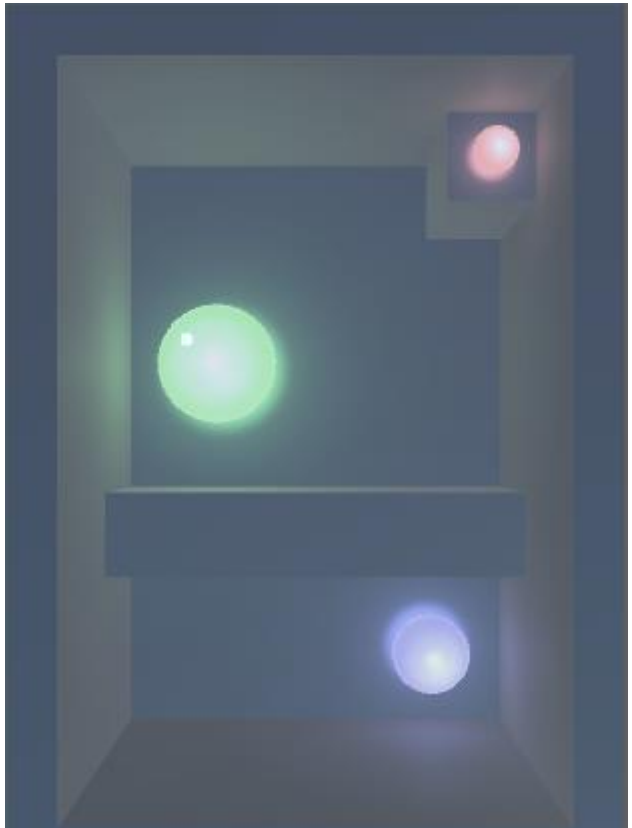
# CHAPTER3_ゲーム作成の基本
# ー物理エンジンとコリジョンイベントをマスターする

衝突（コリジョン）判定等と物理エンジンを使ったゲームを作成する。

**Chapter3 の目的**

- オブジェクトと基本コンポーネントを扱えるようになる
- 物理エンジンを扱えるようになる
- 衝突判定によるイベント制御ができるようになる

図3-1　作成するゲーム（Illumiball）



# 3-1　シーンの初期化

シーンはF2でリネーム可能。プロジェクト作成初期状態は**SampleScene**となっているので、任意の名前にリネームする。

# 3-2　物理挙動を持つBallオブジェクトの作成

## Ballオブジェクトの作成

（特記事項なし）

## ゲームオブジェクトとコンポーネント

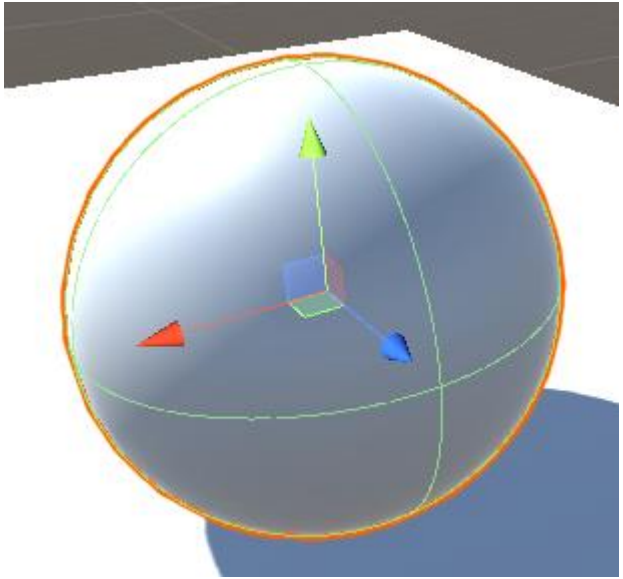
**ゲームオブジェクト**は複数の**コンポーネント**で構成され、必ず**Transform**コンポーネントが含まれる。ゲームオブジェクトは自由にコンポーネントを追加することができる。

ゲームオブジェクト - Unity マニュアル

コンポーネントへの入門 - Unity マニュアル

スフィアはゲームオブジェクト。メインカメラもゲームオブジェクト。ディレクショナルライトもゲームオブジェクト。

図3-2　スフィア



インスペクター

BallGreen　静的

タグ Untagged　レイヤー Default

Transform

| | | | | | |
|---|---|---|---|---|---|
| 位置 | X | -5.61 | Y | 1.84 | Z 0 |
| 回転 | X | 0 | Y | 0 | Z 0 |
| 拡大/縮小 | X | 3 | Y | 3 | Z 3 |

Sphere (Mesh Filter)

メッシュ　Sphere

Mesh Renderer

マテリアル

サイズ　1

要素0　Default-Material

ライティング

投影　オン

影を受ける　✓

グローバルイルミネーション

グローバルイルミネーション　ライトプローブ

## Rigidbodyコンポーネントの追加

オブジェクトに物理挙動をさせるには**Rigidbody**コンポーネントを追加する。

### TIPS　InspectorビューのAdd Componentボタン

**コンポーネントを追加**ボタンでインスペクタービュー上からオブジェクトにコンポーネントを追加することができる。

### COLUMN　コライダー範囲と描画モード変更による確認

ゲームオブジェクト同士が衝突するのはゲームオブジェクトに**Collider（コライダー）**コンポーネントが設定されているからである。どちらかのコライダーを無効化すると、ゲームオブジェクトは衝突せずにすり抜ける。

**TIPS　コライダーの範囲**

コライダーオブジェクトが設定されているオブジェクトはスケール（拡大/縮小）を変化させると自動で合わせて変化する。コライダーはオブジェクトの大きさと異なる範囲に調整することも可能。

## ライティングの設定

ディレクショナルライトはインスペクター上の**強度**で光の強さを、**シャドウタイプ**で影の有無を設定できる。

## マテリアルの発光設定

マテリアルは**放出**の項目にチェックを入れることで発光させることが出来る。色の指定は**HDR（High Dynamic Range）**で可能。

**放出はスタティックなオブジェクトに対してのみ有効**（つまり、動くオブジェクトは放出できない）。

**TIPS　マテリアルの反映**

マテリアルはシーンビューのオブジェクトにドラッグ＆ドロップしても設定可能。

## Point Lightの生成と親子関係の設定

ある1点から光を発するには**Point Light（ポイントライト）**を使う。

ヒエラルキーで右クリック　＞　ライト　＞　ポイントライト

オブジェクトはヒエラルキー上で他オブジェクトに対してドラッグ＆ドロップすると、ドロップしたオブジェクトと親子関係を設定することができる（ドロップされたオブジェクトが親になる）。親子関係にあるオブジェクトは親が動くと子も同時に動くようになる。

インスペクター上のTransformコンポーネントの項目の**位置**は親子関係によって扱いが変わる。親の場合はシーンの原点からの絶対座標である**Global Position（グローバルポジション）**、子の場合は親の座標を起点とする**Local Position（ローカルポジション）**となる。

## Point Lightパラメータの調整

ポイントライトの照明の範囲はLightコンポーネントの項目の**範囲**で設定可能。照明の色は**色**で設定可能。重要度は**レンダーモード**で設定可能。ライティングは高負荷な処理なので、重要でないライトに関してはレンダーモードを**重要ではない**に設定すると良い。

