ゲームのための UNITY

E ブック

# アーティストのための 2D ゲームアート、アニメーション、ライティング

Unity®

# 目次

# はじめに

1980 年から 90 年代にかけて、2D ゲームを遊んで育った人たちにとっては、2D ゲームはノスタルジーを誘う存在です。しかし『*Cuphead*』、『*Hollow Knight*』、『*Among Us*』、『*Skul: The Hero Slayer*』、『*Ori*』シリーズなどの Unity で制作されたタイトルが示すように、今日の 2D ゲームの今までにない革新的なその動向は留まるところを知りません。

ハードウェア、グラフィックス、ゲーム開発ソフトウェアの進化によって、2D ゲームでリアルタイムライトや高解像度のテクスチャを使えるようになり、スプライトの数の制限も、無いも同然のものとなりました。2D グラフィックスの平面性を扱うことで、アーティストは漫画のようで、常識にとらわれない造形美をどのデバイスでも映えるように創造できるようになりました。

この e ブックは、Unity が提供する中でも一番手厚く、多岐にわたるコンテンツを取りまとめた 2D ゲーム開発ガイドです。ある程度 Unity を使用した経験があり、個人またはチームで商用 2D ゲームの制作を目指している開発者やアーティストを対象としています。

本ガイドは Unity の 2D 専門家からの着想を元に執筆されており、ひたむきでエネルギー溢れるクリエイターたちが、プロジェクト立ち上げの際に行うべき重大な判断事項についてわかりやすくするだけでなく、Unity の 2D ツールセットを最大限に使えるように背中を押してくれます。

取り上げているトピックは、プロジェクトの設定、Unity とデジタルコンテンツ制作（DCC）ソフトウェアとのラウンドトリップ、スプライトの作成、レベルデザインのためのレイヤーのソート、カメラの設定、アニメーション、ライト、ビジュアルエフェクトなどがあります。その間に、たくさんの最適化にむけたヒントも紹介されています。

Unity を初めて使用するであったとしても、本ガイドからさまざまな役立つヒントや手法を知ることができるのではないかと思います。とはいえ、先に Unity Learn で無料の初級者向けチュートリアルとコースを受講することをお勧めいたします。当ガイドをお楽しみいただき、実際のゲーム開発で活かしていただければ幸いです。2D クリエイターには明るい未来が待っています。


Unity で作成された美しい 2D ゲーム – 左上から時計回り：『*Ori and the Will of the Wisps*』（Moon Studios 制作）、『*Cuphead*』（Studio MDHR 制作）、『*MegaSphere*』（AK Games 制作）、『*Hollow Knight*』（Team Cherry 制作）

## 執筆者

Jarek Majewski は独学で Unity 開発を学んだプロの 2D アーティストであり、広範な C# スクリプティングスキルを有しています。iOS、Xbox Indie Live、Nintendo Switch のゲーム向けのアートの制作経験があります。現在開発中のゲーム『*Ultimate Action Hero*』は、2019 Unity 2D Challenge で 2 位になりました。Unity の 2D 専門家と共同で開発した『*Dragon Crashers*』という 2D デモプロジェクトではアートディレクターを務めました。このデモプロジェクトでは、Unity 2020 LTS 時点で使用可能となっている 2D 機能が使われています。

Jarek は多くの Unity の専門家や Unity の担当者と緻密な連携の下に、2D ゲーム制作ガイドの決定版となる、本書を上梓しました。

## Unity の貢献者

Eduardo Oriz は本ガイドの制作についての指揮を担っております。Unity でシニアコンテンツマーケティングマネージャーを務める彼は、2D ツールチームをはじめとして Unity の開発チームと長年、共に業務に勤しんでおり、ゲーム開発者やゲームスタジオに対して Unity が提供している製品やサービスについて幅広い知見を持っています。

Rus Scammell は Unity の 2D 開発チームのプロダクトマネージャーです。15 年以上にわたるゲームとソフトウェア開発の経験を持っております。ゲームテクノロジーに関する広範な知識を駆使し、Unity の 2D ツールとワークフローをアーティスト、プログラマー、ゲームデザイナーたちにとって、扱いやすいものになるよう尽力しております。

Andy Touch は Unity のシニアコンテンツ開発者です。『*Lost Crypt*』や『*Dragon Crashers*』など、多数のプロジェクトに携わっております。Unity をダウンロードしてからというもの、彼は 100 個のキューブに 3D 物理演算を追加してその跳ね返る様子を観察するなど、ゲームテクノロジーの実験に明け暮れております。Unity に所属する前は、大学生にゲーム開発を指導していました。

最後に、当ガイドの執筆に貢献してくださったコピーライター、エディター、グラフィックデザイナー、Unity 社内において制作、推敲、配布にご協力くださった皆さまに感謝申し上げます。

> このe ブックに記載されているヒントとワークフローは Unity 2020 LTS バージョンの 2D ツールセットを基にしておりますが、それ以降のバージョンの Unity（Unity 2021 LTS など）でも使用することが可能です。

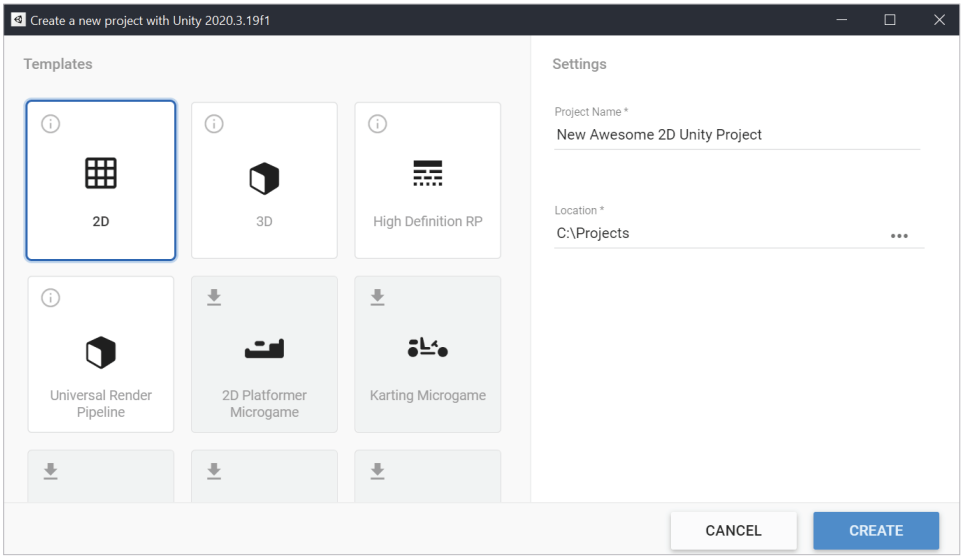
# プロジェクトの設定：2D レンダリング

Unity では、3 つのレンダーパイプライン（1 つのビルトインレンダーパイプラインと 2 つのスクリプタブルレンダーパイプライン（SRP））が提供されています。この 2 つの SRP とは、ユニバーサルレンダーパイプライン（URP）と HD レンダーパイプライン（HDRP）のことです。URP はすべてのプラットフォームに適用されます。一方、HDRP はハイエンド PC とコンソールを対象としたゲームで使用されるように設計されています。

本書では URP を使用します。理由としては、URP では 2D ライティング用のグラフィックスパイプラインが用意されており、2D ライトとライティングエフェクト（Freeform、Sprite、Spot、Global）を作成することができるからです。URP にはシェーダーグラフ、ポストプロセスエフェクト、カメラスタッキング機能との互換性があるというのもあります。

2D Renderer を使用するには、最初に Unity Hub から 2D プロジェクトテンプレートをインストールしましょう。そして、URP をインストールします。2D テンプレートでは、以下の形で、はじめからプロジェクト設定がいくつか 2D ゲーム向けに最適化されています。

- 画像がスプライトとしてインポートされるよう、スプライトモードに設定されている
- シーンビューが 2D に設定されている
- デフォルトのシーンにライトが含まれない
- カメラのデフォルト位置が 0、0、−10 になっている
- カメラが「Orthographic」に設定されている
- 「Lighting」ウィンドウの設定：
  - 全グローバルイルミネーションが無効化されている
  - 「Skybox Material」が「None」へ設定されている
  - 「Ambient Source」が「Color」へ設定されている



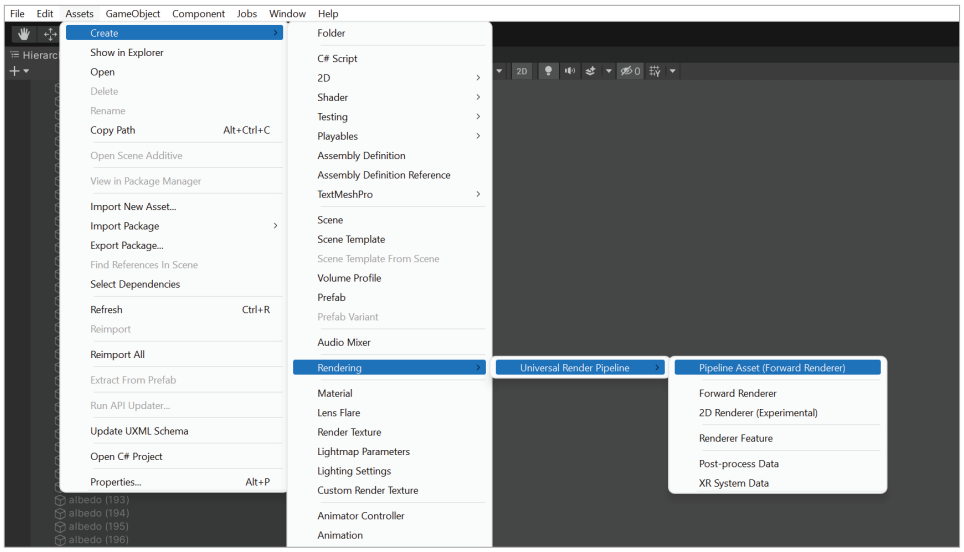

Unity Hub ウィンドウ

### Package Manager で 2D パッケージを選択する

過去に Unity を利用された方は、多数の機能がモジュール式のパッケージとして提供されていることはご存じだと思います。利用可能な各パッケージのバージョンは、エディターの**「Window」>「Package Manager」**で確認できます。ここで、各プロジェクトのパッケージをインストール、削除、無効化、更新することもできます。2D テンプレートに含まれているパッケージは、2D Animation、2D Pixel Perfect、2D PSD Importer、2D SpriteShape です。

次は Package Manager で URP をインストールする必要があります。


Package Manager で URP をイントールする



URP アセットを作成する

### 新規の URP アセットを作成する

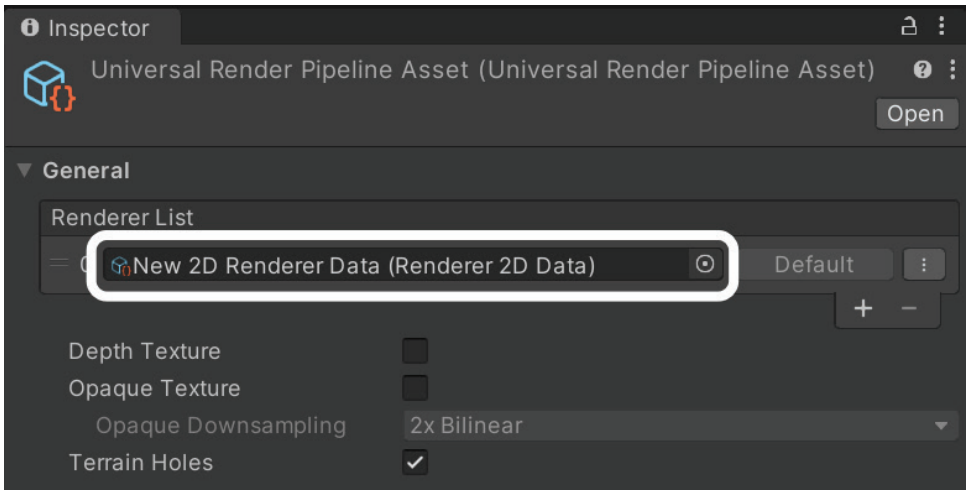
URP を使用するには、URP アセットを作成して、そのアセットをグラフィックス設定で割り当てる必要があります。

「Project」ウィンドウで右クリック（またはツールバーの「Assets」メニューをクリック）してから、**「Assets」>「Create」>「Rendering」>「Universal Render Pipeline」>「Pipeline Asset (Forward Renderer)」**の順に選択します。

URP アセットを作成すると、同時にレンダラーアセットも作成されます。これは 2D Renderer アセットに置き換える必要があります。置き換えるには、「Project」ウィンドウで右クリックするか、「Assets」メニューをクリックして、**「Assets」>「Create」>「Rendering」>「Universal Render Pipeline」>「2D Renderer」**の順に選択します。

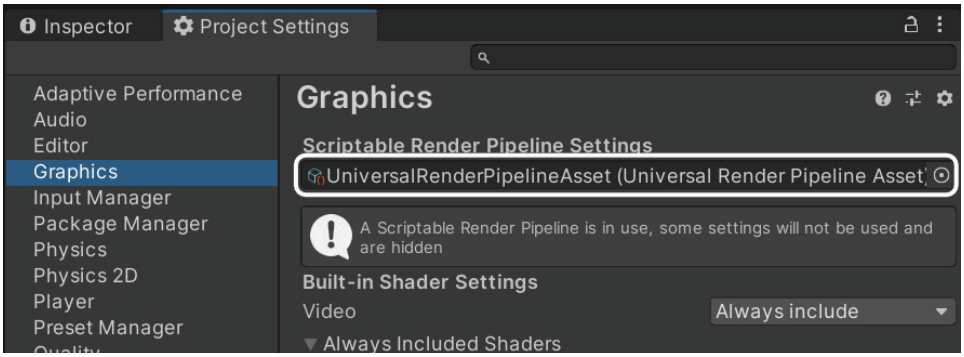
### 2D Renderer を割り当てる

URP アセットをクリックし、インスペクターで 2D Renderer アセットを「**Renderer List**」に割り当てます。フィールドの右端にある丸いボタンをクリックして、リストから選択します。



URP アセットの「Inspector」ビューで新しいレンダラーを設定する

URP アセットと 2D Renderer アセットが揃ったら、プロジェクトの「Scriptable Render Pipeline Settings」に 2D Renderer を割り当てます。「Project Settings」ウィンドウを開き（**「Edit」>「Projects Settings」**）、「Graphics」セクションに移動して、そこでパイプラインアセットを割り当てます。




「Project Settings」でユニバーサルレンダーパイプラインアセットを割り当てる

これで 2D Renderer が設定され、準備が整いました。プロジェクト設定では、「Quality」カテゴリに切り替えてレンダリング設定を調整することもできます。レンダリング設定の詳細については、こちらを参照してください。

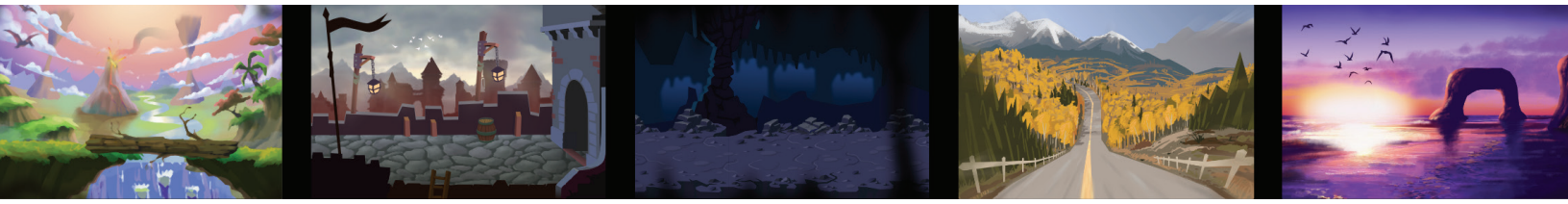
**ノート：**Unity 2021 以降のもので新規プロジェクトを開始する場合、2D テンプレート（「2D（URP）」という名前）を使用するだけで既に 2D Renderer が設定済となるため、上記の手順を行う必要はありません。

# ゲームのアート

コンセプトアートの段階では、プロジェクトの技術面に影響を及ぼすことになる、多くの決断を行うこととなります。

ゲームデザインでは、ゲームのアートスタイル、アクション、UI を提示するために、スクリーンショットやシリーズで画像のモックアップを作成するのが一般的です。モックアップは、コンセプトが有望かどうかを判断し、ゲームの最終的な見た目についてイメージをつかむ、手っ取り早い方法です。

また、シンプルなサムネイルのモックアップを使って、複数のアートの方向性を探ることもできます。こういったモックアップは、簡略化された形状にすることで、全体的な場景や、カメラの角度、オブジェクトのサイズ、カラーパレット、コントラストなどに注目できるようにします。サムネイルを使うのは、試行錯誤すべきさまざまなアプローチをチームへ伝えるのに素晴らしい手段です。



シンプルなサムネイルを描画してアイデアをテストする

モックアップを作成する際に考慮すべき事項を次にいくつか挙げます。

- ゲーム中のカメラ角度や視点はどうなるのか。
- プレイヤーキャラクターのサイズと、ターゲットとなるプラットフォームの画面サイズとの関係はどういう感じになるのか。
- ターゲットとなるプラットフォーム、テーマ、オーディエンスにアートスタイルが合致しているか。例えば、扱っているスタイルがカジュアルゲーマー、若年のプレイヤー、戦略ゲームファンなどに対して魅力的に映るかどうか。
- アートスタイルが全体的なグラフィックスのアプローチにどの程度適合しているか。
    - プレイヤーに反応速度の高さを求めるゲームの場合、プレイヤーキャラクター、敵、弾丸などの要素と背景を一目で区別できるようにするのがお勧めです。
    - モバイルゲームは、日光の下で小さな画面でも見えるように、明るく、高コントラストにしておきましょう。
- GUI 要素のサイズ、位置、可視性をどうするか。

