(\vec{h})}{\left\|\vec{t} - (\vec{t}\cdot\vec{n}(\vec{h}))\vec{n}(\vec{h})\right\|}$$


透過方向
$\cos\theta_1 = \vec{V} \cdot \vec{n}$
$\vec{t} = \frac{1}{n_{12}}\left[\left\{(\vec{V}\cdot\vec{n}) - \sqrt{n_{12}^2 - 1 + (\vec{V}\cdot\vec{n})^2}\right\}\vec{n} - \vec{V}\right]$
$\cos^2\theta_1$
$\vec{n}$
$\cos\theta_1$
Normal Vector
Out(3)
Space
World
View Direction
Out(3)
Space
World
Dot Product
A(3)
B(3)
Out(1)
Multiply
A(1)
B(1)
Out(1)
Multiply
A(1)
B(1)
Out(1)
屈折率(1)
X
-1
Add
A(1)
B(1)
Out(1)
Add
A(1)
B(1)
Out(1)
Square Root
In(1)
Out(1)
Subtract
A(1)
B(1)
Out(1)
Multiply
A(3)
B(3)
Out(3)
Subtract
A(3)
B(3)
Out(3)
Divide
A(3)
B(3)
Out(3)
屈折率(1)


射出法線
Multiply
A(3)
B(3)
Out(3)
$-2(\vec{n}\cdot\vec{t})\vec{t}$
Multiply
X
-2
A(1)
B(1)
Out(1)
$-2(\vec{n}\cdot\vec{t})$
Dot Product
A(3)
B(3)
Out(1)
$\vec{n}\cdot\vec{t}$
$\vec{t}$
$\vec{n}$
Add
A(3)
B(3)
Out(3)
$\vec{n}(\vec{h}) = \vec{n} - 2(\vec{n}\cdot\vec{t})\vec{t}$