# 3-3　ステージの作成

## ステージTransformの初期化

　コンポーネントをまとめるために**空のゲームオブジェクト**を使う。空のオブジェクトはtransformコンポーネントの位置が原点ではない可能性があるので、原点に合わせておく。こうすることで子オブジェクトの位置を絶対座標の様に扱うことができる。

## 周囲の壁の作成

　オブジェクトは**Alt + D**で複製することができる。

## 見えない天井とコライダーの作成

オブジェクトがオブジェクトを通り抜けないようにするには**コライダー**コンポーネントを使う。詳細は後程説明する。

## 障害物の設置

（特記事項なし）

## カメラの調整

（特記事項なし）

### TIPS　Inspectorビューのドラッグによる値の調整

インスペクタービューのX,Y,Z等のプロパティ名は左右にドラッグすることで値を変更することができる。

### COLUMN　カメラとマルチアスペクト比

端末によってアスペクト比は異なるため、ターゲットにしている端末に合うように調整する必要がある。通常はターゲットを2～5機種ほどに搾る場合が多いが、プロジェクトの方針によって変わる。

# 3-4　スクリプトによる重力の操作

## 物理エンジンの事前設定と確認

Unityにしばらく動かないと判断されたリジッドボディコンポーネントを持つオブジェクトは、外部から力を加えられない限り物理挙動を止める。これを**スリープ**と呼ぶ。

スリープになる値の設定は以下の手順で可能。

メニューの編集　＞　プロジェクト設定　＞　物理　＞　Sleep Threshold

重力の設定は以下の手順で可能。

メニューの編集　＞　プロジェクト設定　＞　物理　＞　重力

## キーボード入力による操作

キーボードの入力取得は**Input**クラスを使い、重力の操作は**Physics**クラスを使う。

重力等の物理的な挙動に対する方向は**Vector3構造体**で表す。

UnityEngine.Input - Unity スクリプトリファレンス

UnityEngine.Physics - Unity スクリプトリファレンス

UnityEngine.Vector3 - Unity スクリプトリファレンス

水平方向の入力値を取得（←キーと→キー）

```
Input.GetAxis("Horizontal")
```

垂直方向の入力値を取得（↑キーと↓キー）

```
Input.GetAxis("Vertical")
```

キーが押されているか判断（戻り値bool型）

```
Input.GetKey("z")
```

Vector3のX座標の値を設定

※Vector3のインスタンスvector

```
vector.x = 10.0f
```

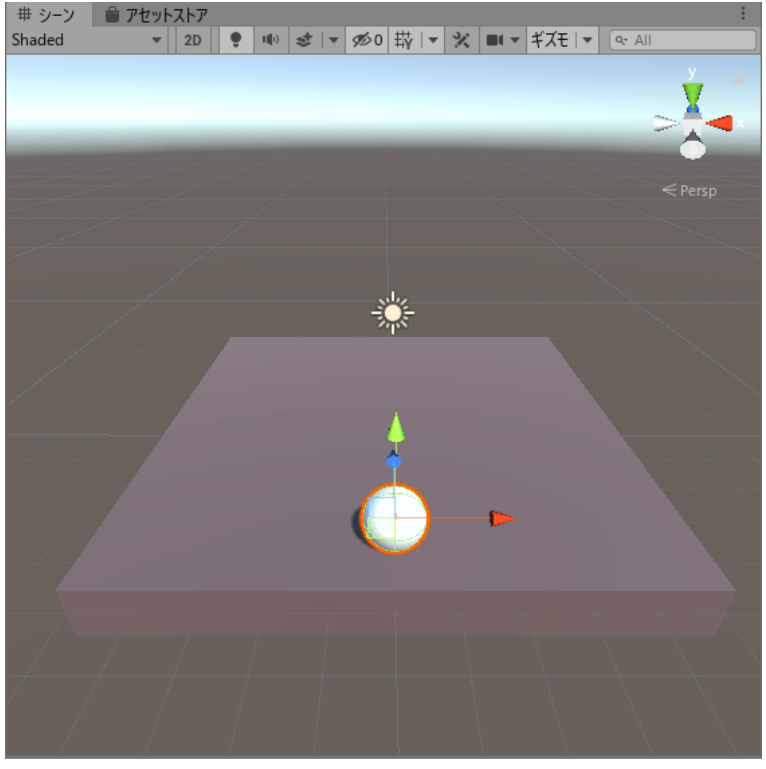
重力を設定（Y軸のマイナス方向に重力加速度9.81m/s^2）

```
Physics.gravity = new Vector3(0f, -9.81f, 0f)
```

重力を設定（ファンタジーの世界でZ軸方向に重力加速度1.0m/s^2）

```
Physics.gravity = new Vector3(0f, 0f, 1.0f)
```

図3-3　ファンタジー世界の重力（SphereにはRigidbodyコンポーネントをアタッチ済）



**COLUMN　3D空間のベクトルを表すVector3と演算**

教科書にはVector3の計算を色々紹介しているが、ベクトルを計算することは単純にX倍する以外は殆ど無い。ただし、normalizedプロパティは方向性を表すのでよく使う。

## スクリプトの起動とパラメータの調整

オブジェクトにアタッチしたスクリプトはコンポーネントとして扱われ、**publicなメンバ変数はインスペクター上で初期値を設定することができる**。

### TIPS　SerializeFieldアトリビュート

publicなメンバ変数以外にも、[SerializeField]の記述を付けたメンバ変数はアクセス修飾子がprivateでもインスペクター上で初期値を設定できる。

例）

```
[SerializeField]
private float gravityScale = 1.0f;
```

### COLUMN　Unityの標準言語C#

昔はUnityScript等が使われていたが、サポート外になった。

## 加速度センサーの利用

（講義では扱わない為、省略）

### TIPS　プラットフォームごとの処理の振り分け

（講義では扱わない為、省略）

## ビルドと実機検証

（講義では扱わない為、省略）

**COLUMN　Unityプロジェクトのバージョン管理**

バージョン2019.3.0.f6では、教科書に記述している内容を満たしている。.gitignoreファイルについては、以下のリンクで最新版を入手できる。

gitignore/Unity.gitignore at master · github/gitignore · GitHub

# 3-5　物理特性の設定

重さ、反発係数を学ぶ。

## 重さの設定

オブジェクトの重さはアタッチされているRigidbodyコンポーネントの**Mass**パラメーターで設定できる。

**COLUMN　Rigidbodyのパラメーター**

以下のリンクに詳細な説明がある。プロパティ**Use Gravity**、**Is Kinematic**、**Constraints**はよく使う。

Rigidbody - Unity マニュアル

## Physic Materialによる反発係数の設定

プロジェクトビューでマテリアルを作成した要領で**物理マテリアル（Physic Material）**を作成することができる。物理マテリアルは設定項目に**Bounciness**（跳ね返ることのできる物質の特性）があり、0で全く弾まず、1で最大限に弾む。