## 技術的問題点

モックアップ段階で考慮すべき、重要な技術的問題点もあります。以下の点を踏まえましょう。

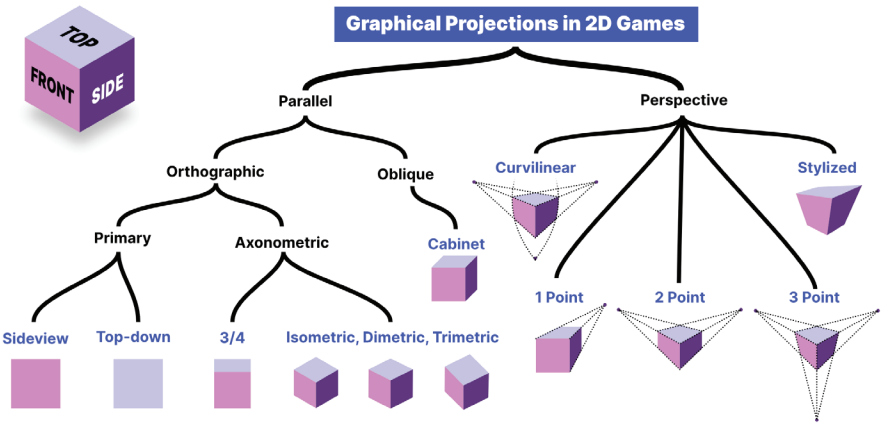
- **アニメーション：**スケルタルアニメーションを使用してアニメーション化するのは、どの要素か。どの要素をフレーム単位でアニメーション化し、どの要素をシェーダーでアニメーション化するのか。
- **環境：**タイルマップとスプライトシェイプのどちらを使って作成するか。また、シーンにプラットフォームスプライトを手動で配置するかどうか。これらツールの外観を模したモックアップをペイントしておけば、対象のモックアップをゲームで直接使用することもできます。
- **ソーティング：**Unity で行うソーティングの方法と同じようにモックアップレイヤーをグループ化することで、スプライトのソーティングを計画する。この作業にはある程度手間がかかってしまうかもしれないのですが、後の制作時に何千ものスプライトを分類しなければならない事態を避けることができます。
- **ライティング：**ライティングとシャドウをスプライトにペイントするのでしょうか。それともリアルタイムライティングを使用しますか？おすすめの 1 つの方法としては、ライティングのない画像をペイントし、画像編集ソフトウェアで別のレイヤーにシャドウとライトを追加していく方法があります。こうすると制作時に、後からいつでも必要に応じてライティングの様子を変更できます。

こういった段階できちんと時間を使い、最終的なゲームの様相にできる限り近いアセットを作成しておきます。そうすれば構想からゲーム制作への展開がより迅速にできるようになります。


『*Dragon Crashers*』の最初のコンセプトアートと最終版の比較。デモ用のアートの制作プロセスについては、このブログ記事をご覧ください。

# 視点の選択

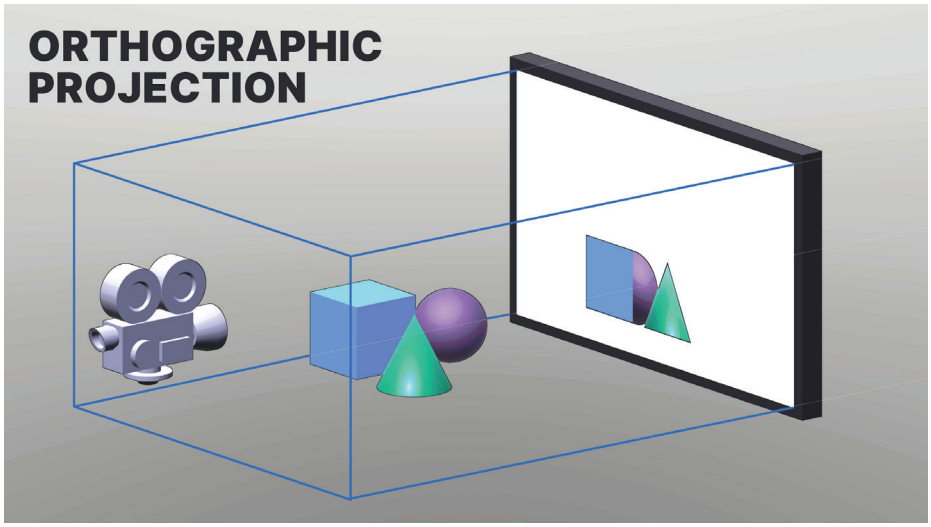
2D ゲームでは、さまざまな視点（ビュー）を使用できます。透視についての選択については、その概念をまとめて「Graphical Projection（グラフィカルプロジェクション）」と呼びます。ここで色々な投影法を見ていきましょう。




出典：Wikipedia

### 平行投影

平行投影というタイプに分類される投影は、3D 世界で平行な線は 2D 平面上に投影しても平行になります。また、消失点がないため、オブジェクトの大きさはカメラからどれだけの距離であっても同じです。アセットを拡大縮小したり、異なるサイズのアセットを描画したりする必要がないので、ゲームでは便利です。




この図は、平行投影の仕組みを示しています。カメラからの投影光線はすべて互いに平行であり、画像プレーンに対して垂直です。

平行投影の場合、光線は画像プレーンに対して垂直（直交）です。一方、斜投影の場合、光線は投影プレーンと斜角に交わって投影画像を生み出します。

平行投影には、Primary（プライマリ投影）と Axonometric（不等角投影）という 2 つのサブカテゴリがあります。

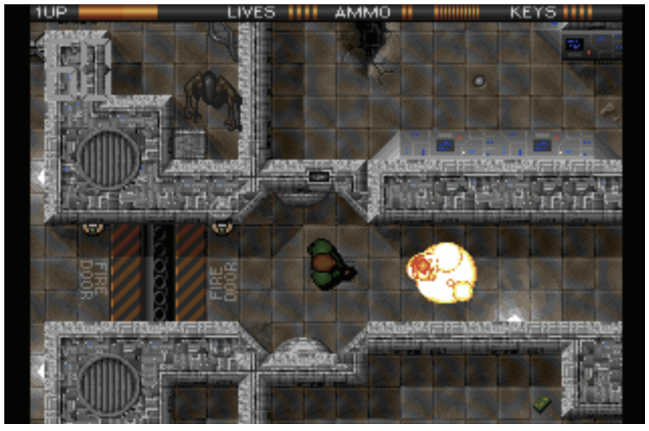
**Primary Projection（プライマリ投影）**

プライマリ投影では、ビュー平面に垂直な次元が 1 つ存在しています。この次元の線はすべて平行なので、投影の描画が完了した状態でも見えることはありません。この種の投影は、2 次元で描けば良いだけということもあり、一番簡単に描画できます。ゲームでは、2 種類あるプライマリ投影方法（サイドビューまたは見下ろし）のうちのどちらかを使用します。

サイドビューは、プラットフォーマー、シューティング、*メトロイドヴァニア*スタイルのゲームで使われます。画面上には 3D 世界の X 軸と Y 軸が描画され、Z 軸は省略されます。奥行きを与えるために、背景の遠くにあるオブジェクトは小さく描画される傾向にあります。


サイドビュー視点：1991 年版の『*Sonic The Hedgehog™*』（Sega 制作）

見下ろし視点では、カメラが地面に向かって垂直に下を向きます。完全な見下ろし視点のゲームは多くはありません。この視点は、¾ または等角投影法と混同されることがよくあります。見下ろし視点はシューティングゲームに最適ですが、人間のキャラクターを見る視点としては面白味がありません。

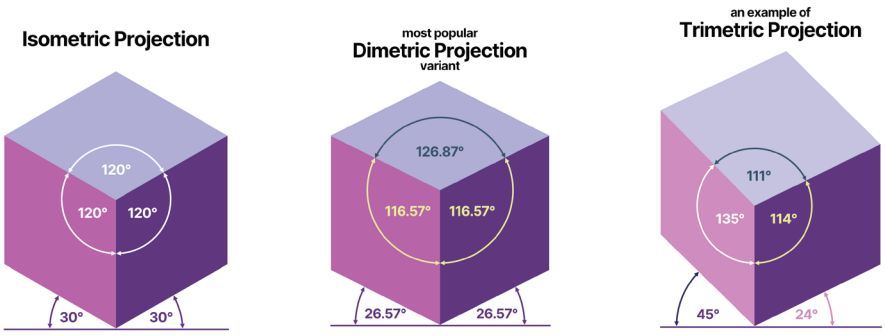



見下ろし視点：1992 年に Amiga 向けにリリースされた『*Alien Breed™*』（Team17 制作）

### Axonometric Projection（不等角投影）

不等角投影において、3D 世界は 1 つ以上の軸に沿って回転させるもので、これはオブジェクトの複数の側面を見せるものとなります。

不等角投影において、種別で主要なものは 3 つです。




等角投影、2 等角投影、斜方投影

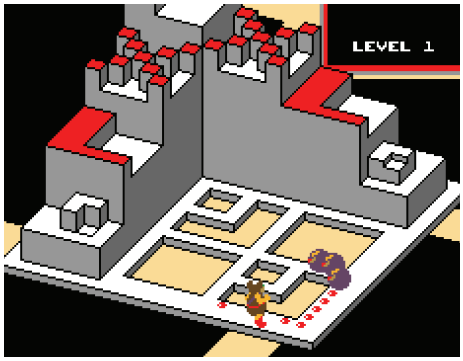
**Isometric Projection（等角投影）**：3 種のうち、最もすっきりした投影方法です。3 つの座標軸のうち任意の 2 つの間の角度がどれも 120 度で、X と Y の線はそれぞれ水平軸から 30 度の角度です。

昔のゲームで等角投影と言われていたもののほとんどは、実際は 2 等角投影です。等角投影の角度は、ピクセルアートでは見栄えが良くありません。そのため、実際に等角投影を扱うゲームは多くはありません。

**Dimetric Projection（2 等角投影）：**この視点では、2 つの角度が同一で、X と Y の線が 26.6 度の投影を使用します。2 等角は、現在も一番よくつかわれている投影です。これは古いハードウェアでは、線が構成するピクセル比が 2:1 であるため、見栄えが良かったのが理由です。

この投影法を使用することで、少なからず上面を見せることができます。

**斜方投影：**この投影法では、すべての角度が同一ではありません。




斜方投影：1983 年にリリースされた『*Crystal Castles*』（Atari 制作）

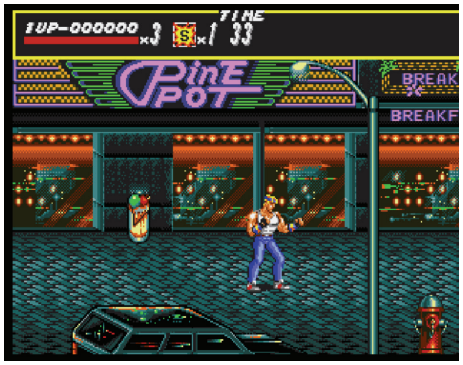
### 4 分の 3（¾）投影

この投影法は、見下ろし視点とよく混同されます。3/4 投影では、カメラが X 軸方向に少し傾いており、オブジェクトとキャラクターの前面が部分的に見えます。

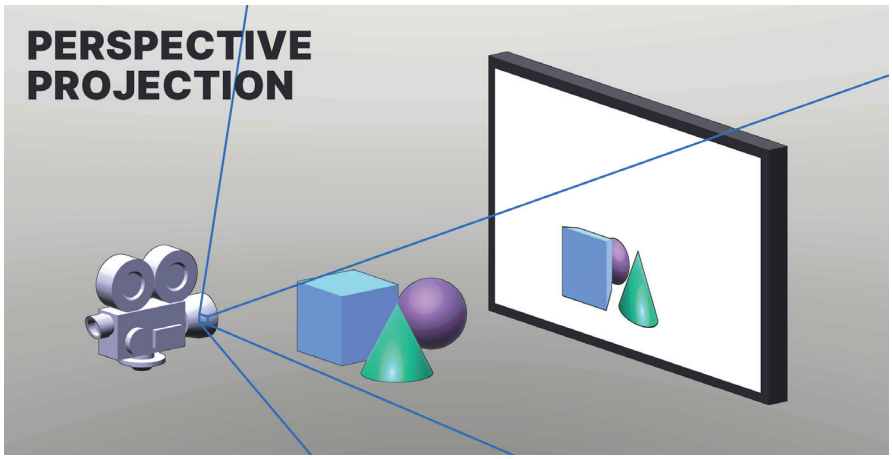

¾ 投影：1991 年にリリースされた『*The Legend of Zelda: A Link to the Past*』（Nintendo 制作）

### 斜投影

斜投影では、光線が投影プレーンと斜角に交わって投影画像を生み出します。最も一般的な斜投影のバリエーションはキャビネット投影で、ベルトスクロールアクションゲームでよく使われています。




斜投影：1991 年にリリースされた『*Streets of Rage: Mega Drive*』（Sega 制作）




透視投影では、投影光線は投影カメラの中心から放射されます。

## 透視投影

透視投影では、遠くのオブジェクトが小さく見える状態を再現するために消失点を利用します。消失点とは、透視図法における平面上の点のことで、3 次元空間において、2 次元的に投影または描画された、互いに平行な線が収束するようかのように映る点です。

消失点の数に応じて、1 点透視図法、2 点透視図法、3 点透視図法と呼び分けます。3D グラフィックスの登場により、このような透視図法は 2D ゲームではあまり使われなくなりました。しかし、いぜんとして、アドベンチャーゲームや背景が固定されているゲームでは効果的に利用することができます。

想定しているプロジェクトに適した視点を選ぶには、どうしたらよいでしょうか。ゲームプレイがアートスタイルを決めるのでしょうか。それとも、アートスタイルがゲームプレイを決めるのでしょうか。多くの場合、開発者はゲームのアイデアを現実に落とし込む際のアイデアを基盤として、アートスタイルとカメラの投影方法を選択するものです。決定はさまざまな条件に基づいて行います。例えば、以下のような要素が条件になります。

- ゲームプレイとジャンル
- ターゲットプラットフォーム
- シーンのわかりやすさ – 例：カメラの角度を高くすると、キャラクターと敵ユニットの視認性が高まる
- 予算 – 見下ろし視点の等角投影法のゲームの場合、複数の角度に対応するためにキャラクターアニメーションを増やす必要があったり、より高度なコーディングが必要になったりする可能性がある
- ターゲットとするオーディエンス

アートスタイルをゲームの基準にすることもできます。例えば、古代エジプト象形文字を基にしたゲームを想像してください。この場合、アートテーマによってカメラやゲームプレイのルールが決まるでしょう。つまり、必然的にプライマリ視点のサイドビューゲームになります。

2D 開発に初めて取り組む場合や予算が厳しい場合は、いずれかのプライマリ投影（見下ろし視点またはサイドビュー）を選択することをお勧めします。そうすると、描画とコーディングが併せて簡単になります。表示するのが 2 つの次元だけであれば、憂慮すべき点が少なくなります。また、そういったビューでは、スプライトソーティングの設定も簡単です。

留意していただきたい点としては、3D 空間の場合と同様に、投影は 100% 正確にする必要はなく、現実世界のルールに完全に従う必要もないということです。何より大事なのは、見栄えを良くし、ゲームプレイを向上させることなのです。

# アセットの解像度

元々、3D ゲームのために作られたエディターから Unity の 2D ツールは進化してきています。結果、数点独特な部分が特徴として存在しています。例えば、シーン内の 2D スプライトは、画面解像度に密接に結びついてはおりません。Unity のスプライトは、メッシュ上に描画されたテクスチャであるため、簡単に拡大縮小ができます。2D ゲームのカメラも拡大縮小可能であり、好きなようにズームイン、ズームアウトできます。

そのため、Unity で 2D コンテンツを作成する場合には、「Adobe Photoshop」、Serif 製の「Affinity Photo」、「GIMP」、「Krita」など従来のラスターグラフィックスソフトウェアで作業する場合とは異なるアプローチが必要になります。このような従来のアプリケーションでは、特定のドキュメントキャンバスサイズが決められた解像度で用意されており、全レイヤーがその解像度に結び付けられています。各レイヤーのピクセルサイズとドキュメントピクセルサイズの比は 1:1 となっています。


ゲームアセットを描画するときに、ピクセルサイズは一定であり、画像は常にピクセルグリッドに拘束されるため、見栄えが良くなります。ただし、Unity では、スプライトに異なる解像度を設定できます。また、カメラのズームレベルも最終的な見た目に影響します。

一方、Unity では、画面とアセットの解像度は互いに独立しているので、スプライトの解像度を計算する必要があります。

設定できる最大解像度はハードウェアの性能によって決まるので、ターゲットプラットフォームから計算を開始します。

モバイルデバイスの場合、取りうる解像度の範囲は広いですが、1920 × 1080 を想定すると安全です。この解像度を設定すれば、ローエンドからハイエンドまでのデバイスをターゲットにすることができます。

PC に関しては、Steam の調査によると、デスクトップゲーマーの大半がフル HD（1920 × 1080）を使っており、4K 画面を使っているのはたったの 2% です。とりわけ、ノート PC を使っている多くのユーザーが、1366 × 768 などの低解像度でゲームをプレイしています。さらに、ウルトラワイド画面のサポートの検討が必要な可能性もあります。21:9 などの画面比率をテストして、レベル内の映さなくてよい領域がカメラで表示されないようにします。

コンソールの場合、4K テレビの普及率が高いので、解像度は 4K が一般的です。Nintendo Switch は、携帯モードの解像度は 1280 × 720 で、HDMI で外部画面に接続した場合の最大解像度は 1920 × 1080 です。

このように、解像度の範囲というものは実に範囲が大きくなりうるものです。特定のデバイスをサポートしようと考えている場合は、こちらのまとめ記事を見ると、適切なターゲット解像度を特定するのに役立ちます。