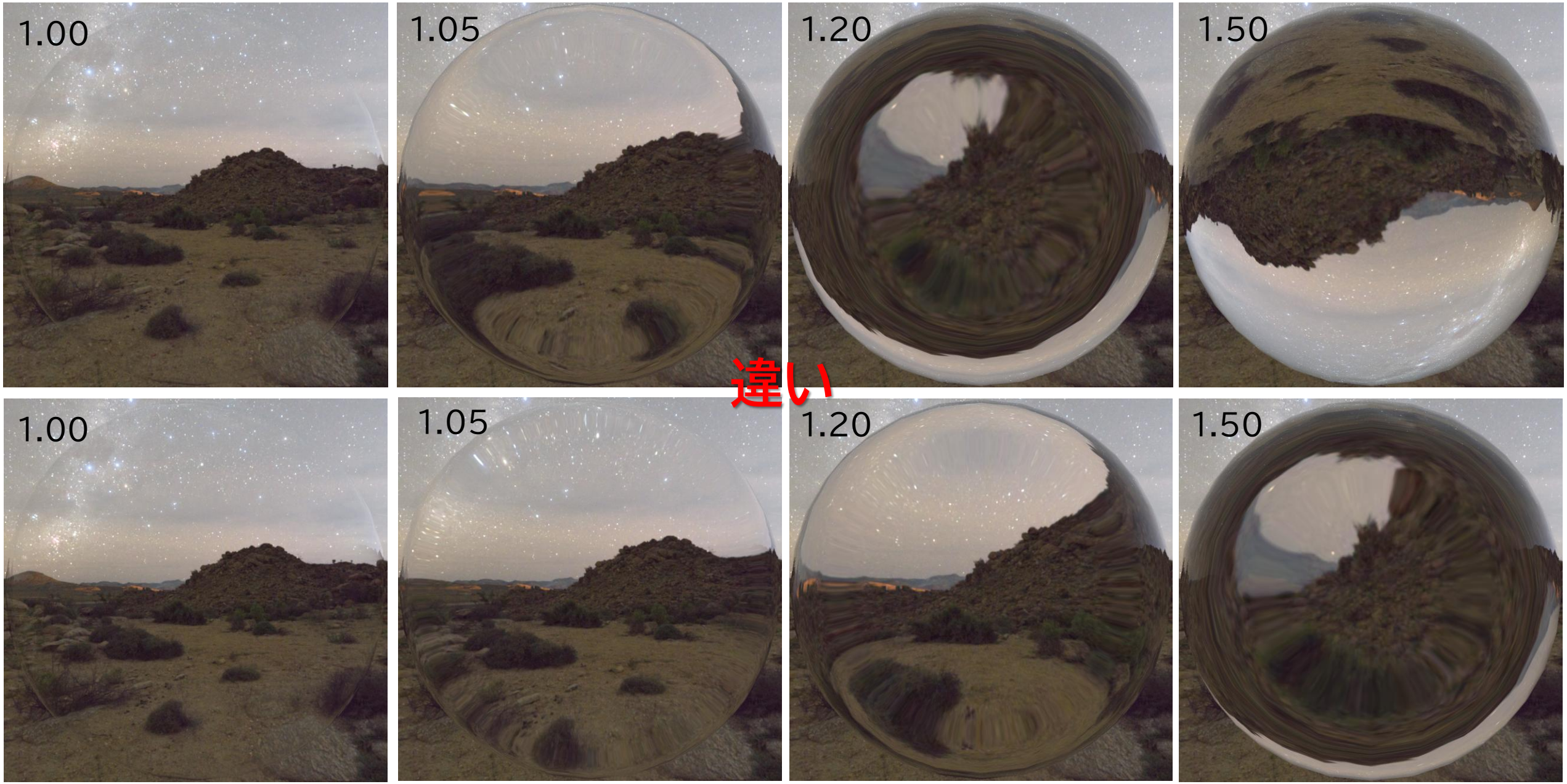

$\cos^2\theta_1$
SIn(入射角)
$\sin\theta_1 = \sqrt{1-(\vec{V}\cdot\vec{n})^2}$
One Minus
In(1)
Out(1)
Square Root
In(1)
Out(1)
$\vec{t}$
接ベクトル
Subtract
A(3)
B(3)
Out(3)
Normalize
In(3)
Out(3)
Dot Product
A(3)
B(3)
Out(1)
Multiply
A(3)
B(3)
Out(3)
$\vec{t}-(\vec{t}\cdot\vec{n}(\vec{h}))\vec{n}(\vec{h})$
$\vec{n}(\vec{h})$
$\cos\theta_1$
Multiply
A(3)
B(3)
Out(3)
Multiply
A(3)
B(3)
Out(3)
$\vec{e} = \cos\theta_1\,\vec{n}(\vec{h}) + \sin\theta_1 \frac{\vec{t}-(\vec{t}\cdot\vec{n}(\vec{h}))\vec{n}(\vec{h})}{\|\vec{t}-(\vec{t}\cdot\vec{n}(\vec{h}))\vec{n}(\vec{h})\|}$
Add
A(3)
B(3)
Out(3)
Sample Cubemap
Cube(C)
Dir(3)
Sampler(SS)
LOD(1)
X 0


球の屈折の効果を入れてみよう
やってみよう



カメラ位置(z):-1.0

球として処理



表面だけ処理

まとめ

- 概説
  - レンダリング技法
  - Unityでキューブマップを使う
  - キューブマップテクスチャの入手
- 今回の実習
  - キューブマップを使う
    - オブジェクトに反射キューブマップを貼る
  - キューブマップを動かす
    - キューブマップを背景と合わせて回転させる
  - 自分のキューブマップに差し替えてみよう
    - 外部サイトからテクスチャを持ってくる
  - PBRでのキューブマップ
    - Litシェーダの環境マップとしてキューブマップを使う
  - 反射プローブ
    - 視差補正キューブマップを行う
    - 視差補正キューブマップを投影表現に用いる
    - CPUからパラメータを送る
  - 屈折
    - 透明な球の屈折表現を行う