作成した物理マテリアルはコライダーに設定することが可能。

# 3-6　ホールの作成とトリガーイベントの制御

接触・衝突検知を判断できるColliderコンポーネントについて学ぶ。

## コライダーの2種類の使い方

コライダーは2種類の使い方がある。

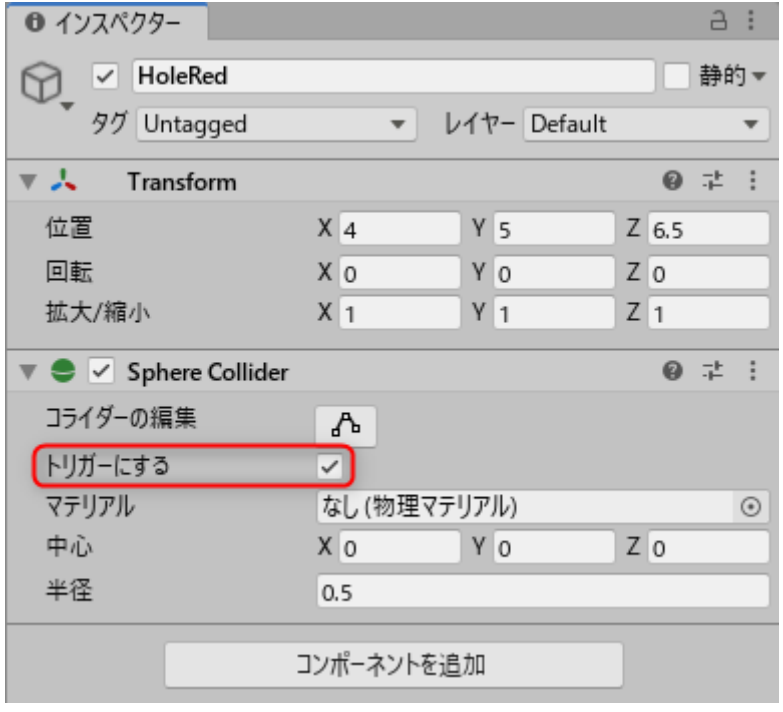
１．押し合い等の物理シミュレーション

２．オブジェクトが接触したことを判断する

## Holeオブジェクトの設置とトリガーの設定

コライダーをトリガーにするかどうかはコライダーコンポーネントのプロパティであるトリガーにするにチェックを入れるかどうかである。

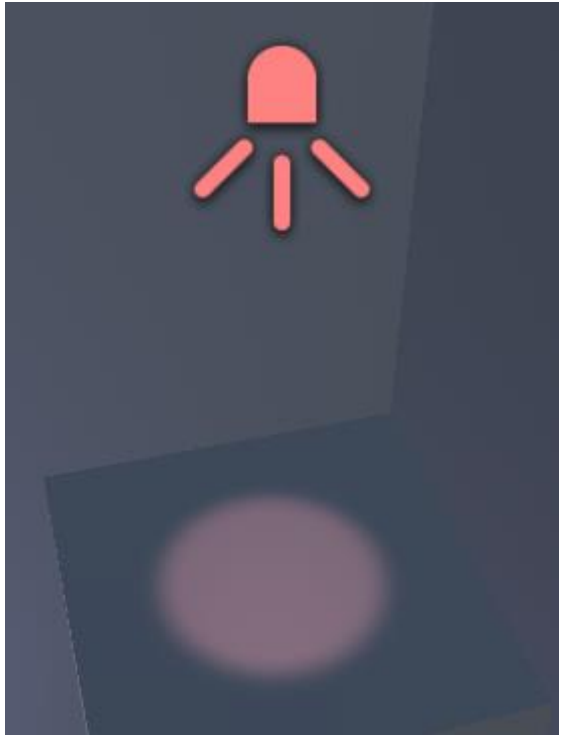
図2-4　トリガーにする


インスペクター
HoleRed
静的
タグ Untagged
レイヤー Default
Transform
位置 X 4 Y 5 Z 6.5
回転 X 0 Y 0 Z 0
拡大/縮小 X 1 Y 1 Z 1
Sphere Collider
コライダーの編集
トリガーにする
マテリアル なし (物理マテリアル)
中心 X 0 Y 0 Z 0
半径 0.5
コンポーネントを追加

## Spot Lightの利用

スポットライト（Spot Light）を用いれば、一部分だけ照らすライトを作ることができる。

図2-5　スポットライト



## Holeオブジェクトの複製と設定

（特記事項なし）

### COLUMN　オブジェクトの無効化

インスペクター上のゲームオブジェクト名の左にあるチェックボックスのチェックを外すと、そのゲームオブジェクトは無効化される（存在しないことになる）。コンポーネントのチェックボックスの操作も同様で、コンポーネントの有効・無効を切り替えることができる。

## タグによるオブジェクト識別の設定

衝突イベントの検知など、どのオブジェクトでイベントが起きたのか判別したい場合がある。その場合は各オブジェクトに識別用の名前を付ける機能である**タグ**の仕組みを利用するとよい。タグはプロジェクト単位で共有され、複数のオブジェクトに同一のタグをつけることも可能。タグの設定・追加はインスペクターのタグ欄から行うと便利。

図2-6　インスペクター上からタグを追加しようとしているところ

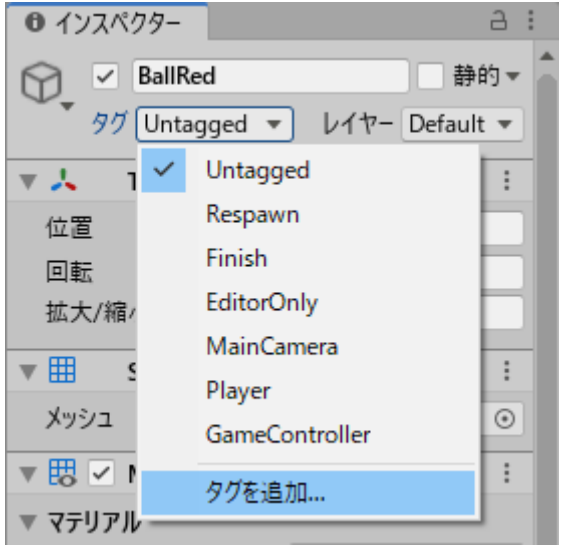
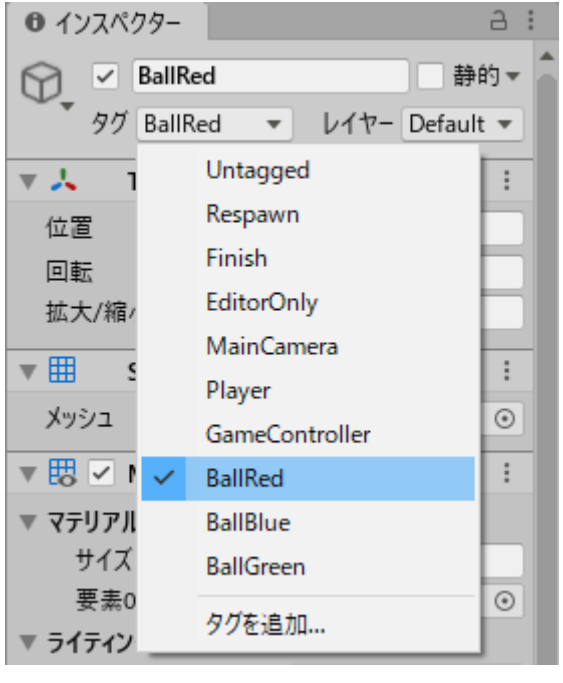