フル HD または 4K をターゲットにする場合は、次の 2 つのベストプラクティスを考慮してみてください（このルールはピクセルアートには該当しません）。

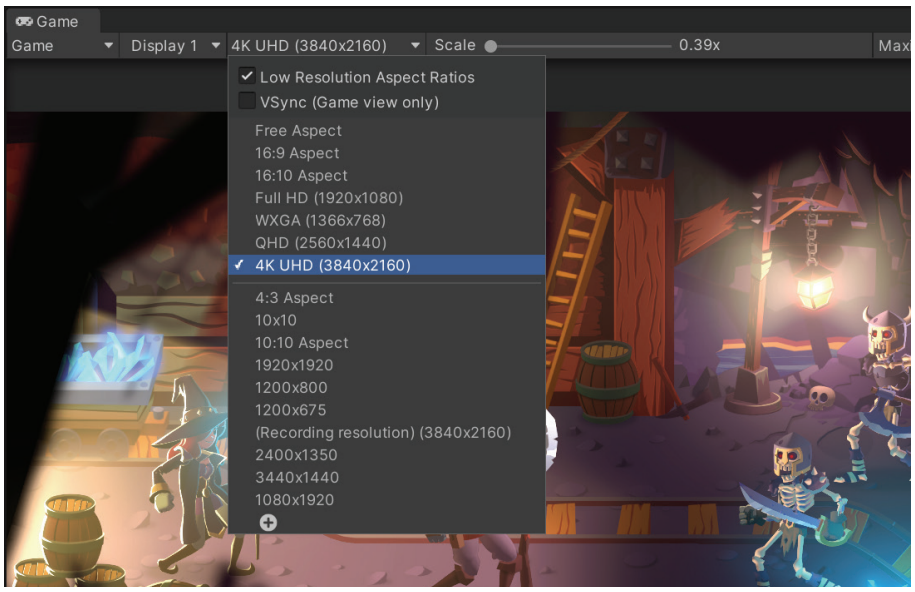
- ターゲットデバイスが取りうる最大解像度でアートを作成する
- 全アセットを 1 つの解像度で統一する。後に低スペックのデバイスのサポートが必要になった場合には、解像度を下げる

ピクセレーションやぼやけが発生し、ビジュアルクオリティーが低下するおそれがあるため、ラスター形式のアートは拡大しないようにします。常に、サポートされている最高の解像度から始めて、後にグラフィックスアプリケーションからエクスポートする際にはアートの解像度を下げるようにします。この方法の詳細については、このセクションの「DCC ツールと Unity を兼ねて行う作業」を参照してください。また、バリアントスプライトアトラスのスケールを使用してみることもできます。

有効なテクニックの 1 つに、必要なサイズの 2 倍でアートを描き、Unity にエクスポートするときに 50% に縮小するという方法があります。この手法を使うと、スプライトが滑らかで鮮明な見た目になるうえ、ブラシの線がガタガタすることがなくなります。ただし、アートを縮小するので、細かく描き込みすぎないようにします。これは、手書きアートで発生する小さく不徹底な部分を隠すのに効果的なテクニックです。もちろん、ゲームの見た目を手書き風のスタイルにしたい場合は、この方法はやめておきましょう。

解像度を選択した場合には、ゲームビューでアートの見た目をテストしてみて、ターゲットデバイスでどのように表示されるのかを確認するようにしましょう。

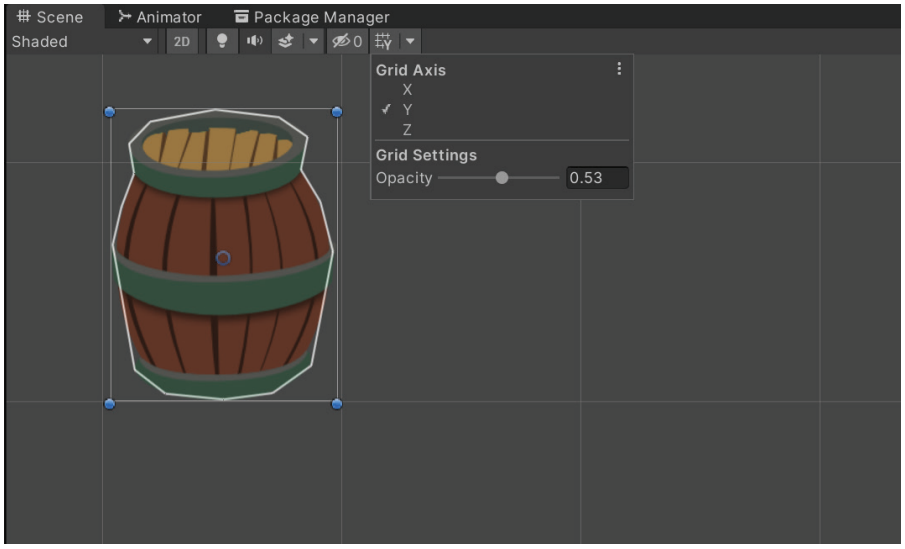
シンプルに保つため、スプライトの解像度は、透視投影ビューではなく平行投影ビューで計算します。可視化も簡単に行えるように、スケールを 1、1、1、Z 深度を 0 に設定します。




ゲームビューでは、さまざまな比率や解像度でゲームの見た目を簡単にプレビューできます。

Unity のグリッドと単位を駆使して、スプライトの配置や大きさを統一するようにし、カメラのズームやオブジェクトのサイズも計算しておきます。

Unity 単位は、シーンビューでグリッドにより可視化されています。1 Unity 単位は 1 メートルに等しいと考えてください。最初にベースサイズを設定して、ゲーム全体でサイズの一貫性を保つようにします。




シーンビューの Unity のグリッド

プレイヤーキャラクターのサイズから始めます。プレイヤーキャラクターの高さは、0.5 ～ 2 単位の間にします。タイルマップを使用している場合はタイルサイズを 1 単位として設定しましょう。

キャラクターやオブジェクトが、ゲーム内の他のビジュアル要素と比べて小さすぎたり大きすぎたりすると、Tramsform の値がおかしくなったり、物理演算で問題が発生してしまうことにつながります。

メインシーンにおけるベースオブジェクト（プレイヤーキャラクター、敵、収集物、ステージハザード）のサイズを確定したら、「Orthographic」に設定したカメラの「Size」プロパティを設定して、ズームレベルを選択します。そのあと、カメラの「Size」の値を確認します。この値を 2 倍にすると、カメラの縦のサイズが Unity 単位で得られるようになります。

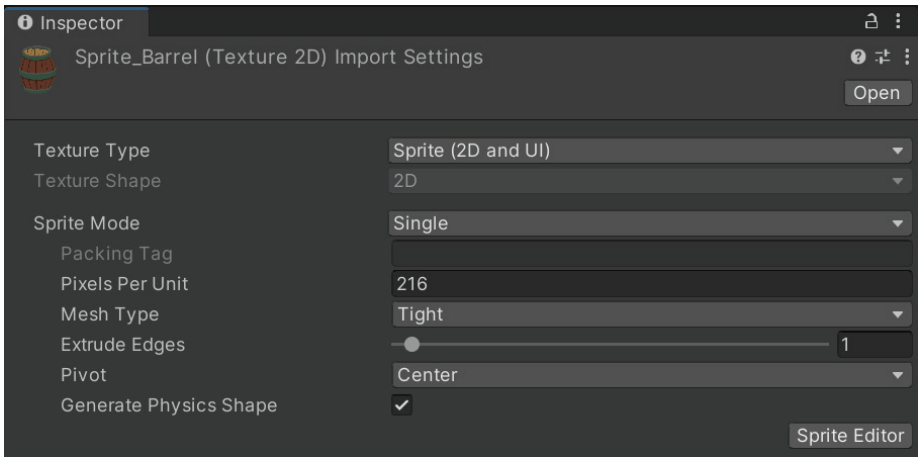

平行投影カメラのサイズは、中心から上部までの垂直方向のサイズを表す単位で表されます。

カメラの「Size」が 5 の場合、高さは 10 単位です。例えば、4K 解像度がターゲットの場合、画面（カメラ）の高さは 2160 ピクセルです。簡単な計算で、アートに必要な単位あたりのピクセル数（PPU）を求めることができます。

**2160 : 10 = 216**
**垂直方向の最大解像度 : (平行投影カメラのサイズ *2) = スプライトの PPU**

ゲーム内のすべてのスプライトは、ネイティブ 4K 解像度で美しく表示するために、約 216 PPU 必要です。

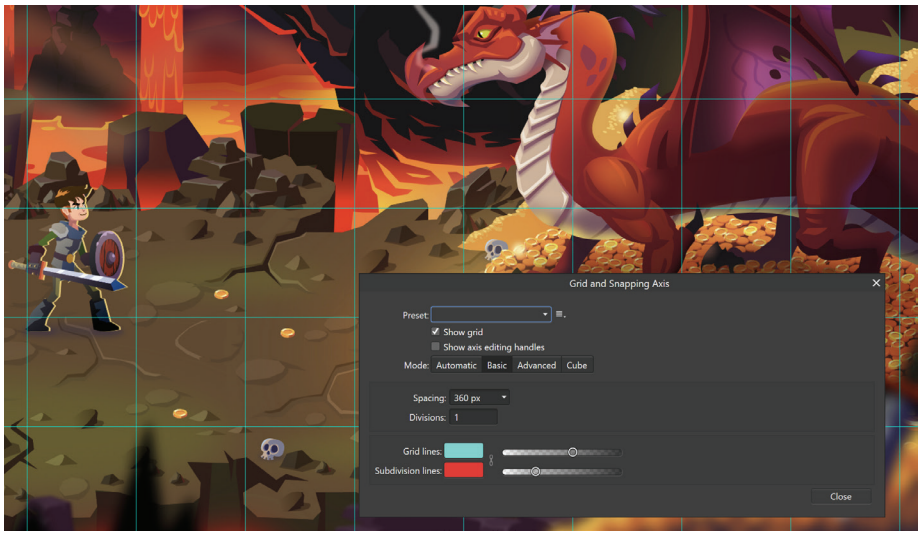
以上は単純な例でした。カメラをズームイン、ズームアウトする場合は、それも考えていく必要があります。平行投影ビューで最大ズーム割合をサイズ 3 に設定した場合、PPU は 360（2160 : (3 *2)）にする必要があります。




スプライトの PPU を設定する

スケルタルアニメーションを使用する場合は、スプライトの解像度を推奨 PPU よりも少し高くします。というのも、スケルトンのメッシュは時折極端な形で回転、伸縮、ワープします。その際に、見栄えが悪くならないよう、余裕を持った解像度が必要になります。カメラがズームインするときは、低解像度で行うように進めます。

**ヒント：**ゲームの PPU の値を設定するときは、参考になるように、グラフィックスソフトウェアでグリッドを適用しておくようにしましょう。タイルマップを使用する際はこのようにしておくと、タイルサイズをピクセル単位で取得できるうえ、グリッドにスナップできて便利です。




ゲームの PPU に合わせて Affinity Photo でグリッドを設定する

ただし、これらの数字に厳密に従う必要はありません。大抵、推奨値よりわずかに低いスプライト解像度を使用することでメモリを節約できます。背景スプライトなど、重要ではない要素は取り除きましょう。

また、デバイスでゲームをテストして、見栄えを確認するようにします。高解像度のアセットを使用しても、多くの場合、その良さは明確には現れません。代わりに、その描画にかかる時間とデバイスのメモリを、他の重要なゲーム要素（メインキャラクター、ビジュアルエフェクト、UI など）に割り当てましょう。モバイルデバイスでは、ゲームのサイズが重要です。そのため、スケールダウンできるアセットを確認して、メモリ制限を念頭に置くようにします。2D アセットの解像度の詳細については、こちらのブログ記事をご覧ください。

## スプライトアトラス

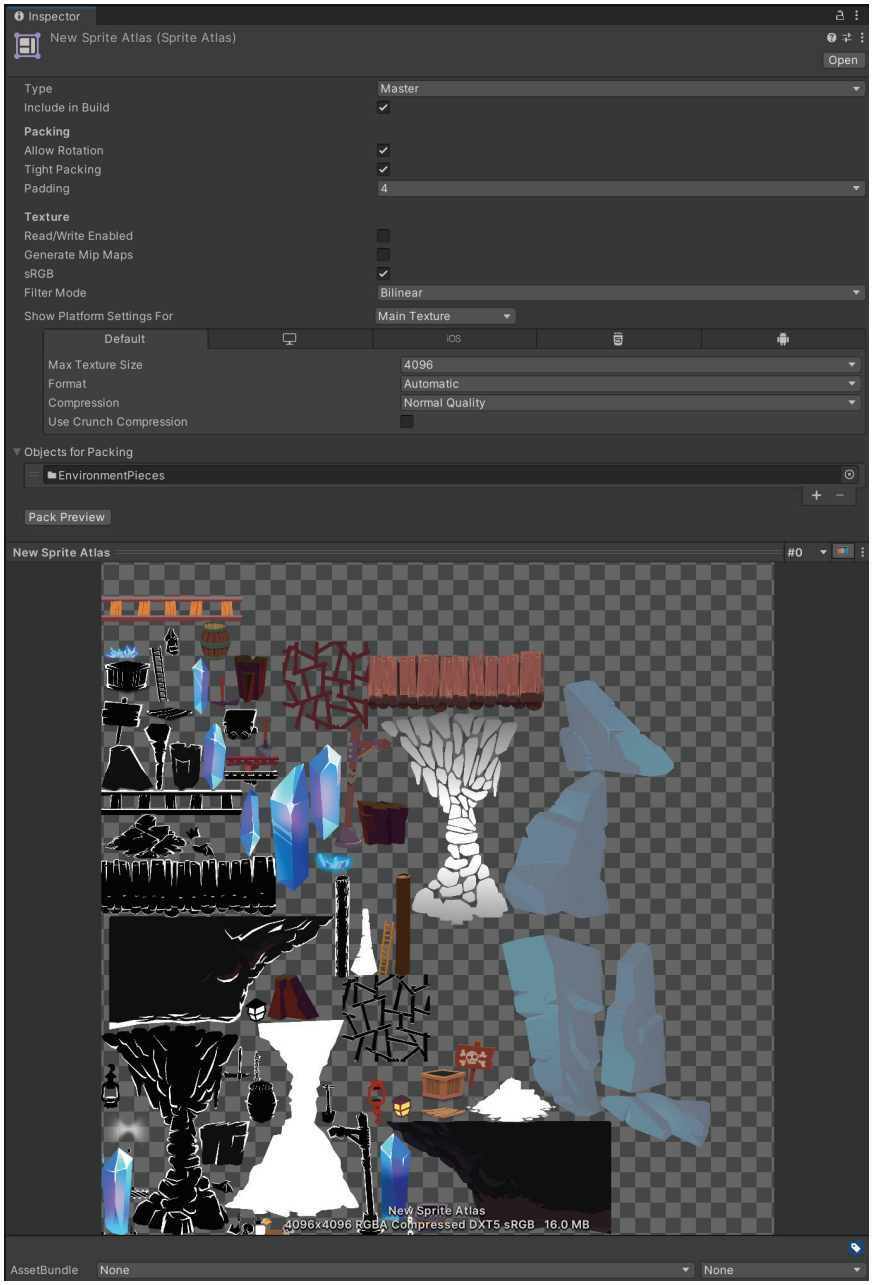
Unity のスプライトアトラス機能には主に 2 つのメリットがあります。

**複数のスプライトを 1 つのテクスチャにまとめてドローコールを削減：**異なるマテリアルまたはテクスチャが設定されたメッシュは、個別に画面上に描画されます。画面上に配置するスプライト（メッシュ）の数だけレンダリング時間が増え、ゲームのフレームレートが低下するおそれがあります。

スプライトアトラスを使用することで、複数のテクスチャを 1 つの合成されたテクスチャとして統合できます。Unity では、この 1 つのテクスチャを呼び出して 1 つのドローコールとすることができるので、複数のドローコールを行わなくて済みます。まとめられたテクスチャへのアクセスを一度にまとめて行うので、パフォーマンスのオーバーヘッドが小さくなります。

**デバイスタイプに応じたスプライトのスケールダウン：**Unity では、個々のスプライトではなくスプライトアトラス全体をスケールして、スケール済みのアトラスをさまざまなデバイスに割り当てると効率的です。

新しいスプライトアトラスを作成するには、「Project」ウィンドウ内で右クリック（またはツールバーの「Assets」メニューをクリック）し、**「Create」>「2D」>「Sprite Atlas」**の順に選択します。内容がわかるように名前を付けてください。次に、「Objects for Packing」オプションで、アトラスに含めるスプライトを選択するか、フォルダー全体を含めます。「Include in Build」オプションはデフォルトでオンになっています。これで、最初のスプライトアトラスができました。

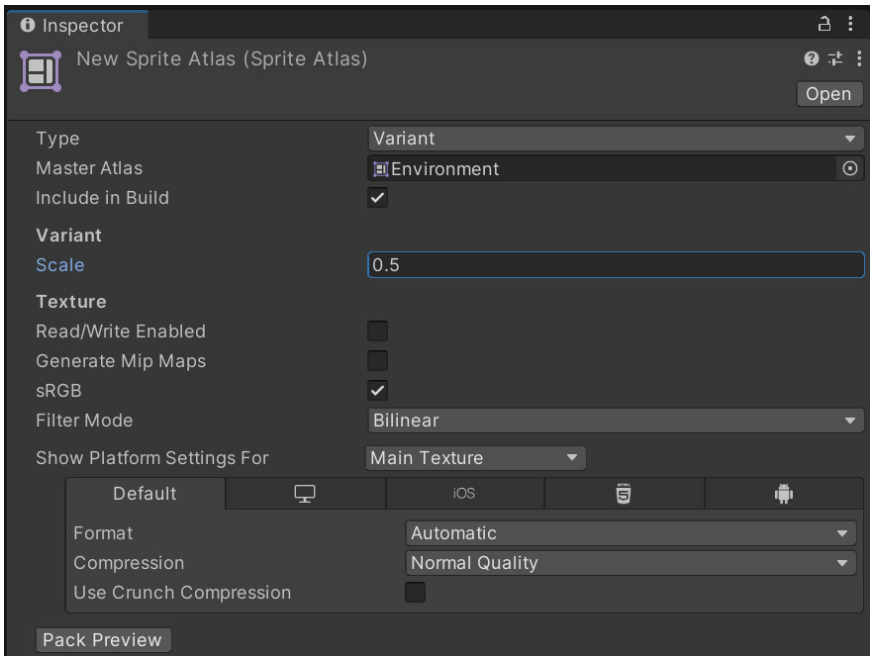



Unity のスプライトアトラス

## バリアントスプライトアトラス

特定のデバイス向けにスプライトをスケールダウンするには、バリアントスプライトアトラスを使用します。前述の説明に従ってマスタースプライトアトラスを作成したら、インスペクターで「Type」を「Variant」に変更します（以下の画像を参照）。

バリアントスプライトアトラスには、スプライトそのものは含まれていません。バリアントスプライトアトラスは、「Master Atlas」フィールドで選択できるマスタースプライトアトラスに依存しています。このアトラスとそれに含まれるすべてのスプライトのスケールを変更するには、「Scale Value」フィールドの値を 0.1 ～ 1 の範囲で選択します。




バリアントスプライトアトラスを設定する

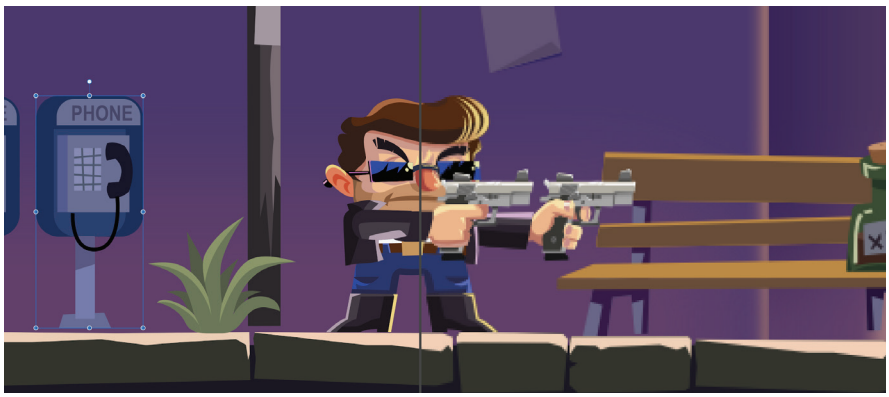
マススプライトアトラスとバリアントスプライトアトラスの両方を含むプロジェクトにおいて、両方のスプライトアトラス上で「Include in Build」オプションがオンの場合、共同のスプライトを使用したテクスチャでは、どちらかのスプライトアトラスから選択されることになります（Unity ドキュメントの「さまざまなスプライトアトラスのシナリオの解決」セクションのシナリオ 3 を参照）。

マスターアトラスではなくバリアントアトラスからスプライトテクスチャを自動的に読み込むには、「Include in Build」をバリアントアトラスでのみ有効にし、マスターアトラスでは無効にします。こうすればランタイムにマスターアトラスではなくバリアントスプライトアトラスがビルドに自動的に読み込まれるようになります。

## ベクターアプリケーションでの描画

「Adobe Illustrator」や Serif 製の「Affinity Designer」などのベクターベースのソフトウェアで作成したアートは、解像度の制約を受けることはありません。解像度設定を誤る懸念を持つこともなく、いつでもアートアセットのサイズが変更可能なので、こういった制約がないのは大きな利点となります。

ただし、ベクターベースのアートを PNG ファイルとして Unity へエクスポートすることを要するのはいぜんとして変わりません。エクスポートのワークフローについては、ラスターアートやピクセルアートの場合と同様です。




Affinity Designer で作成されたベクトルアート – この図の左側はアプリケーションでのベクトル表示を示し、右側はエクスポート後のピクセル表示を示しています。

## DCC ツールと Unity で兼行して作業を行う

アートに関しては、Unity での作業に進む前にやるべきことがたくさんあります。力を尽くしてキャラクター、要素、小道具、背景アートをできる限りに美しく、思い描いたとおりに作ったとなったら…、早く Unity に取り込みたいと思うのは自然なことです。

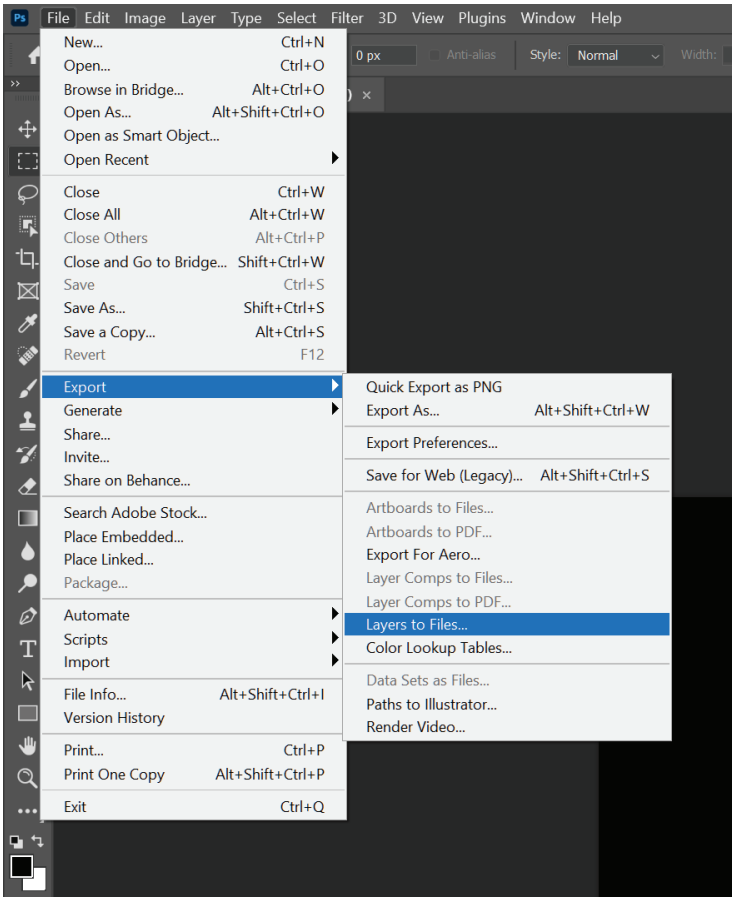
個々のレイヤーを 1 つずつ PNG ファイルとしてエクスポートする方法は、長い時間がかかるので避けます。代わりに、以下のヒントを参考にして、各種グラフィックスソフトウェアからゲームスプライトを効率的にエクスポートしてみてください。

### Adobe Photoshop からのエクスポート

さまざまな形式（PSD、BMP、JPEG、PDF、Targa、TIFF など）を使用してレイヤーを個別のファイルとしてエクスポートし、保存します。レイヤーは保存時に自動的に名前が付けられます。各種オプションを設定し、名称づけの制御をするようにします。

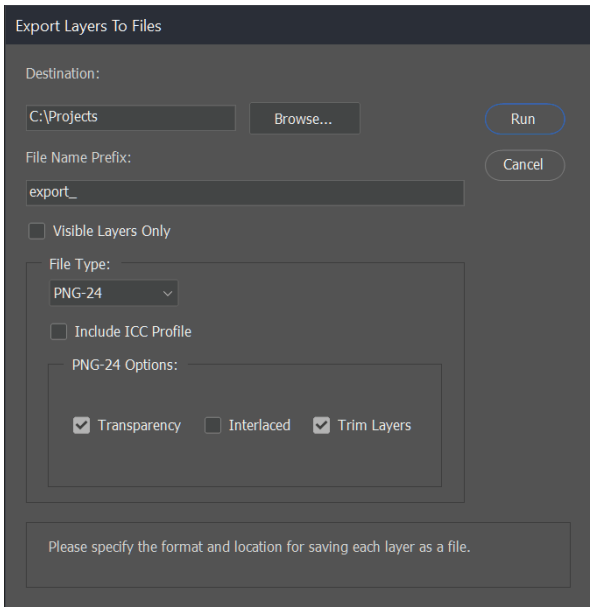
レイヤーをファイルとしてエクスポートするには、以下の手順を実行します。

1. メニューで、**「File」>「Export」>「Layers To Files」**の順に選択します。



Adobe Photoshop からのエクスポートの準備をする

2. 「Export Layers To Files」ダイアログボックスの「Destination」の下にある「Browse」をクリックして、エクスポートするファイルの保存先を選択します。デフォルトでは、生成されたファイルはソースファイルとしてサンプルフォルダーに保存されます。




「Export Layers to Files」ダイアログボックス

3. スプライトに共通のファイル名プレフィックスを付けたい場合は、プレフィックスをテキストフィールドに入力します（例：tile_wood01.png、tile_bricks01.png など）。

4. 「Layers」パネルで表示状態のレイヤーだけをエクスポートするには、「Visible Layers Only」を選択します。

5. 「File Type」は「PNG-24」を選択します。ICC プロファイルを含める必要はないので、「Include ICC Profile」はオフにします。

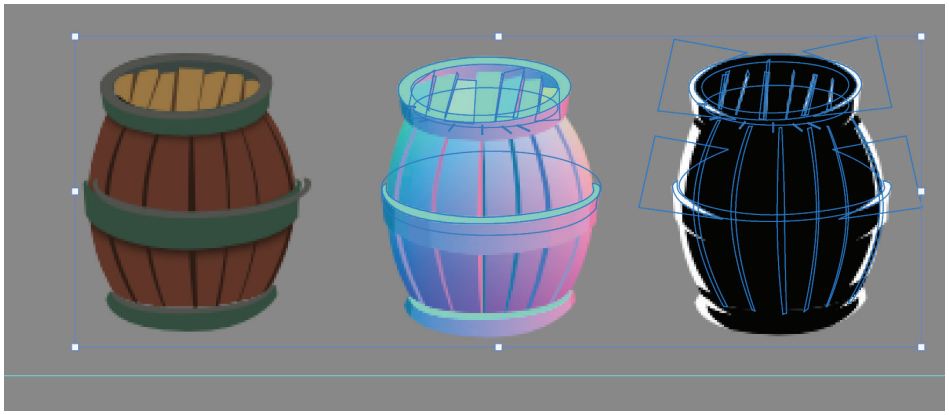
6. 「PNG-24 Options」セクションの「Transparency」オプションと「Trim Layers」オプションをオンにして、「Interlaced」をオフにします。

7. 「Run」をクリックすると、短い待ち時間の後、指定したフォルダーにスプライトがエクスポートされます。

8. 上記の手順に従うと、レイヤーの数を限定して、レイヤーをすばやくエクスポートできます。すべてのレイヤーが個別にエクスポートされることに注意してください。この方法では、レイヤーグループは考慮されません。そのため、複数のレイヤーで構成されるキャラクターやオブジェクトがある場合は、エクスポートの前にマージしておきます。

複数のレイヤーから成るスプライトを「Photoshop」からエクスポートする方法を見ていきましょう。必要な設定の量は増えますが、エクスポートした画像をより細やかに制御できます。

はじめに、スプライトを生成する元となるレイヤー（またはレイヤーグループ）の名前が必要です。例えば、以下の画像ですが、これは法線マップとマスクマップから構成されている樽のスプライトです。各スプライトは、個別のフォルダーに PNG ファイルとして保存されています。

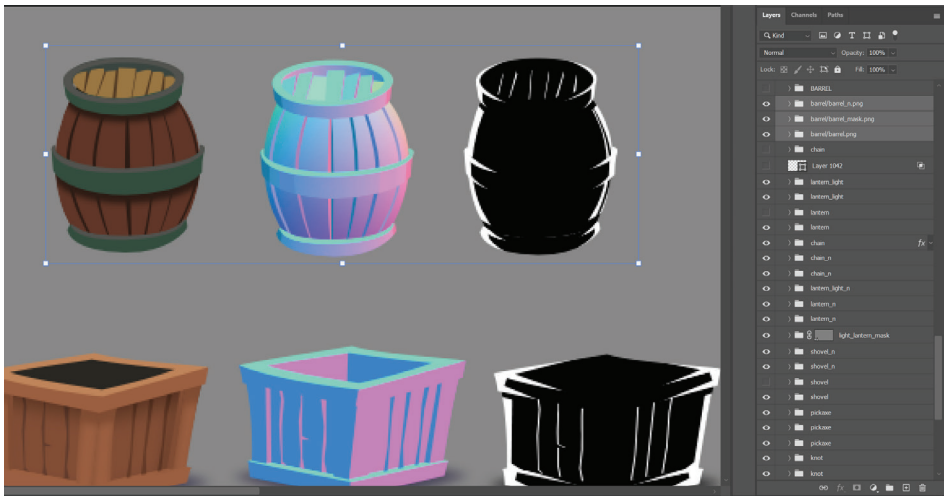


法線マップとマスクマップを使用したスプライト

メニューから**「File」>「Generate」>「Image Assets」**を選択します。Photoshop によって PSD ファイルと同じ場所にフォルダーが作成され、レイヤーまたはグループがすべてエクスポートされて PNG ファイルとして保存されます。PSD ファイルがまだ保存されていない場合は、デスクトップにフォルダーが作成されるようになっています。

この方法の利点は、エクスポートのマークが付いているレイヤーやフォルダーを変更した場合、画像アセットがバックグラウンドで自動的にエクスポートされることです。

レイヤー名の前に名前とスラッシュを追加することで、各スプライトのフォルダーを指定します。このケースでは、樽のスプライト 3 つすべてを 1 つのフォルダーにまとめるために、レイヤー名の前に「barrel/」を追加します。



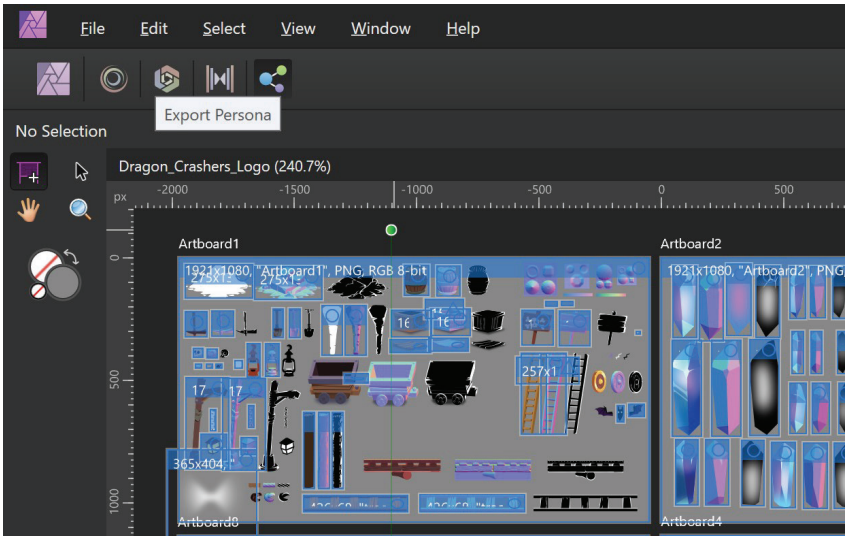
Photoshop から同じスプライトの複数のレイヤーをエクスポートする

スプライトのサイズを変更する場合は、レイヤー名の前に、寸法またはスケールを割合で指定して追加します。例えば、「50% barrel/barrel.png」、「80 × 160 barrel/barrel.png」などとなります。寸法を指定する際では、Photoshop ではピクセルがデフォルトの単位になります。

### Affinity Photo または Affinity Designer からのエクスポート

Affinity Photo と Affinity Designer には、どちらも「Export Persona」というエクスポート用のモードがあります。これを使用する手順を見ていきましょう。

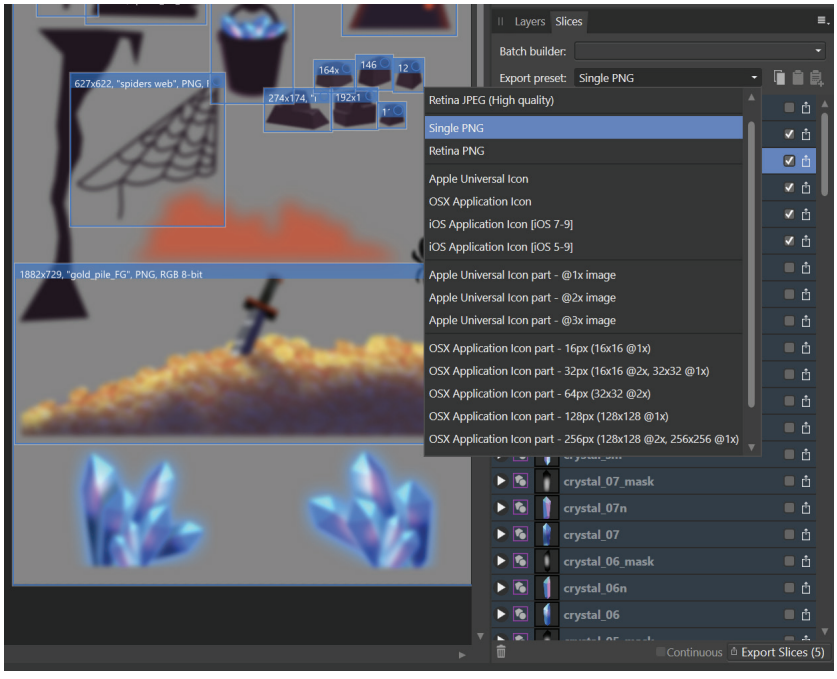
下の画像のように、アプリケーションのツールバーの左上にあるアイコンをクリックし、「Export Persona」に切り替えます。




Affinity Photo の「Export Persona」

スプライトをエクスポートするには、レイヤーまたはレイヤーグループからスライスを作成します。右側の「Layers」タブに切り替え、エクスポートするレイヤーを選択し、「Create Slice」ボタンをクリックします。

「Slices」タブに切り替え、エクスポートするスライスを選択します。次の画像のように、すべてのスライスで「Export preset」が「Single PNG」に設定されていることを確認します。