図2-7　追加されたタグ


インスペクター
BallRed
静的
タグ BallRed
レイヤー Default
Untagged
Respawn
Finish
EditorOnly
MainCamera
Player
GameController
BallRed
BallBlue
BallGreen
タグを追加...
位置
回転
メッシュ
マテリアル
サイズ
ライティン

## トリガーイベントの検知とRigidbodyの操作

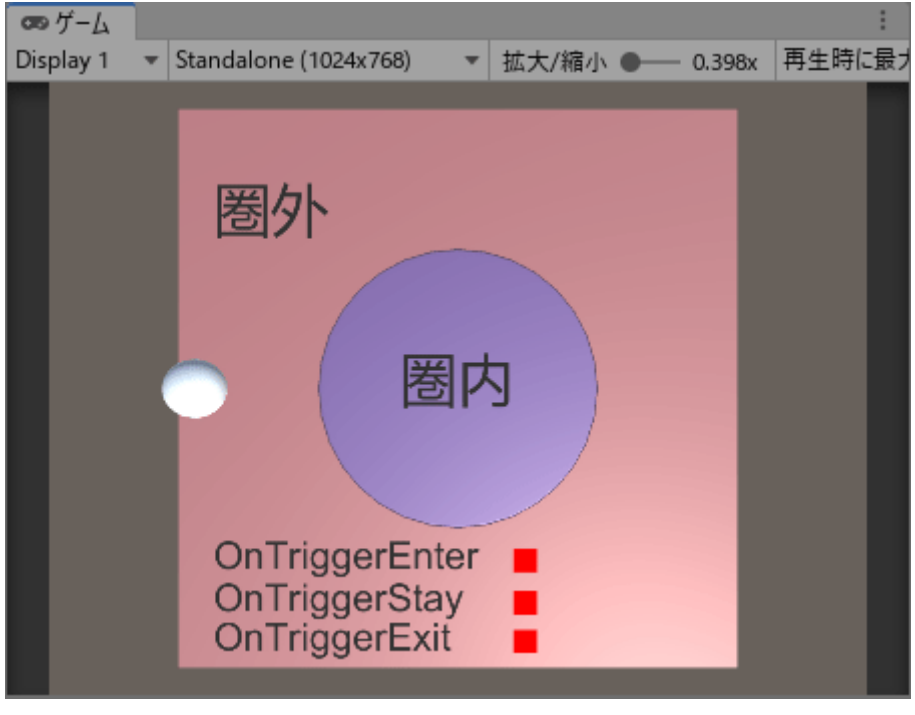
プロパティ**トリガーにする**に対してチェックを付けている**コライダー**をアタッチしているゲームオブジェクトは、他のオブジェクトとの接触を検知することができる。検知にはOnTriggerEnter、**OnTriggerStay**、OnTriggerExitメソッドを使う。全てMonoBehaviourクラスのメソッド。

| | |
|---|---|
| **OnTriggerEnter** | 他コライダーが自コライダー内部に入ってきたことを検知する。このメソッドが呼ばれるのは内部に入ってきた瞬間の1度だけ。 |
| **OnTriggerStay** | 他コライダーが自コライダー内部に留まっていることを検知する。他コライダーが留まり続けている限り、毎フレームこのメソッドが呼び出される。 |
| **OnTriggerExit** | 他コライダーが自コライダー内部から出たことを検知する。このメソッド |

が呼ばれるのは内部から出た瞬間の1度だけ。

UnityEngine.MonoBehaviour - Unity スクリプトリファレンス

図2-8　OnTriggerXXXイベント（Enter・Exitはイベントが起きた瞬間に■を青色に、Stayはイベント中は青色・それ以外は赤色）


ゲーム
Display 1
Standalone (1024x768)
拡大/縮小
0.398x
圏外
圏内
OnTriggerEnter
OnTriggerStay
OnTriggerExit

上記3つのメソッドは**Collider**型の仮引数を持っていて、仮引数の変数**gameObject**を利用することで、接触したコライダーコンポーネントを含むゲームオブジェクトを取得できる。

より正確に言うと、**Component**型のインスタンスはpublicな変数gameObjectを持っている。**ColliderクラスはComponentクラスを継承している**ので、変数gameObjectを利用できる。

UnityEngine.Component - Unity スクリプトリファレンス

UnityEngine.Collider - Unity スクリプトリファレンス

（再開ポイント）

ゲームオブジェクトのインスタンスに対して**GetComponent**メソッドを使うと、ジェネリックに指定した型のアタッチされたコンポーネントを取得することができる。複数ある場合はひとつのみ取得する。複数取得する際は**GetComponents**メソッドを使う。

Component-GetComponent - Unity スクリプトリファレンス

Component-GetComponents - Unity スクリプトリファレンス

Vector3構造体は**Normalize**メソッドと変数**normalized**を持つ。NormalizeメソッドはVector3の値自体を大きさ1にするメソッドで、変数normalizedはVector3の大きさ1の値を返す。ベクトルを大きさ1の状態にすることを**正規化**と呼ぶ。

※ベクトルについてよくわからない方は、正規化されたベクトルは方向を表す　と覚えればよい。

変数**gameObject**のタグを取得する場合は、**gameObject.tag**と記述すれば良い。変数tagはタグの名称を格納しているpublicな変数である。

GameObject-tag - Unity スクリプトリファレンス

リジッドボディコンポーネントの速度は**リジッドボディのインスタンス.velocity**と記述すれば取得できる。代入も可能。

Rigidbody-velocity - Unity スクリプトリファレンス

オブジェクトに力を加えるには**AddForce**メソッドを使う。

Rigidbody-AddForce - Unity スクリプトリファレンス

ForceMode - Unity スクリプトリファレンス

# 3-7　ゲームクリアの判定

## Enter、Exitトリガーイベントの検知

（特記事項なし）

## ホールの状態監視と簡易GUIの表示

**OnGUI**はほぼ使う機会が無く、**uGUI**をメインに使うとよい。uGUIについてはCHAPTER5で改めて説明する。

### TIPS　仮UIの表示サイズ

（ほぼ使う機会が無いので省略）

### COLUMN　簡易的な表示を行うシステムOnGUI

**OnGUI**はほぼ使う機会が無く、**uGUI**をメインに使うとよい。uGUIについてはCHAPTER5で改めて説明する。

### COLUMN　その他のスクリプトエディターの選択

標準のエディターであるVisual Studio以外にもエディターを指定できるが、講義ではVisual Studioのみ使用する。

# 3-8　GIによるグラフィックの最終調整

## GIの有効化

　親子関係にあるオブジェクトの親オブジェクトに対してStaticのチェックを有効にすると、全ての子オブジェクトに反映するかどうかを問われる。GIを有効化する際にはYesを選択すれば良い。