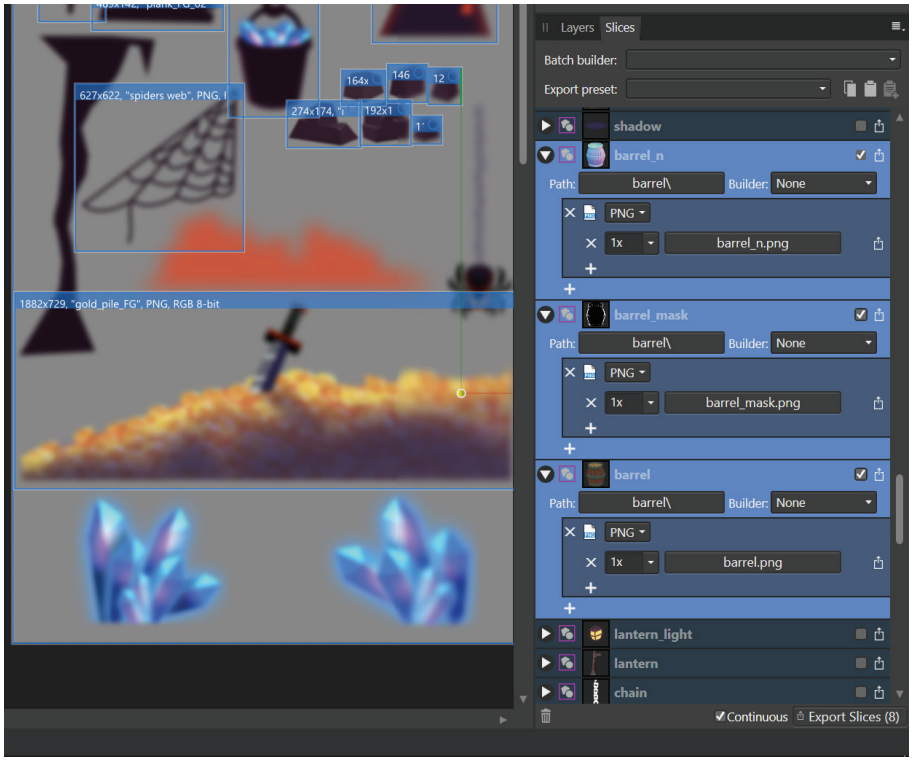



Affinity Photo で「Export preset」を「Single PNG」に設定する

「Export Slice」をクリックし、画面の指示に従ってエクスポート先のフォルダーを選択します。スプライトは、Unity プロジェクトの Assets フォルダーに直接エクスポートすると、時間を節約できます。

エクスポート先のフォルダーを選択したら、「Continuous」オプションが利用可能になります。このオプションから、レイヤーを変更したときにスライスをエクスポートできます。

エクスポートしたスプライトをより精緻に管理するためには、スライス名の左にある矢印アイコンをクリックして、各スプライスのエクスポートオプションを手動で設定します。

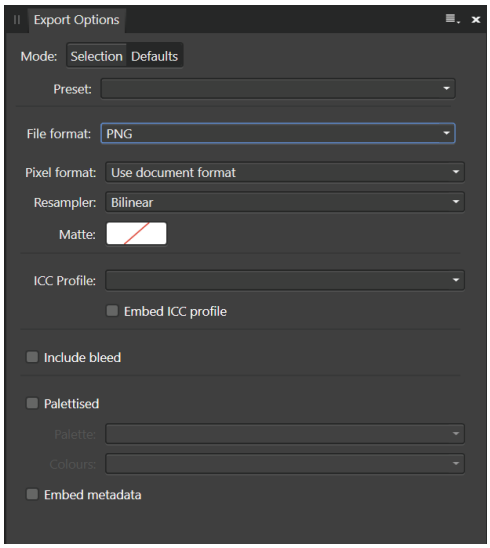



スプライトのエクスポートオプションを設定する

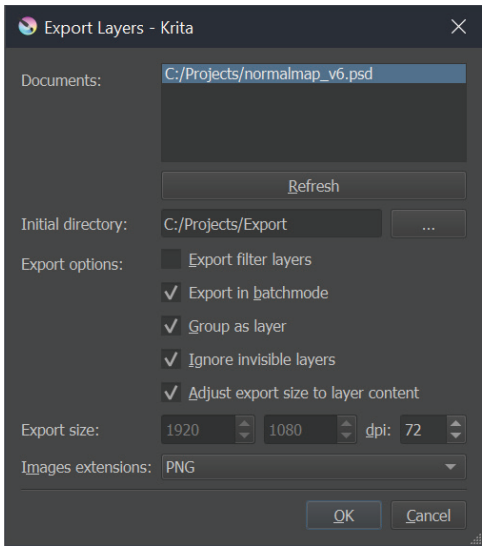
「Path」セクションでスプライトのサブフォルダーを選択します。この例では、樽のスプライトはすべて「barrel」サブフォルダーに入っています。ファイル形式は「PNG」で、解像度のスケーリングは「1x」に設定されています。これはスケーリングを行わず、スプライトが元のサイズのままエクスポートされるということです。「+」ボタンをクリックし、一度に複数のサイズを出力することもできます。

マルチプラットフォーム対応のゲームの場合は、他のデバイスでプロジェクトをビルドする際に使用できるスプライトのサイズを追加で選択しておきましょう。例えば、「Retina」のスプライトを取得する際はスケーリングで「2x」を選択します。

展開されたウィンドウの下部にある別の「+」ボタンでは、PNG スプライトとともに JPG ファイルもエクスポートしたい場合に、ファイル形式を追加できます。




Affinity アプリケーションの「Export Options」ウィンドウ




Krita からスプライトをエクスポートする

### Krita からのエクスポート

まず、必ずドキュメントを保存するようにしてください。**そして、「Tools」>「Scripts」>「Export Layers」を順に選択します。**ポップアップウィンドウで、「Initial directory」を選択します。これがスプライトファイルがエクスポートされる場所です。

「Export options」で、「Adjust export size to layer content」をオンにします。これをオンにすると、レイヤーがサイズに合わせて切り抜かれ、空ピクセルのパディングがなくなります。オプション設定はいくつかあり、以下の通りとなっています。

— Group as layer：個々のレイヤーではなく、グループでエクスポートします。スプライトが複数のレイヤーで構成されている場合に適しています。このエクスポーターでは、階層内の最上位のグループのみがチェックされるため、スプライトを別のグループにすることはできません。

— Ignore invisible layers：一部のスプライトを非表示にして、エクスポート対象から除外する場合に便利です。

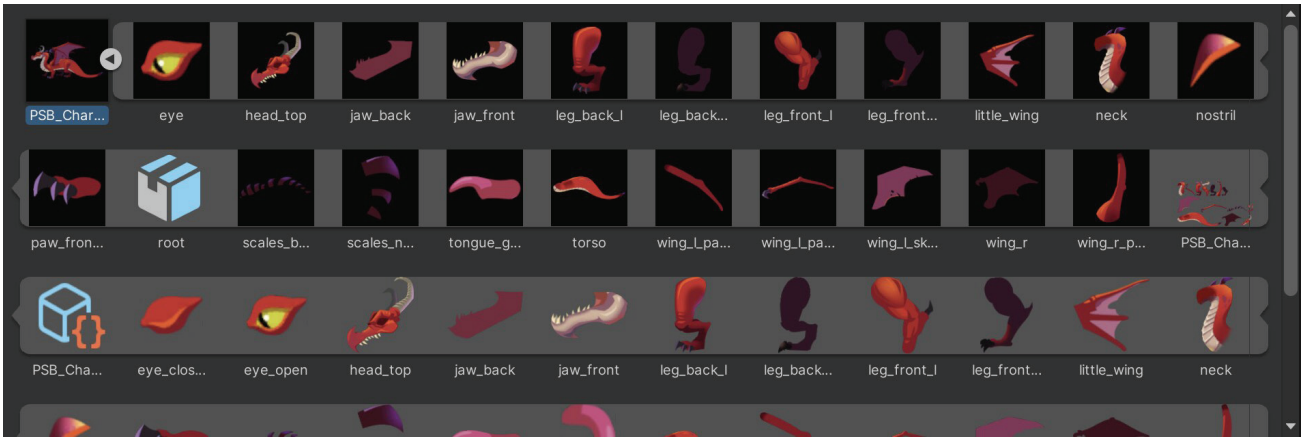
「Image extensions」として「PNG」を選択し、「OK」をクリックします。選択したフォルダーにスプライトがエクスポートされます。

## Unity への Photoshop PSD ファイルのインポート

スプライトは PNG ファイル形式でエクスポートするのが最も便利ですが、PSD ファイルを Unity に直接インポートすることもできます。デフォルトでは、これを行うと PSD のレイヤーがフラット化され、1 つの画像になります。

この方法は、背景アートのスプライトに適しています。レイヤーを編集、ペイントします。ファイルを保存すると、変更内容が即時にエディターにも反映されます。以降は PSD ソースファイルが Unity プロジェクト内にそのまま保存されます。このファイルはソース管理に含めることができます。

Unity の 2D PSD Importer パッケージには、レイヤーを別個のスプライトとしてインポートできるオプションがあります。このパッケージは 2D アニメーション用に設計されていますが、1 つの PSD ファイルに含まれる通常のスプライトを複数インポートする際に使用することもできます。コマ送りアニメーションを作成することもできます。このパッケージの使用方法の詳細については、2D アニメーションに関するセクションかこちらのブログ投稿をご覧ください。

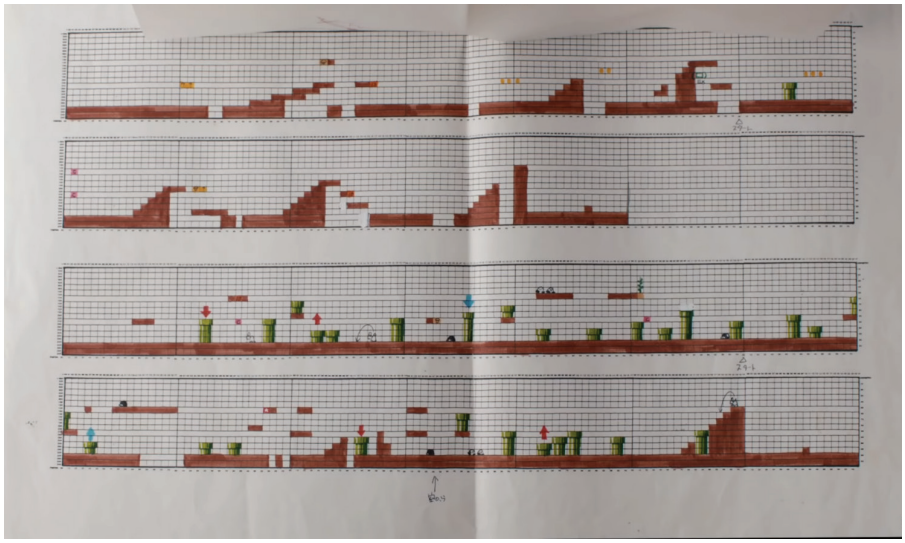



Unity にインポートされたマルチレイヤーの Photoshop ファイル

# ステージデザイン

ステージのプロトタイプを作成する手法は多数あるため、自分のクリエイティブスタイルやワークフローに最適な手法を選んでください。

シンプルで手っ取り早く、持ち運びが容易でコストもかからないため、通常は紙にアイデアを書き出す方法が好まれると思います。1985 年の『スーパーマリオ』のステージデザイン方法をご覧ください。



『スーパーマリオ』のレベルデザイン（出典：Nintendo YouTube チャンネル）

これは長年にわたって最適なステージデザイン方法として使われてきていて、現在でも極めて有効です。しかし、Unity にスケッチすることで、より早い段階で、より活発にアイデアを検証することができます。利用できるツールを見てみましょう。

## ホワイトボックス作成

White Boxing（ホワイトボックス作成）とは、よく知られている 3D ステージデザインの用語で、平板な白いキューブを配置し、ステージデザインのアイデアを手早く検証してみるものです。白いボックスを置くのは、余分なディテールを排してゲームフローのアイデアを図示するためです。

この手法は 2D 図形やその他のプロトタイプ作成ツールを使用して、2D グラフィックスにも応用できます。

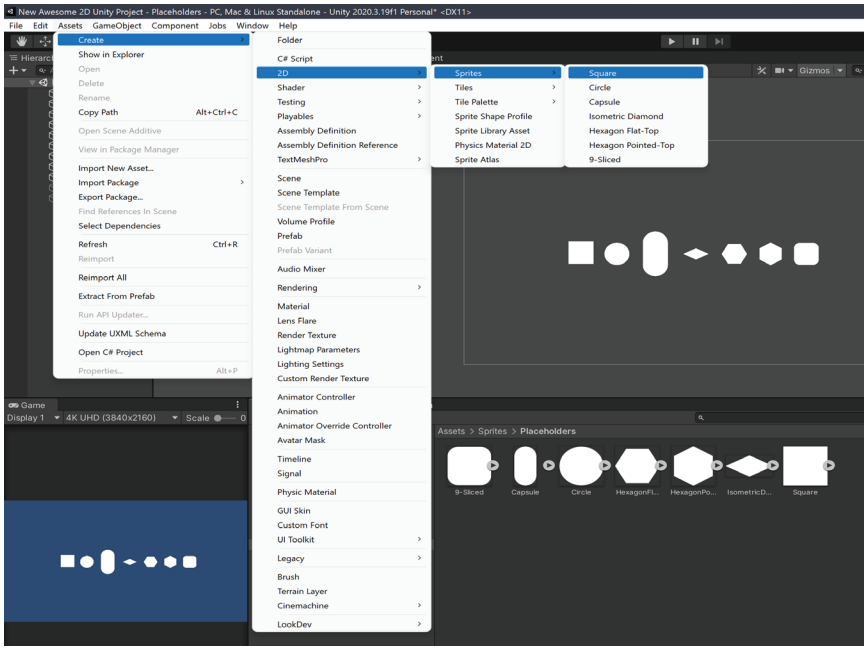
ホワイトボックス作成では、地面、敵、ステージハザード、拾得物など、プレイヤーと相互作用する要素のみに焦点を当てる必要があります。こうした要素は、まず非常に簡単な形で作成してから、色分けして区別しやすくします。例えば、敵やその他の危険要素は赤にして、拾得物は緑、スイッチ類は青などとするとよいでしょう。

まず、地面や壁、プレイヤーがジャンプする足場などの衝突 / インタラクションレイヤーから着手します。

### 基本的なスプライトを使用したホワイトボックス作成

はじめから Unity で使用可能なシンプルな形状のデフォルト 2D アセットから始めます。

「Project」ウィンドウ内で右クリック（またはツールバーの「Assets」メニューをクリック）し、**「Assets」>「Create」>「2D」>「Sprites」**を順に選択します。すべての形状に定義済みの衝突形状が含まれているため、スプライトシェイプに追加する必要があるのは Polygon Collider 2D コンポーネントのみです。これにより、スプライトシェイプが重力や衝突の影響を受けるようになります。




Sprite Creator で利用できるすべてのスプライト

「Project」ウィンドウで開かれているフォルダーに、選択したスプライトが作成されます。

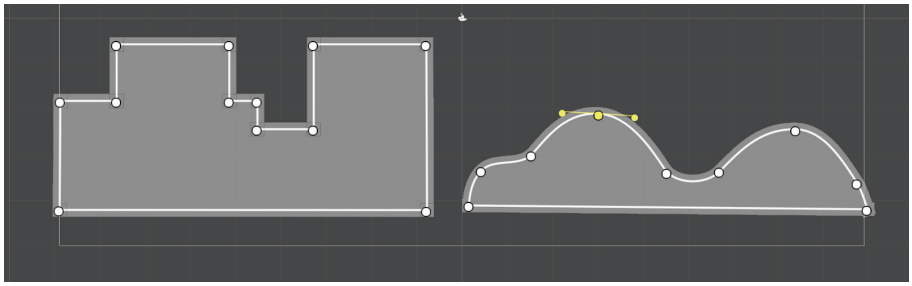
用途に応じて異なる形状を使用します。例えば、プレイヤーにはカプセル、拾得物には円、敵には六角形、地面や壁には正方形を使用します。スプライトレンダラーの色プロパティを使用して、カテゴリごとにスプライトを色分けします。

スプライトで 2D 物理演算の衝突を使用する場合は、2D Rigidbody とともに、オブジェクトの全体的な形状と一致する Collider 2D コンポーネントを追加します。Rigidbody が重力、力、衝突に反応するように、「Body Type」に「Dynamic」を選択します。

正確に配置するために、Ctrl を押しながら移動させることで、スプライトをグリッドにスナップします。この方法では、0.25 単位で移動させることができます。

## 2D Sprite Shape

2D Sprite Shape は、パスを作成し、そのパスに沿ってタイル状のスプライトを作成できるツールです。このパスは、グラフィックスソフトウェアで一般的なペンツールと機能がよく似ています。シーン内で直接編集できるベジェ曲線なので、オプションで削除したり、タイル状テクスチャで埋めたりすることもできます。