## ライティングの調整

　Indirect Intensityの値がグレーアウトしている場合は、realtime GIが有効になっていない可能性がある。

## COLUMN　ライトマップによる負荷の軽減

高原の処理は非常に高負荷なため、モードをベイクにすれば事前にライティングの計算を済ませることができる。ただし、光源が動かない場合に限る。このように、事前に影や光をテクスチャに書き込む機能をベイクド・ライトマップと呼ぶ。ベイクド・ライトマップ以外に徐々にライティング処理を行うプログレシップ・ライトマッパーという機能がある。ベイクド・ライトマップの欠点である動くオブジェクトに対してライティングできない点は、事前計算しておいた光の情報を近似的に反映させるライト・プローブという機能を使うことである程度回避できる。

ベイクしたライティング - Unity マニュアル
プログレッシブ CPU ライトマッパー - Unity マニュアル
移動するオブジェクトのためのライトプローブ - Unity マニュアル

図2-9　ベイク

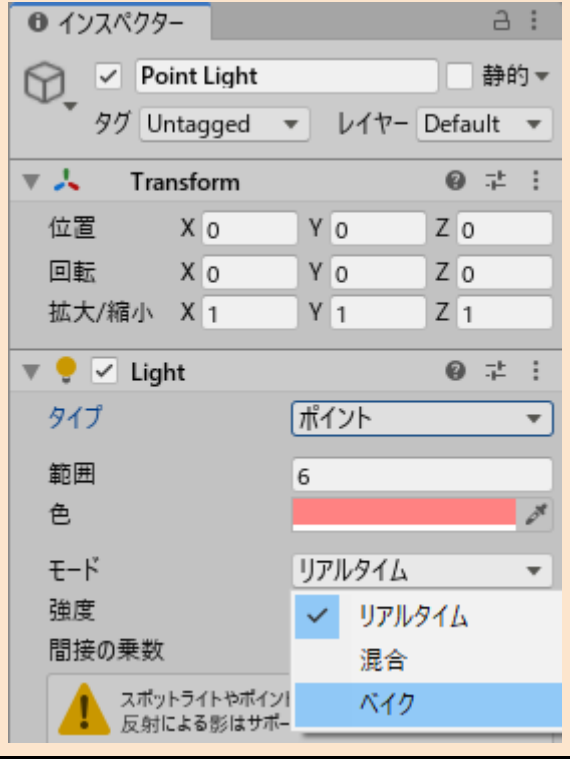

インスペクター
Point Light
静的
タグ Untagged
レイヤー Default
Transform
位置 X 0 Y 0 Z 0
回転 X 0 Y 0 Z 0
拡大/縮小 X 1 Y 1 Z 1
Light
タイプ
ポイント
範囲
6
色
モード
リアルタイム
強度
間接の乗数
リアルタイム
混合
ベイク

図2-10　ベイクしていないスポットライト10個（FPS2858）

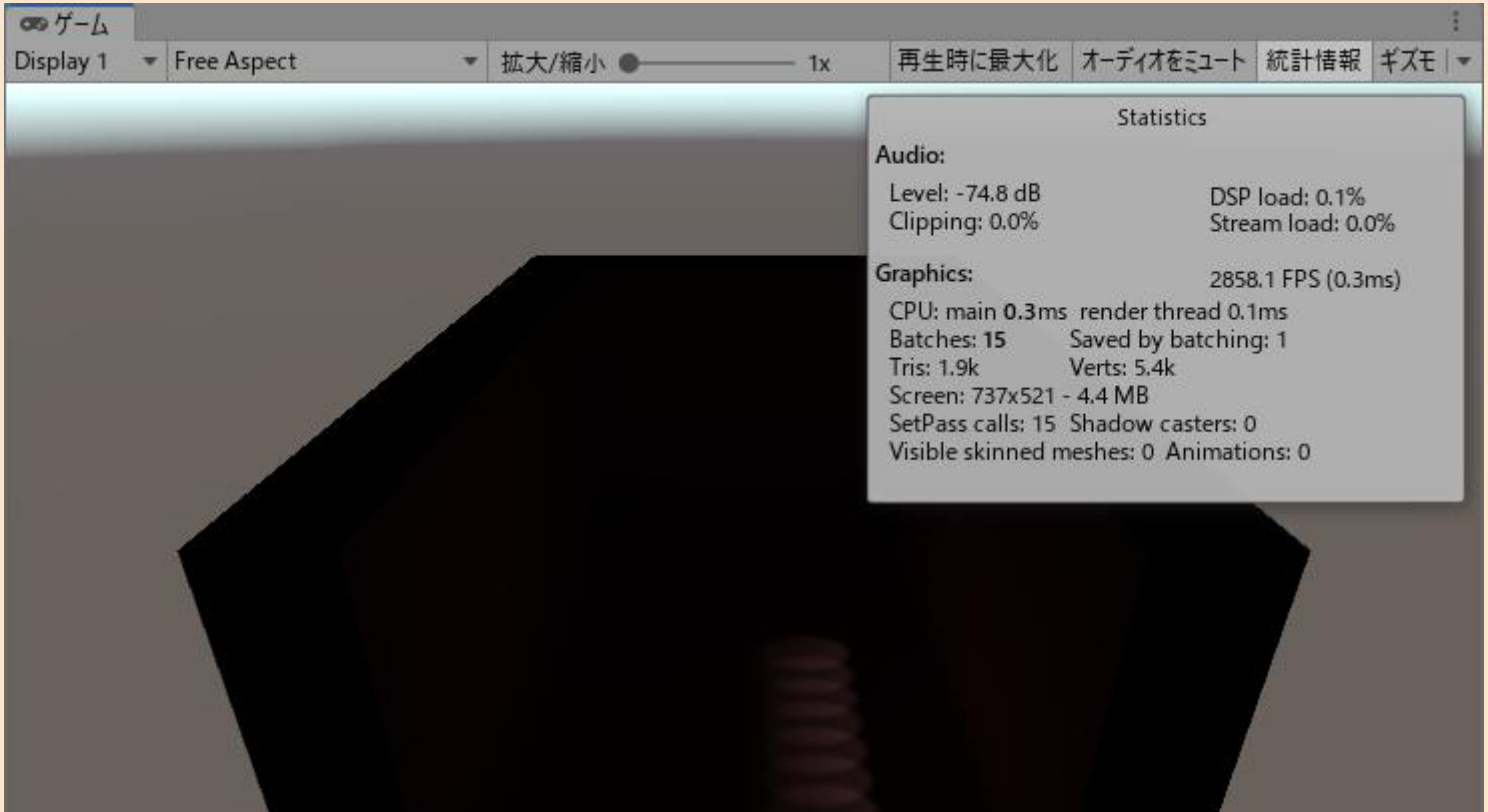

ゲーム
Display 1
Free Aspect
拡大/縮小
1x
再生時に最大化
オーディオをミュート
統計情報
ギズモ
Statistics
Audio:
Level: -74.8 dB
Clipping: 0.0%
DSP load: 0.1%
Stream load: 0.0%
Graphics:
2858.1 FPS (0.3ms)
CPU: main 0.3ms render thread 0.1ms
Batches: 15
Saved by batching: 1
Tris: 1.9k
Verts: 5.4k
Screen: 737x521 - 4.4 MB
SetPass calls: 15
Shadow casters: 0
Visible skinned meshes: 0
Animations: 0