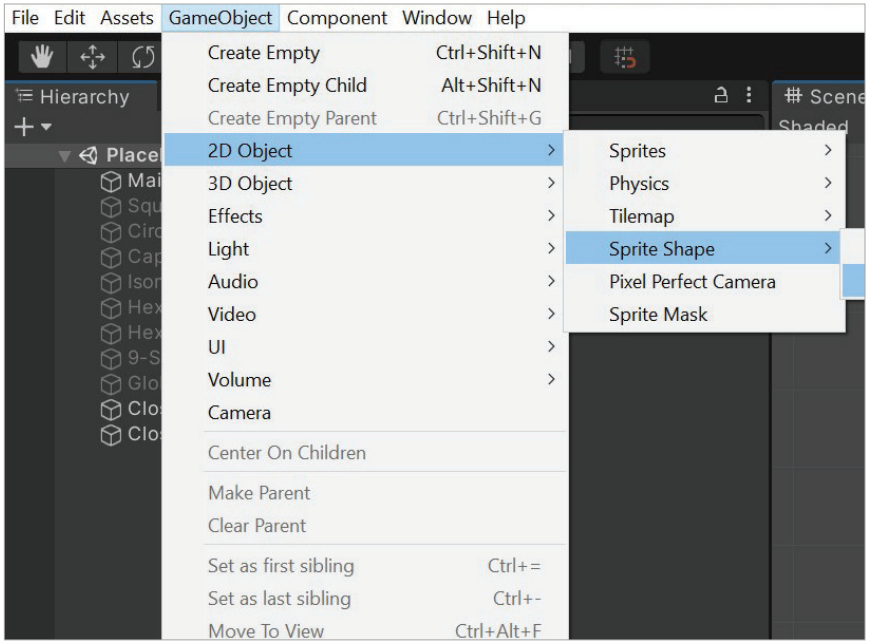


Linear Sprite Shape オブジェクトと Continuous Sprite Shape オブジェクト

Sprite Shape は Unity の使いやすくプロトタイピングを行える機能です。これで新規スプライトシェイプを効率よく、作成、編集することができます。

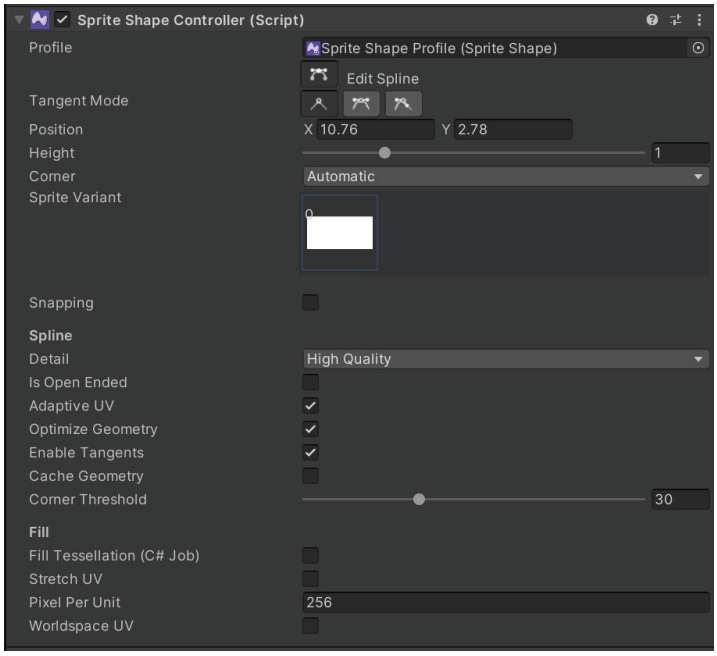
2D テンプレートを使用して 2D プロジェクトを作成する場合、デフォルトでは Sprite Shape がインストールされます。もしくは、Package Manager を使用してインストールすることも可能です。

新しいスプライトシェイプを作成する場合は、「GameObject」メニューをクリックし、**「2D Object」>「Sprite Shape」**を選択してから、「Open Shape」か「Closed Shape」を選択します。




スプライトシェイプを作成する

スプライトシェイプに変更を加える場合は、選択してからインスペクターの「Edit」ボタンをクリックします。




「Inspector」ウィンドウでスプライトシェイプを変更する

スプライトシェイプの一部を左クリックすると、ポイントを追加したり、delete キーを押してポイントを削除したりできます。ポイントを選択し、「Tangent Mode」の 3 つのボタンのうちいずれかを選択することで、モードを変更できます。

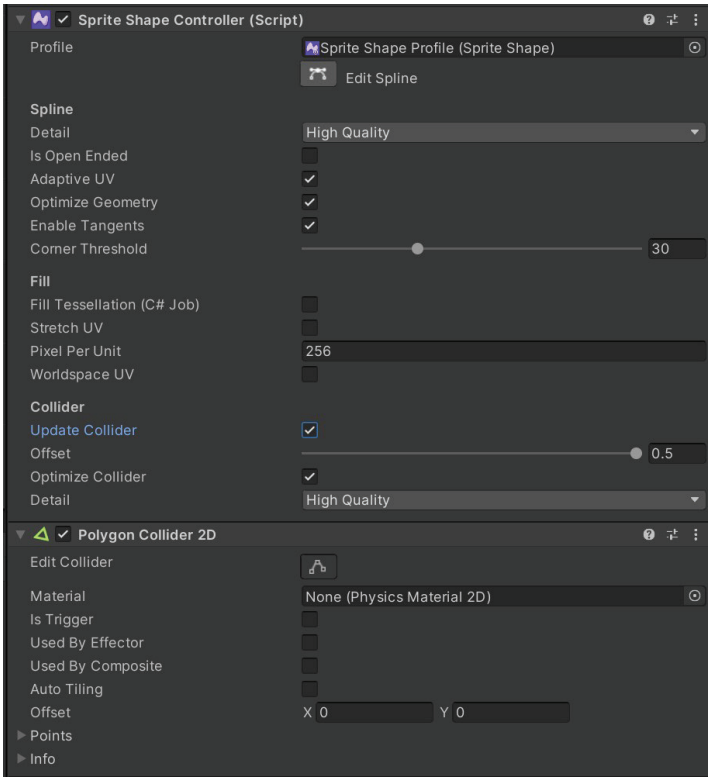
— **Linear：**曲線を形成せず、ポイントとポイントを結ぶ直線を引きます

— **Continuous：**反対方向を向いたハンドル付きで、ポイントを中心に曲線を形成します

— **Broken：**個々に移動させることができるハンドル付きで、ポイントを中心に曲線を形成します

ポイントを中心とするコーナーの表示方法も選択できます。スナップツールも、ポイントをグリッドにスナップする便利なオプションです。

物理演算用に Polygon Collider 2D コンポーネントを追加します。プロトタイプ作成が目的であれば、デフォルトのプロパティで十分です。



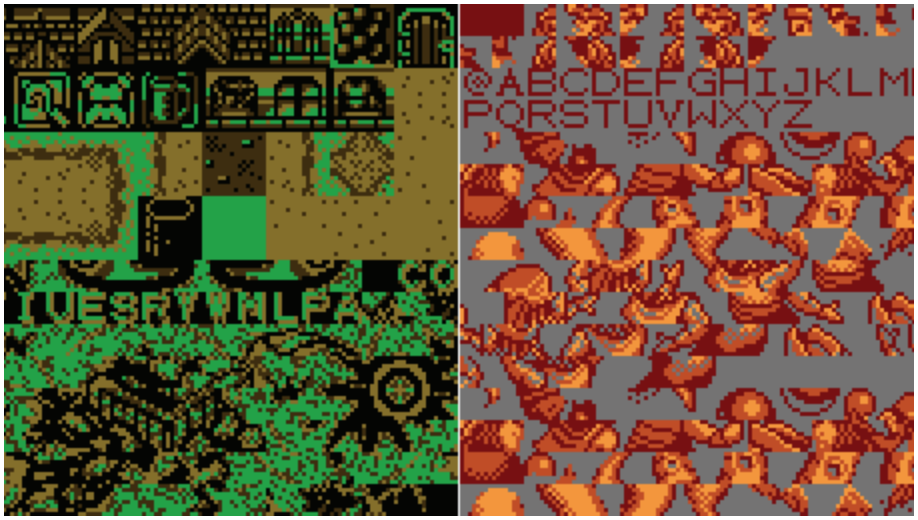

Polygon Collider 2D コンポーネントを追加する

こうした手順に即すことで、相互に作用するスプライトシェイプを作成し、インスペクターで簡単に修正できます。

## タイルマップ

タイルマップ機能は、即時のプロトタイピングに適しています。タイルマップを使用すると、グリッドに配置されるタイルと呼ばれる小さなスプライトを使用して、ゲーム世界を作ることができます。1 つの大きな画像としてゲーム世界をレイアウトするのではなく、画像を、ステージ全体で繰り返し描画するレンガのような塊に分割します。

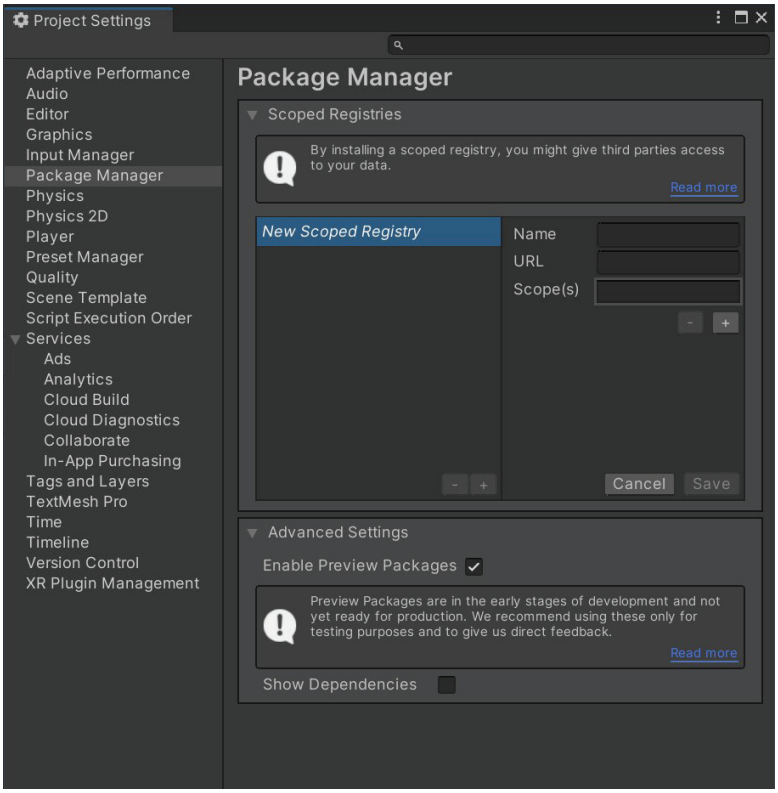
タイルマップを使用すると、画面上に表示されていないタイルは無効化できるため、メモリや CPU の負荷低減につながります。ブラシツールを使うと、グリッド上のタイルを効率的にペイントできます。いくつかのペイントルールを使用して、スクリプト化することも可能です。タイルマップには衝突判定が自動生成される機能があるため、効率的にテストや編集を行えます。



VRAM に読み込まれる、レトロな 8 ビットゲームの背景タイルとスプライトタイルの視覚的表示。コンソールでは、背景とスプライトが別々のページに整理して保持されます。

2D Tilemap Editor は、2D プロジェクトテンプレートを使用してインストールするか、「Package Manager」からインストールします。

2D Tilemap Extras パッケージをインストールすると、再利用可能な 2D およびタイルマップエディタースクリプトを独自のプロジェクトで使えるようになります。これらのスクリプトは、カスタムのブラシやタイルを作成する際のベースにもなります。「Project Settings」（**「Edit」 > 「Project Settings」**）で「Preview Packages」を有効にしてパッケージを取得します。「Package Manager」で、「Advanced Settings」ボックスにある「Preview Packages」オプションをオンにします。

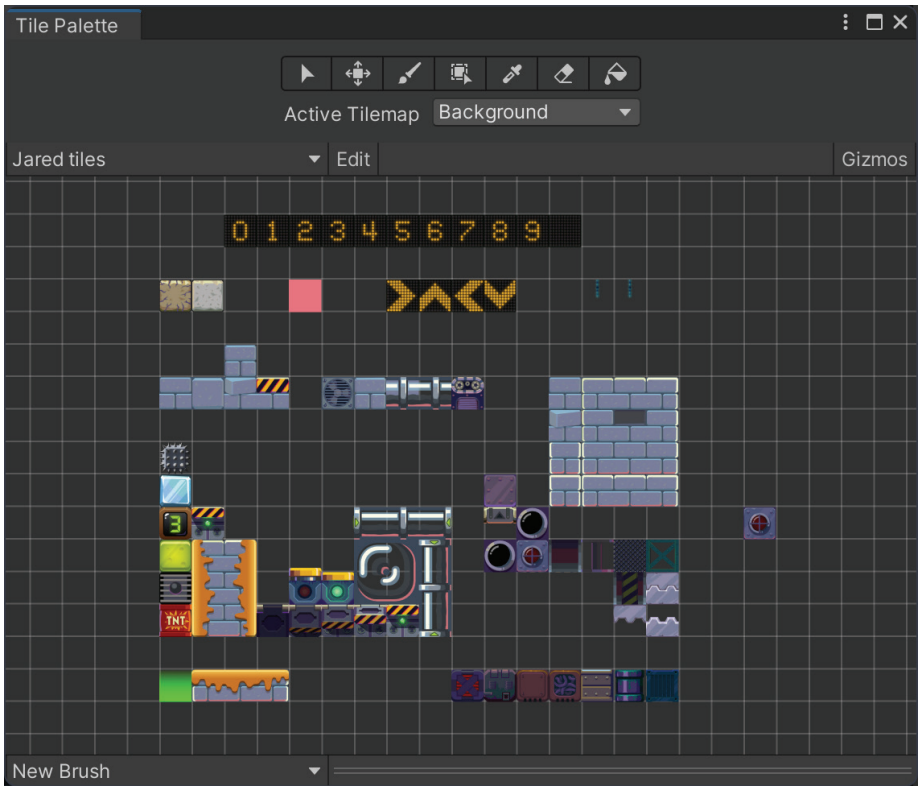



Package Manager でプレビューパッケージを表示する

2D Tilemap Extras とともにサンプルをインストールし、パッケージに付随するスクリプトを使用しているサンプルを確認します。Unity のドキュメントで、パッケージに含まれているスクリプタブルブラシやスクリプタブルタイルの詳細を確認できます。

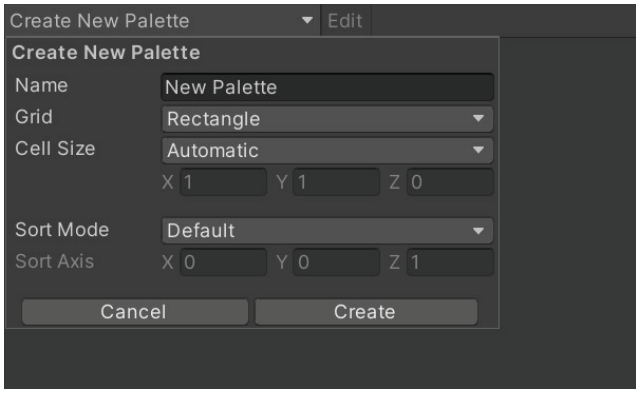
両方のパッケージをインストールしたら、**「Window」>「2D」>「Tile Palette」**から、「Tile Palette」ウィンドウを開きます。

このウィンドウで、タイルマップのペイントと編集に役立つタイルやツールを利用できます。



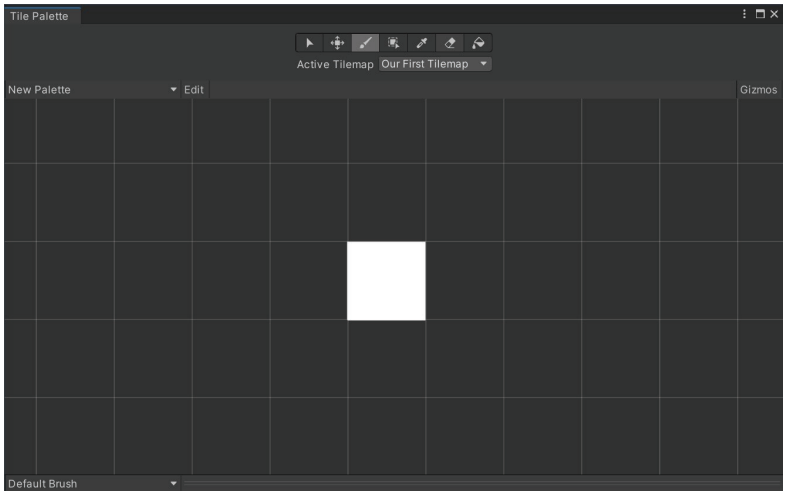

パレットが読み込まれた「Tile Palette」ウィンドウ

「Create New Palette」ボタンをクリックして、新しいパレットを作成します。オプションを設定できるドロップダウンウィンドウが表示されます。パレットに名前を付け、オプションを設定してから、「Create」をクリックし、選択したフォルダーに保存します。



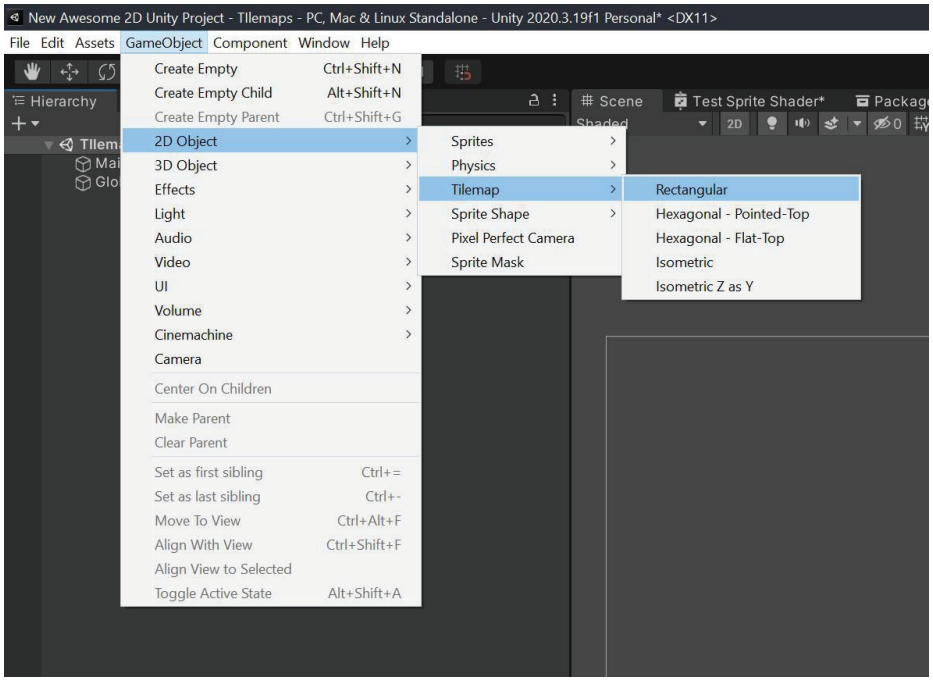

新しいパレットを作成する

続いてタイルをパレットに追加します。スプライト、テクスチャ、またはタイルアセットを「Palette」ウィンドウにドラッグします。前に作成した簡単なスプライトシェイプのいずれかを使用して、ウィンドウの空白部分に配置します。アセットを作成するファイルを選択します。パレットのグリッドにスプライトが表示されます。