図2-11　ベイクしていないスポットライト100個（FPS2170）

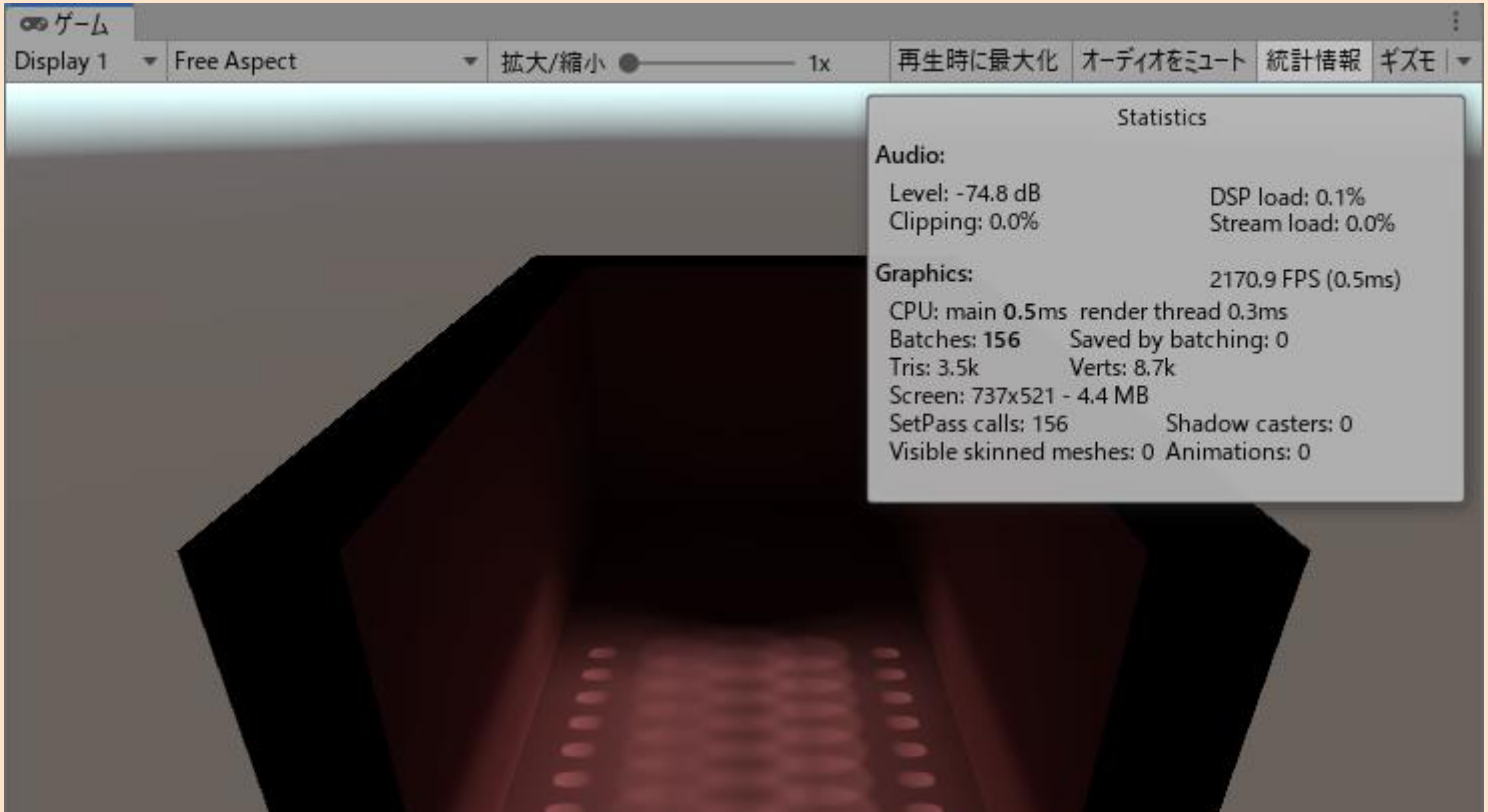

ゲーム
Display 1
Free Aspect
拡大/縮小
1x
再生時に最大化
オーディオをミュート
統計情報
ギズモ
Statistics
Audio:
Level: -74.8 dB
Clipping: 0.0%
DSP load: 0.1%
Stream load: 0.0%
Graphics:
2170.9 FPS (0.5ms)
CPU: main 0.5ms render thread 0.3ms
Batches: 156
Saved by batching: 0
Tris: 3.5k
Verts: 8.7k
Screen: 737x521 - 4.4 MB
SetPass calls: 156
Shadow casters: 0
Visible skinned meshes: 0
Animations: 0

図2-12　ベイクしていないスポットライト500個（FPS951）

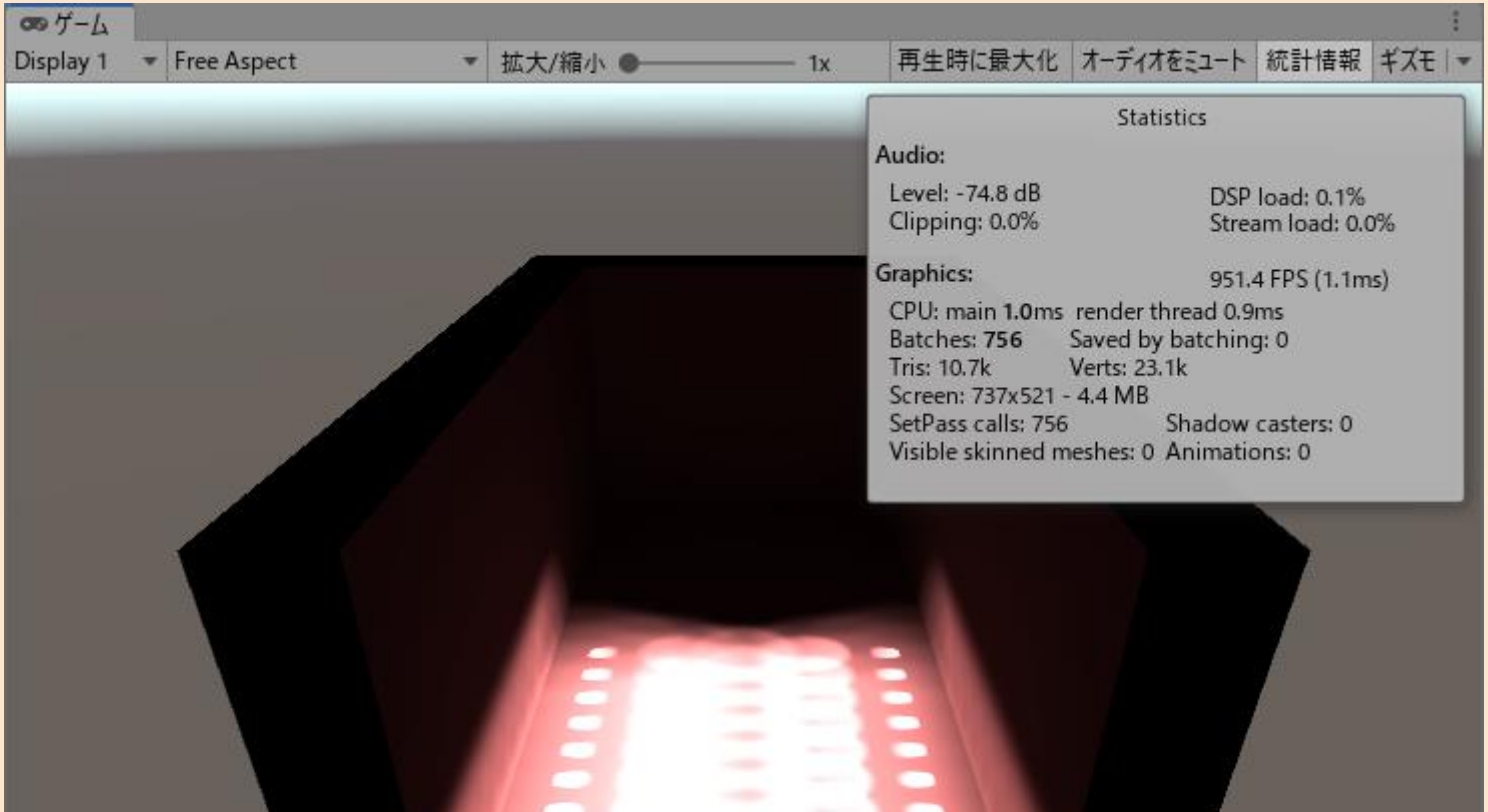

ゲーム
Display 1
Free Aspect
拡大/縮小
1x
再生時に最大化
オーディオをミュート
統計情報
ギズモ
Statistics
Audio:
Level: -74.8 dB
Clipping: 0.0%
DSP load: 0.1%
Stream load: 0.0%
Graphics:
951.4 FPS (1.1ms)
CPU: main 1.0ms render thread 0.9ms
Batches: 756
Saved by batching: 0
Tris: 10.7k
Verts: 23.1k
Screen: 737x521 - 4.4 MB
SetPass calls: 756
Shadow casters: 0
Visible skinned meshes: 0
Animations: 0

図2-13　ベイクしていないスポットライト1000個（FPS471）

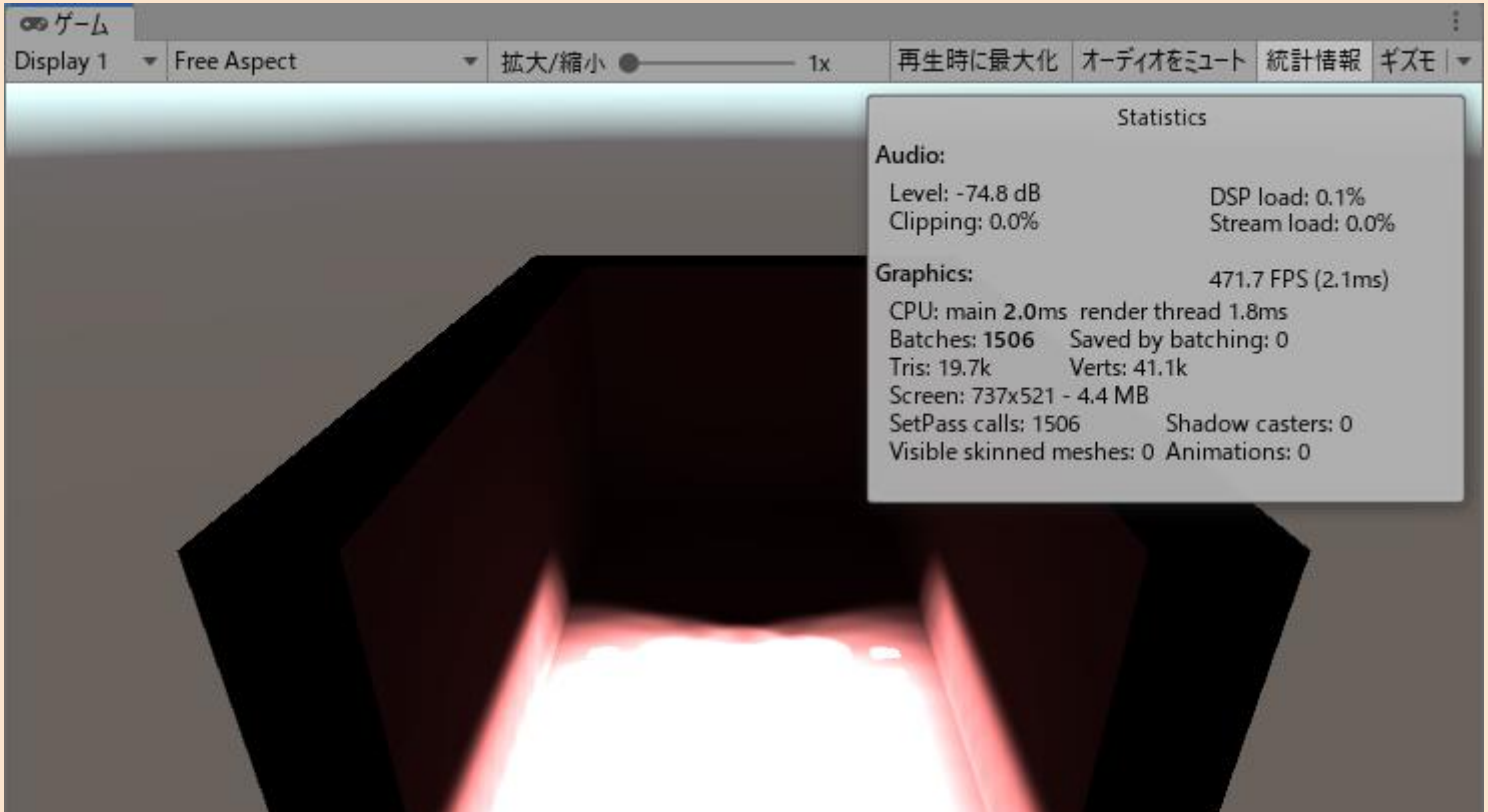

ゲーム
Display 1
Free Aspect
拡大/縮小
1x
再生時に最大化
オーディオをミュート
統計情報
ギズモ
Statistics
Audio:
Level: -74.8 dB
Clipping: 0.0%
DSP load: 0.1%
Stream load: 0.0%
Graphics:
471.7 FPS (2.1ms)
CPU: main 2.0ms render thread 1.8ms
Batches: 1506
Saved by batching: 0
Tris: 19.7k
Verts: 41.1k
Screen: 737x521 - 4.4 MB
SetPass calls: 1506
Shadow casters: 0
Visible skinned meshes: 0
Animations: 0