スプライトをパレットにドラッグすると、新しいタイルが作成されます。

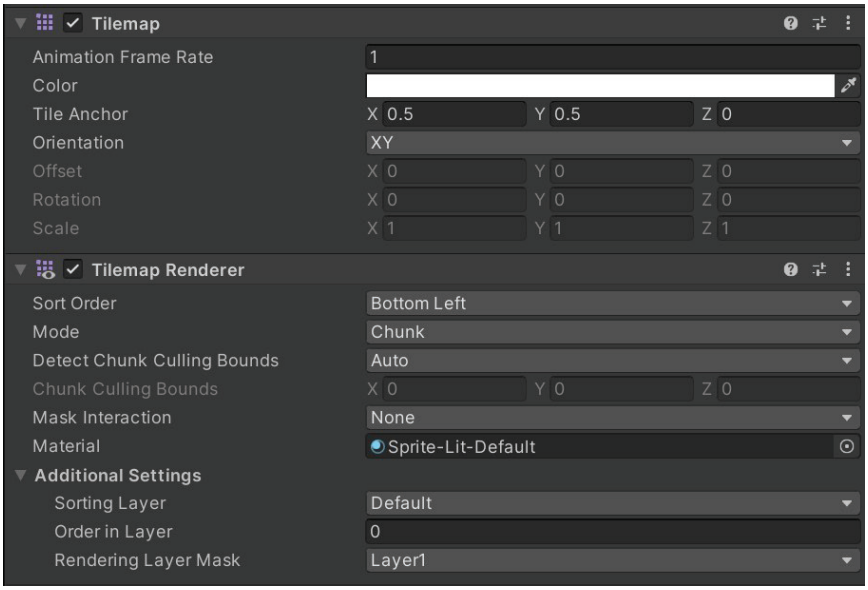
続いて、タイルをペイントするためのタイルマップを作成します。メニュー項目の**「Gameobject」>「2D Object」>「Tilemap」>「Rectangular」**を順にクリックします。




タイルマップを作成する

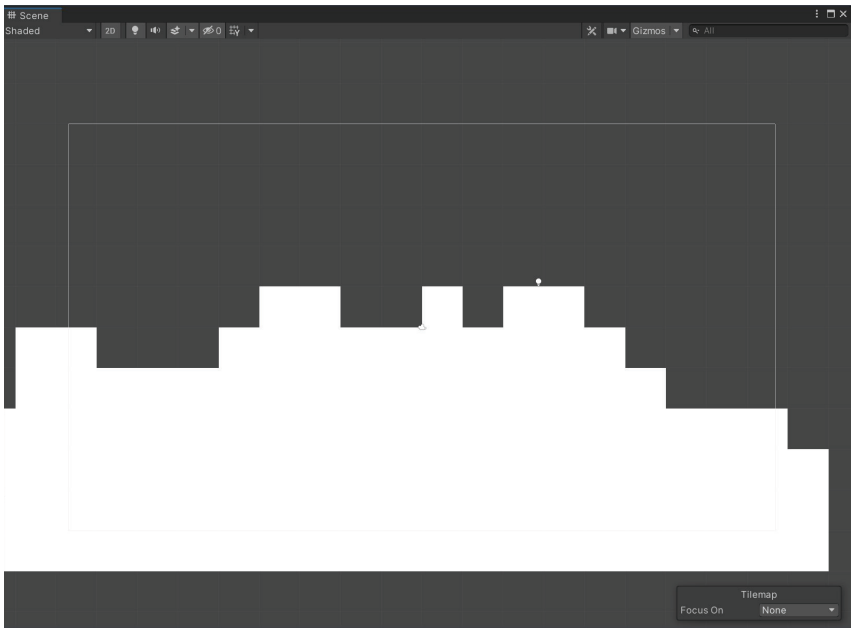
配下にグリッドとタイルマップが作成された状態になりました。タイルマップの名前をわかりやすいものに変更します。グリッドには複数のタイルマップを保持できます。1 つのシーンに、セルのサイズが異なる複数のグリッドを収めることができます。ここでは、「Grid」コンポーネントの設定はすべてデフォルトのままにしておきます。

「Tilemap」ゲームオブジェクトには、「Tilemap」と「Tilemap Renderer」の 2 つのコンポーネントがあります。これらの設定はそのままにしておきます。もし必要であるなら、レイヤー構造に合わせて「Sorting Layer」設定を変更しておきましょう。



Tilemap コンポーネントと Tilemap Renderer コンポーネント

タイルマップとタイルがパレット上に揃えば、ペイントの開始準備は完了です。「Palette」ウィンドウのツールバーでブラシツールをクリックし、ペイントするタイルを選択します。

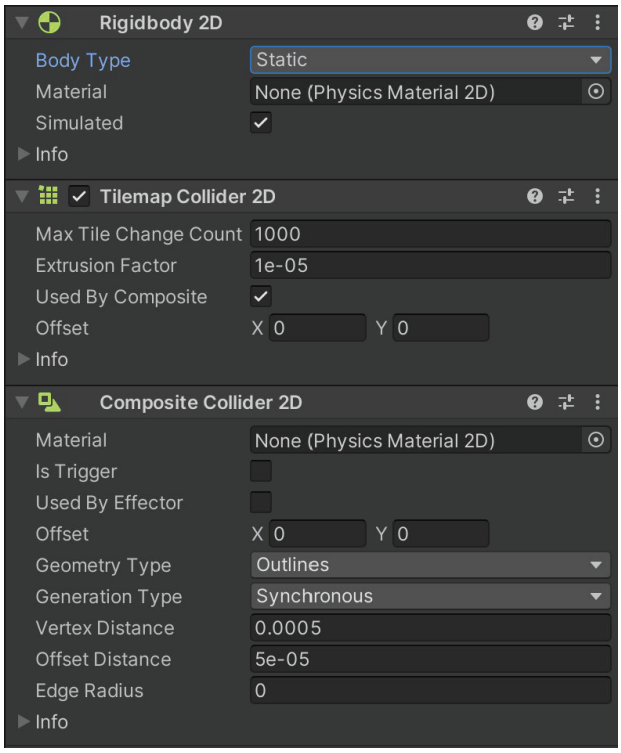


ペイントされたプレースホルダータイルを含むタイルマップ

次に、ステージのプロトタイプを作成できます。物理演算用に「Tilemap Collider 2D」をタイルマップに追加します。これにより、タイルアセットで設定したコライダーのタイプに基づき、コライダーが各タイルに追加されます。

「Composite Collider 2D」を追加し、コライダーを 1 つに統合します。ジオメトリを「Outlines」に設定し、衝突時の動作をスムーズにします。タイルは、個々のタイルではなく、連続する地形のように動作します。この設定においては、パフォーマンスもわずかですが向上しています。しかし、ゲームでのランタイムにタイルを、追加もしくは削除する予定がある場合は、各タイルのコライダーは保持したままが良いでしょう。

「Collider 2D」コンポーネントで、忘れずに「Used by Composite」オプションをオンにして、落下しないように「Rigidbody 2D」の「Body Type」を「Static」に設定してください。




Tilemap ゲームオブジェクトの Collider 2D コンポーネント

それでは、多彩なパレットツールを見ていきましょう（かっこ内はキーボードショートカットです）。


タイルパレットのツール

- **Selection（S）**：特定のタイルをクリックして選択するか、ドラッグして矩形領域内のタイルを選択します。
- **Move（M）**：選択したタイルを移動します。
- **Brush（B）**：選択したタイルとブラシを使用して、アクティブなタイルマップ（「Active Tilemap」ドロップダウンから選択）にペイントします。
- **Fill Selection（U）**：ドラッグし、矩形領域内を選択したタイルで塗りつぶします。
- **Tile Sampler（I）**：タイルマップからタイルを選択し、ペイント用にアクティブに設定します。
- **Eraser（D）**：タイルマップからタイルを削除します。
- **Fill（G）**：タイルで領域を塗りつぶします（この領域は、他のタイルで縁取りされている必要があります）。

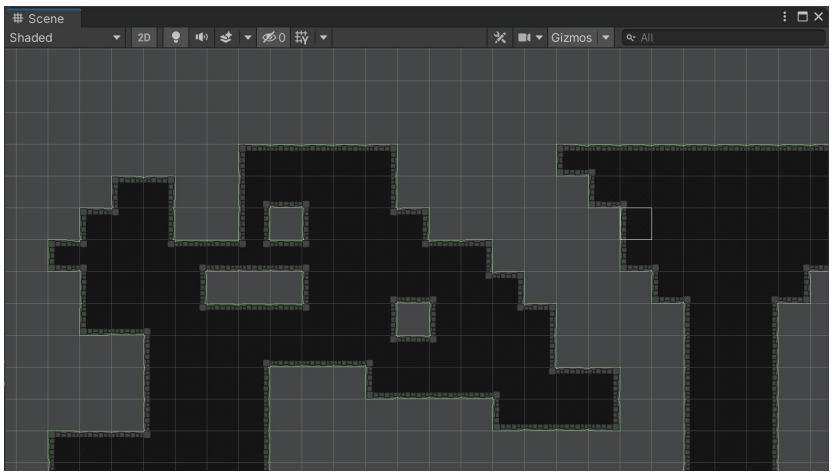
こういったツールだけでもタイルを効率的にペイントしたり編集したりすることができます。また、Tilemap Extras のアセットには、Rule タイルなどの便利なスクリプトもあります。

## Rule タイル

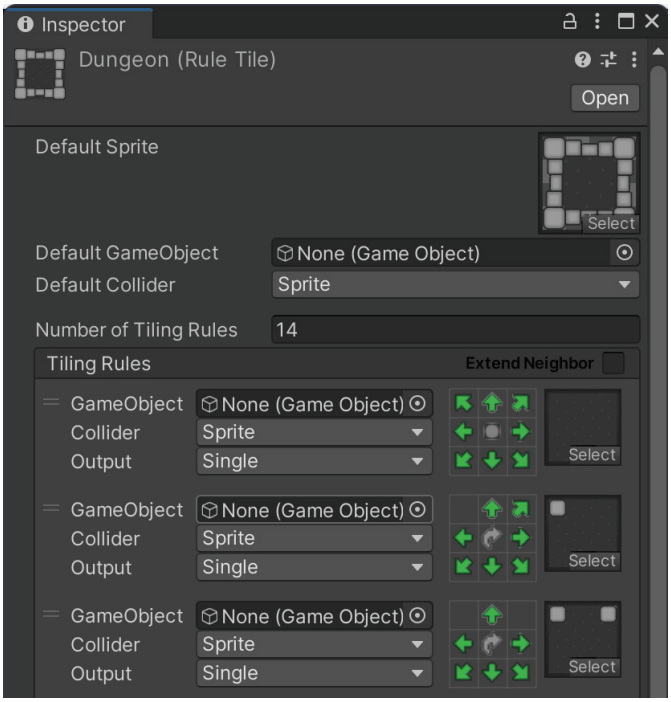
これは、囲んでいるものを識別し、応じた適切な画像を選択するようスクリプト化されたタイルです。例えば、上部に草、下部に影がある地面タイルなどです。

2D Tilemap Extras パッケージには、「Dungeon Rule Tile」というサンプルが含まれています（**「Assets」>「Samples」>「2D Tilemaps Extras」>「バージョン番号」>「Dungeon Rule Tile」>「Tiles」**）。これをタイルパレットにドラッグすると、ペイントが開始されます。

Rule タイルは、隣接タイルの位置と適切な境界線の位置を自動的に識別します。自動で適切なスプライトを選択してくれるため、境界線をペイントするために別のタイルを選択する必要はありません。




タイルマップをペイントする




インスペクターの Rule Tile アセット

Rule タイルのインスペクターでは、隣接タイルに基づいて、どのスプライトを選択するかを指定するためのルールのリストが配されています。各ルールの右側には、マトリックスとスプライトがあります。この緑色の矢印が指している位置の、すべてのタイル側へ向けてスプライトが適用されることになります。

Rule タイル機能の詳細については、こちらをご覧ください。

これは Rule タイルの使い方のほんの一例です。このサンプルのスプライトを別のものへ換えたり、ゲームのニーズに合わせて新しい Rule タイルを作成したりすることもできます。

コミュニティや Asset Store から、即時使用可能なタイルマップテンプレートを入手できます。選りすぐりのものから、数点挙げると Pandaroo の TileMap Auto Rule や、Devil’ s Work.shop の 2D PixelArt – Isometric Blocks、Cainos の Pixel Art Platformer と Pixel Art Top Down などがあります。

タイルマップのベストプラクティスについては、こちらの動画で確認できます。タイルマップのパフォーマンスの最適化については、こちらの記事で確認できます。

## スプライトシェイプのステージデザイン

グリッドまたはタイルベースのデザインは、特にピクセルアートを使用する場合など、レトロな 2D のビジュアルスタイルによくなじみます。横スクロールの水平プラットフォーマーや、トップダウンの RPG、建物や城などの非自然物の形状でも、大変よく合います。

グリッドベースのデザインであれば、直に経路検索やステージ作成を行うことができます。グリッドにコンストレインすると、経路検索がシンプルになります。つまり、ステージデザインをしている時に、一定の距離感を持った状態を保つことが容易になります。例えば、キャラクターのジャンプの高さが 3 単位の場合は、足場を配置する場所を計画しやすくなり、プレイヤーもジャンプできるかどうかをすぐに予測できるようになります。

個々のタイルをアニメーション化させることも可能です。例としては、落ちる滝や燃えているたいまつ、回るファンなどが加えられます。タイルマップを使用して傾斜を作ることもできます。ですが、その場合 45 度や 26.5 度といった角度の直線状のサーフェスになります。

より森林などの自然の雰囲気で、ブロックが少ないスタイルにする場合は、2D Sprite Shape を使用します。この設定では、グリッドや特定の角度の制約を受けることはありません。曲線を使用して、どのような形状でも作成できます。地形、丘、草原、滑らかなサーフェスはどれもスプライトシェイプに適しており、くっきりとしたピクセルアートのスタイルよりも、洗練された見栄えをシーンに与えることができます。

スナップツールをオンにすると、角度や曲線を簡単に設定して、要素を正確に配置できます。

左側の『*Skul: The Hero Slayer*』（SouthPAW Games 制作）はタイルマップベースのゲームで、右側の『*Oddmar*』（Mobge LTD 制作）はスプラインベースのゲームです。

タイルマップまたはスプライトシェイプで作成したステージや要素は、後で編集するにしても簡単です。ビジュアルスタイルに合わせて、両方を併せてゲーム内で使用できます。

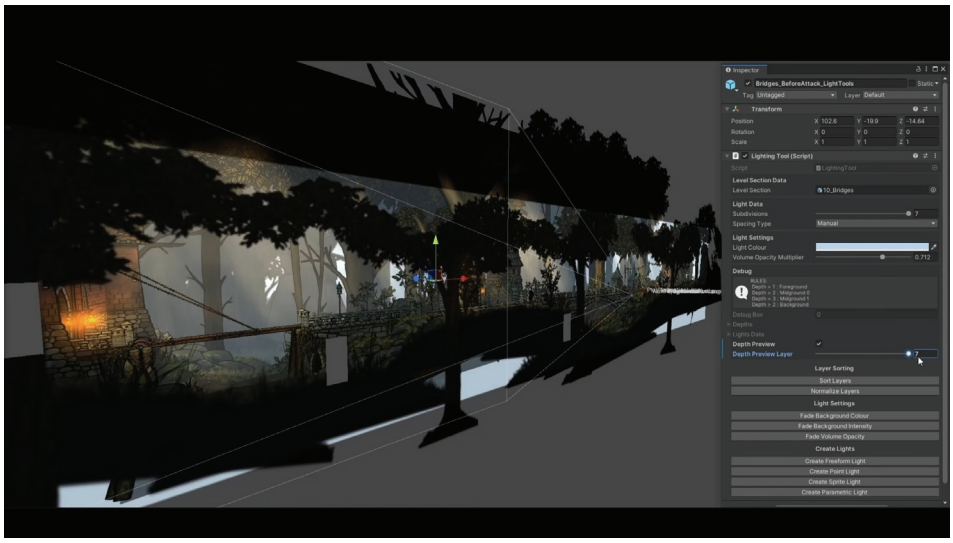
タイルマップとスプライトシェイプはどちらも、ランタイムで変化を与えることで刺激的なこれまでにないゲームプレイを生み出すことができます。例えば、タイルマップの方は壊せるものにすることもできますし、スプライトシェイプはモーフィングさせることができます。可能性は無限大なのです。このオプションの詳細については、Tilemap API と SpriteShape API に関するドキュメントをご覧ください。

## 平行投影カメラと透視投影カメラ

大半の 2D ゲームにおいては、カメラは平行投影モードへ設定することを推奨しています。1 次元の線は平行で、作業は 2 次元でのみ行うため、このモードが 2D プロジェクトでは基本となっています。平行投影モードは、ピクセルアートのゲームやパズル、2D アイソメトリック、トップダウン、プラットフォーマーなど、あらゆるスタイルに適しています。

2D ゲームは実際には奥行きがありませんが、パララックス（視差）効果を使用して奥行きがあるように見せかけることはできます。パララックス効果は、カメラが移動したときに、背景と前景の複数のレイヤーを異なる速度でスクロールします。目的は、人間による奥行きの知覚方法を模倣し、遠くのオブジェクトの方が、近くのオブジェクトよりもゆっくり動いて見えるようにすることです。

2D では、カメラの移動速度の割合に合わせてレイヤーを動かすスクリプトを使用して、これを表現するのが一般的です。各パララックスレイヤーの速度を調整するためには、いくつかの手動設定が必要です。




『*Tails of Iron*』（Oddbug Studio 制作）の透視投影カメラの設定。Unity でこのゲームのライティングがどのように製作されているのか見てみましょう。