図2-14　**ベイクした**スポットライト1000個（FPS553）

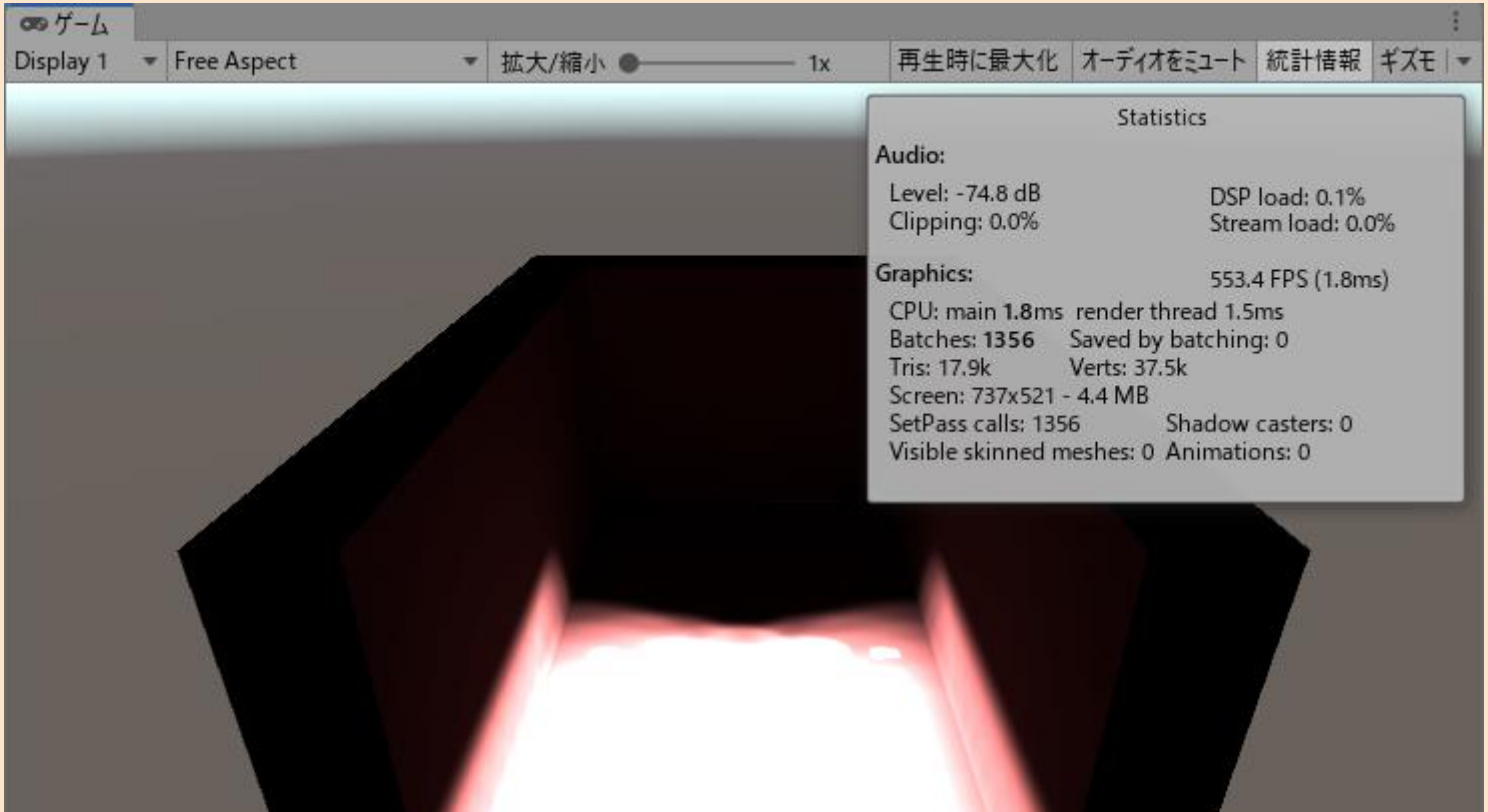
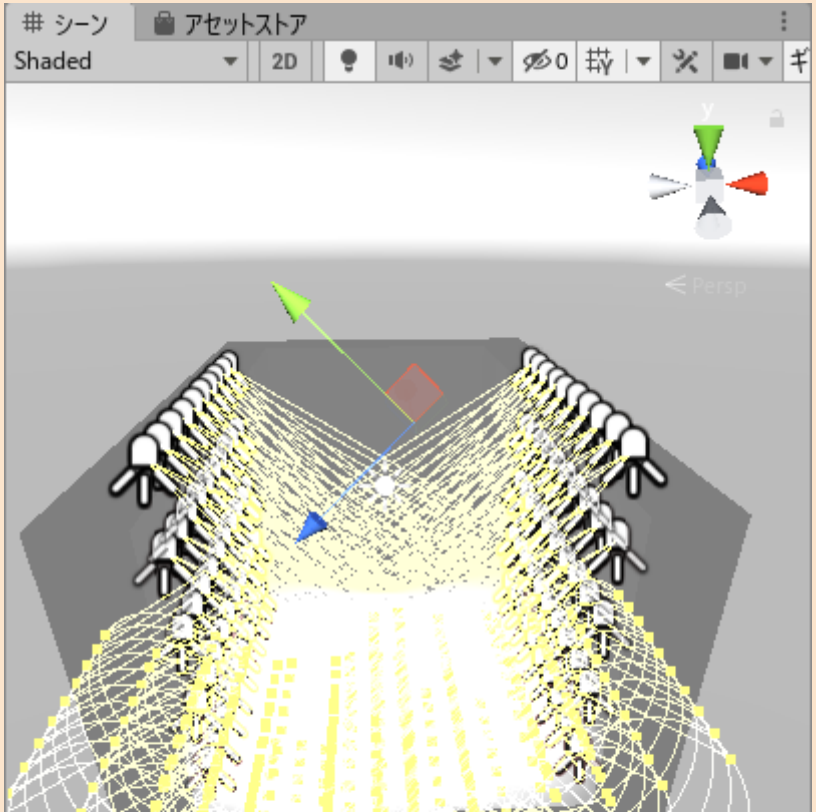

ゲーム
Display 1
Free Aspect
拡大/縮小
1x
再生時に最大化
オーディオをミュート
統計情報
ギズモ
Statistics
Audio:
Level: -74.8 dB
Clipping: 0.0%
DSP load: 0.1%
Stream load: 0.0%
Graphics:
553.4 FPS (1.8ms)
CPU: main 1.8ms render thread 1.5ms
Batches: 1356
Saved by batching: 0
Tris: 17.9k
Verts: 37.5k
Screen: 737x521 - 4.4 MB
SetPass calls: 1356
Shadow casters: 0
Visible skinned meshes: 0
Animations: 0

図2-15　スポットライト配置図


シーン
アセットストア
Shaded
2D
Persp

## まとめ　物理エンジンとコリジョンイベントをマスターする

- 物理挙動を扱いたい場合は**リジッドボディコンポーネント**をアタッチする
- リジッドボディに**力**を加えれば、オブジェクトを動かすことができる
- オブジェクトは**親子関係**を持つことができる
- 親子関係をもつオブジェクトは位置が親：**グローバルポジション**、子：**ローカルポジション**となるので注意が必要
- **空のオブジェクト**でオブジェクトをまとめることができる
- **コライダーコンポーネント**を使うことでオブジェクトが**作用・反作用**をするようになったり、**トリガー**として設定できる
- **Inputクラス**を使うことでキーボード入力を制御できるようになる
- アクセス修飾子public、もしくは**SerializeFieldアトリビュート**付与でメンバ変数をコンポーネントのプロパティにできる
- 反発係数などの物理特性を持つ**物理マテリアル**を作成・オブジェクトに適用すると物理特性を得られる
- ゲームオブジェクトは**タグ**を持つことができ、オブジェクトの判別に役立つ
- ライティングで**ベイク**できる場合は積極的にベイクを行う