平行投影モードは、どのような 2D ゲームのスタイルにも適しています。では、それでも透視投影モードを使用する理由はどこにあるのでしょうか。

理由の 1 つは、このモードでは、その名のとおり透視投影の視点を使用するため、標準機能としてパララックス効果を利用できるからです。パララックスレイヤーのスクロールを処理するために、スクリプトを使用する必要はありません。その代わりに、Z 軸を基準にカメラから離れた位置に要素を配置し、それを拡大することで、カメラからの相対的な距離に基づく視点に対応します。

パララックス効果が適用されたスプライトは、カメラに合わせて毎フレーム移動するわけではないため、背景のオブジェクトは、静的としてマークしてバッチ処理することができます。シーンに 3D オブジェクトを追加して奥行きを強調することもできます。

スクリプト化されたパララックス効果を含む平行投影モードを使用するには、すべてのレイヤーのスクロール速度を手動で設定する必要があります。レイヤーのスクロール速度が不適切な場合、奥行きの効果が破綻する場合があります。また、パララックスレベルを編集するには、追加のコーディングが必要です。

例えば、複数の開始点があるシーンを使用して、*メトロイドヴァニア*のようなスタイルのゲームを作っているとします。その場合は、パララックス要素を編集した場所とは違う X 軸上の位置からカメラが開始される可能性があるため、パララックス効果が破綻することがあります。

設定が済んだら、透視投影カメラがパララックス効果には有効です。唯一のマイナス面として挙げられるのは、スプライトを Z 軸上で移動させる際に、スケーリングする必要があるという点です。また、この手法では、シーンを小さなシーンに分割し、動的に世界をロードできるルームシステムなど、追加ロードを行うこともできます。

選択したカメラは、アートの作成プロセスには影響しませんが、ステージの設定方法はそれによって決まります。正しく設定されていれば、平行投影でも透視投影でも、パララックスがプレイヤーには同じように表示されます。

まとめると、平行投影モードはほとんどのケースに適しています。ただし、高度なスクロールやパララックスを使用する場合は除きます。その場合は透視投影モードを使用します。

『*Dragon Crashers*』の平行投影モードのカメラ

## 2D と 3D の融合

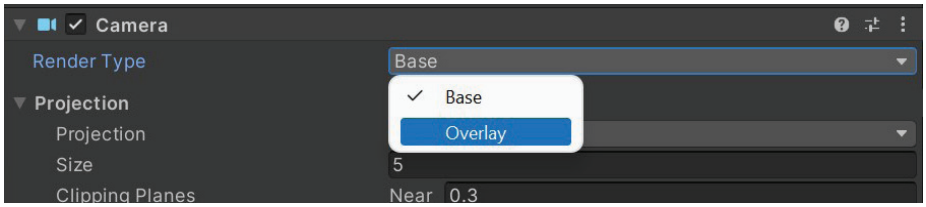
プロジェクトでビルトインレンダーパイプラインやユニバーサルレンダーパイプライン（URP）を使用しているのであれば、同じシーンで 2D と 3D の要素を簡単に合わせることができます。2D レンダリングでは、ソートレイヤーとソーティンググループを使用することで、ゲーム要素のレンダリングの順序を定義します。ソーティンググループコンポーネントを 3D ゲームオブジェクトに追加すると、同じゲーム内で 3D オブジェクトと 2D オブジェクトを簡単に統合することができます。共通の物理演算システム（ゲームにどのような制作手法が適しているかに応じて 2D 物理演算と 3D 物理演算のいずれか）を使用してそれらのオブジェクトを相互作用させることができるほか、URP のカメラスタッキング機能を使用して 2D と 3D のライティングシステムを融合させることもできます。


2D ゲームの背景に 3D アセットを導入すると、ゲームプレイを損なわずにシーンに奥行きを加えることができます。上段と中段の画像は、近日発売予定の *2D 横視点スクロールの探索型ゲーム『Aeterna Noctis』*（Aeternum Game Studios S.L. 制作）です。下段の画像は、*『Last Night』*（Odd Tales 制作）です。

## Camera Stacking（カメラスタッキング）

カメラスタッキングを使用すると、カメラの出力を重ね合わせて結合し、1 つの最終的な画像を生成できます。カメラスタッキングでは、2D、3D、UI オブジェクトを結合することができます。

重ね合わせを行うためには、シーン内に少なくとも 2 つのカメラが必要です。まず、レンダリングする画像のベースカメラを選択し、それ以外のカメラは「Overlay」に設定する必要があります。

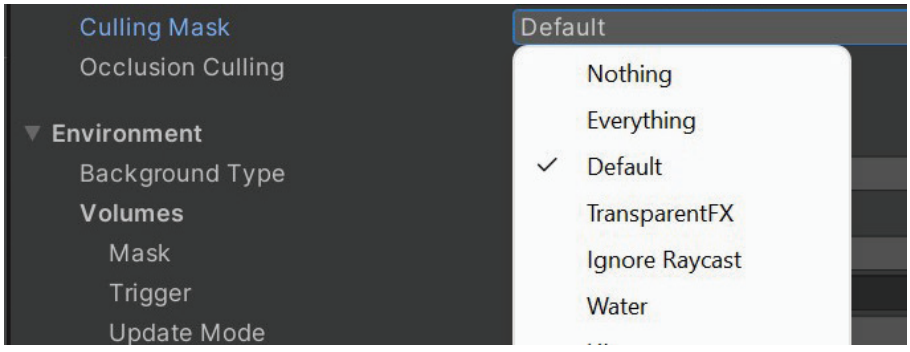



カメラスタッキングの設定をする

ベースカメラは、どのカメラオーバーレイを、どういった順序で重ねるのかを認識する必要があります。そのためには、「Stack」オプションに移動し、「+」ボタンを使用してオーバーレイのカメラを選択します。必要に応じてリスト内のオーバーレイカメラを上下にドラッグすることで、レンダリングの順序を変更します。


オーバーレイとしてレンダリングするカメラを選択する

レイヤーを使用して、各カメラによってレンダリングするオブジェクトを設定します。デフォルトでは、カメラはすべてのレイヤーをレンダリングしますが、Camera コンポーネントの「Culling Mask」オプションでカメラによってレンダリングするレイヤーを選択することで、それを変更できます。




カメラの「Culling Mask」オプションでレンダリングするレイヤーを選択する

ここで、カメラでレンダリングするゲームオブジェクトを選択し、前に設定したものに合わせてレイヤーを変更します。これにより、3D 背景や 2D の足場レイヤーなど、オブジェクトが分類され、別々のカメラでレンダリングされるようになります。

## 2D 用 Cinemachine

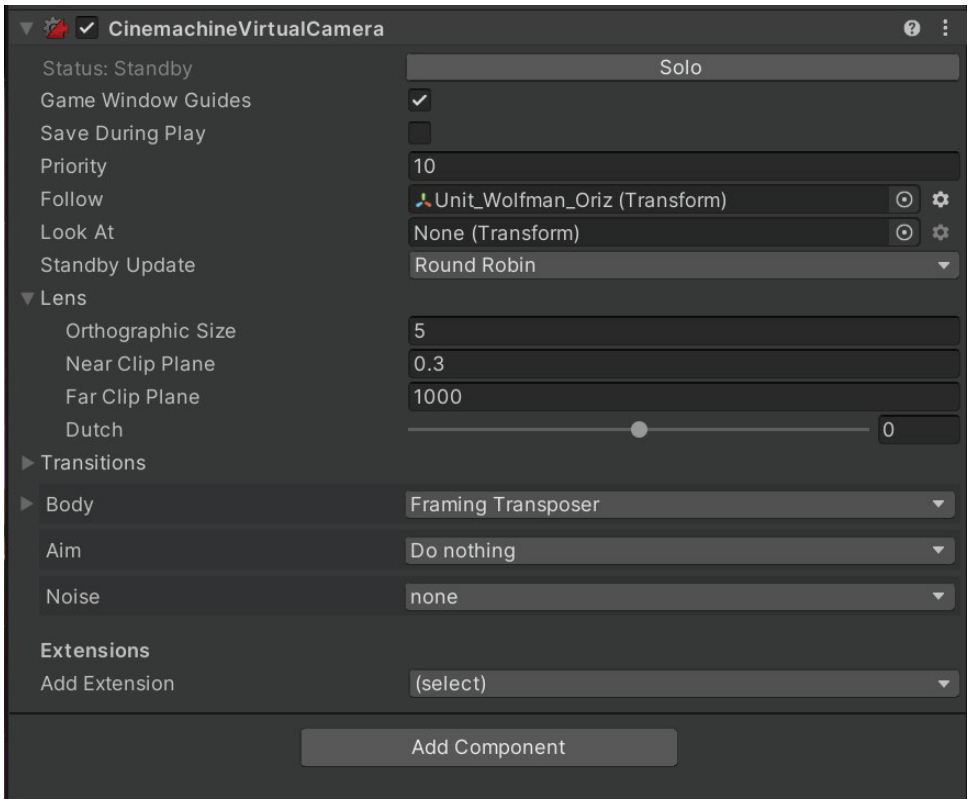
カメラタイプの選択をしたら、そこからゲームプレイに沿った設定をしていく必要があります。Unity の Cinemachine システムには、そうした設定を含めてさまざまな機能があり、カメラをステージの境界内に制限したり、カメラの遷移やノイズを設定したりできます。このセクションでは、2D ゲーム向けの主な Cinemachine 機能をいくつか紹介します。

まず、Package Manager から Cinemachine をインストールします。バージョン 2.7.1 以降を選択します。

Cinemachine は、シーン内に新しい Unity カメラを作成することはありません。その代わりに、新しい Cinemachine バーチャルカメラ（vcam）がシーンに追加されると、デフォルト名が「CM vcam」の新しいゲームオブジェクトとともに CinemachineBrain コンポーネントがメインカメラに追加されます。

Cinemachine Brain は、シーン内のアクティブなバーチャルカメラをすべて監視します。キーフレームを使用して vcam をアニメーション化したり、複数のカメラをブレンドしたり、カメラ間をスムーズに遷移させたり、2 つのカメラを組み合わせて、一方をアニメーション化しつつ、他方に遷移したりすることができます。アニメーションはすべて Cinemachine Brain によって処理され、メインカメラに適用されます。Cinemachine Brain は、メインカメラを動かす強力なアニメーションシステムのようなものです。

2D バーチャルカメラを作成するには、**「Cinemachine」>「Create 2D Camera」**の順にクリックします。これにより、2D 環境向けに設定されたバーチャルカメラが作成されます。作成したのがシーン内の最初のバーチャルカメラの場合は、メインカメラに Cinemachine Brain コンポーネントも追加されます。



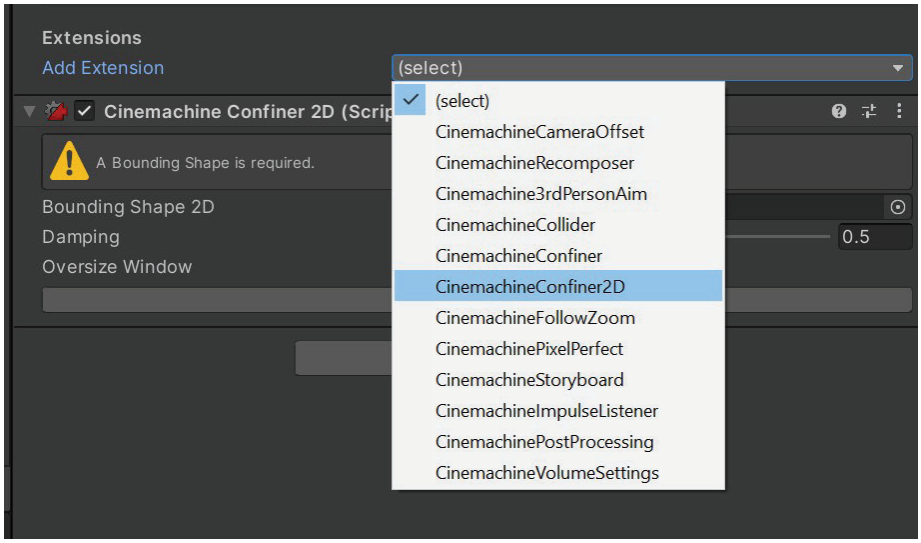

Cinemachine バーチャルカメラを作成する

Cinemachine には追従するオブジェクトが必要であるため、「Follow」フィールドにプレイヤーキャラクターを割り当てます。作業対象を 2D 平面上にして、カメラのパンや傾きを無効にするために、「Look At」フィールドが空白、「Body」が「Framing Transposer」に設定されていることを確認します。最後に、プロジェクトの「Lens」プロパティを設定します。なお、一部のオプションはメインカメラのプロパティを継承します。

「Body」プロパティのフィールドを展開し、「Offset」、「Damping」、「Dead」、「Soft Zones」などの有用なオプションを使用して、バーチャルカメラをどのようにターゲットに追従させるかを変更します。再生モードでこれらのオプションをテストします。「Save During Play」オプションをオンにすると、変更内容が保存されます。

「CinemachineConfiner2D」というバーチャルカメラ用の拡張機能もあります。この機能を有効にすると、ステージの境界線の外にはカメラが移動しないようになり、プレイヤーに見せたい要素だけを表示できます。そして、ステージの不要な部分までデザインしなくても済むようになります。

CinemachineConfiner2D を追加するために、「Add Extension」ドロップダウンメニューから「CinemachineConfiner2D」を選択します。




CinemachineConfiner2D 拡張機能を追加する

Cinemachine2DConfiner を利用するには、「Bounding Shape 2D」を「Collider 2D」(「Composite」または「Polygon」)に設定する必要があります。空のゲームオブジェクトを作成して、Composite Collider 2D と Box Collider 2D を追加します。RigidBody 2D が自動的に追加されるので、その「Body Type」を「Static」に設定します。また、「Box Collider 2D」で「Used by Composite」オプションをオンにします。続いて、このゲームオブジェクトを「CinemachineConfiner2D」スクリプトの「Bounding Shape 2D」フィールドにドラッグします。「Box Collider 2D」のサイズを忘れずに編集し、カメラのサイズ以上にしてください。

以上で、カメラの錐台がコライダーのバウンディングボックス外に出なくなります。詳細については、Cinemachine Confiner 2D に関するドキュメントをご覧ください。

## 描画の順序

2D ゲームでは、すべてのスプライトとオブジェクトは同じ深度となります。それらをソートして、手前に表示するものとそれ以外を区別する方法を説明します。

Unity では、レンダラーがタイプや使用方法に基づく優先順位に応じてソートされます。レンダラーのレンダリング順序は、レンダーキューを使用して指定できます。一般に、主なキューには不透明キューと透過キューの 2 種類があります。2D レンダラーは主に透過キュー内にあり、スプライトレンダラー、タイルマップレンダラー、スプライトシェイプレンダラーといったタイプがあります。

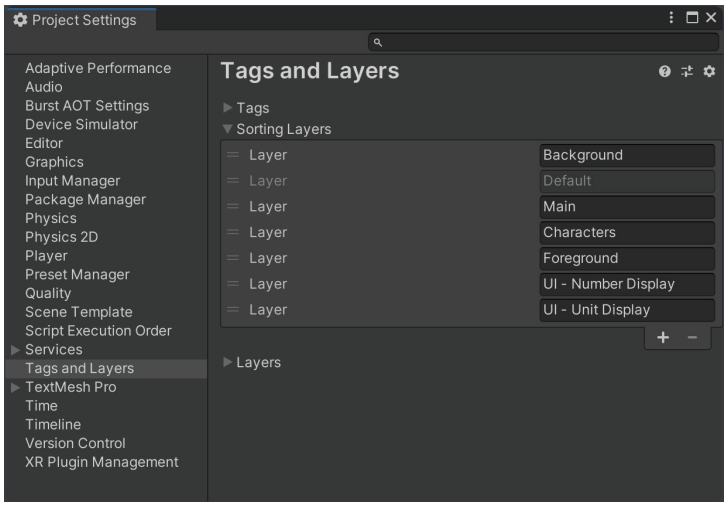
通常、透過キュー内の 2D レンダラーは優先順位に従います。Unity では、同じ空間内に 2 つ以上のオブジェクトが重なっている場合に、以下のリストを確認して、どのオブジェクトを手前に描画すべきかをチェックします。両方のオブジェクトの優先順位が同等で、値も同一の場合は、リスト内の次の条件が評価されます。優先順位は次のようになります。

1. ソートレイヤーとレイヤーの順序
2. レンダーキューの指定
3. カメラからの距離
4. ソーティンググループ
5. マテリアル / シェーダー

両方のオブジェクトで上記の値がすべて同一で、どちらを優先するか決める必要がある場合は、このプロセスによって、どのオブジェクトを手前にレンダリングするかを選択しなければなりません。これは最適なソリューションではないため、ソートレイヤーとソーティンググループを使用して、明確なソート順を設定してください。

## ソートレイヤー

最も重要なソート条件はソートレイヤーです。このオプションはすべての 2D レンダラーにあり、最初に設定する必要があります。デフォルトのソートレイヤーがありますが、「Project Settings」を開き、「Tags」と「Layers」のオプションを指定することで編集できます。




ソートレイヤーを編集する

レイヤーの左にあるハンドルをドラッグすることで、順序を追加、削除、変更できます。リストの上位のレイヤーが最初にレンダリングされ、カメラから遠くに表示されます。

モックアップをデザインするときでもソートレイヤーを計画的に構成し、シーンを早い段階から整理しておきましょう。スプライトのソーティンググループは、スプライトをシーン内に配置してからすぐに設定します。そうすることで、プロジェクト開発を進めていく中で、場合によっては数千枚にも及ぶスプライトを対象に、不意にソート設定の変更を余儀なくされる状況を回避しやすくなります。

**ソートレイヤーをむやみに増やすことは避ける**

2D Lights はソートレイヤーに依存するため、ソートレイヤーの構成を設定する際はライティングも考慮しましょう。ゲーム内でライトをどのように動作させ、どのグループに当てるかを事前に確認しておきます。

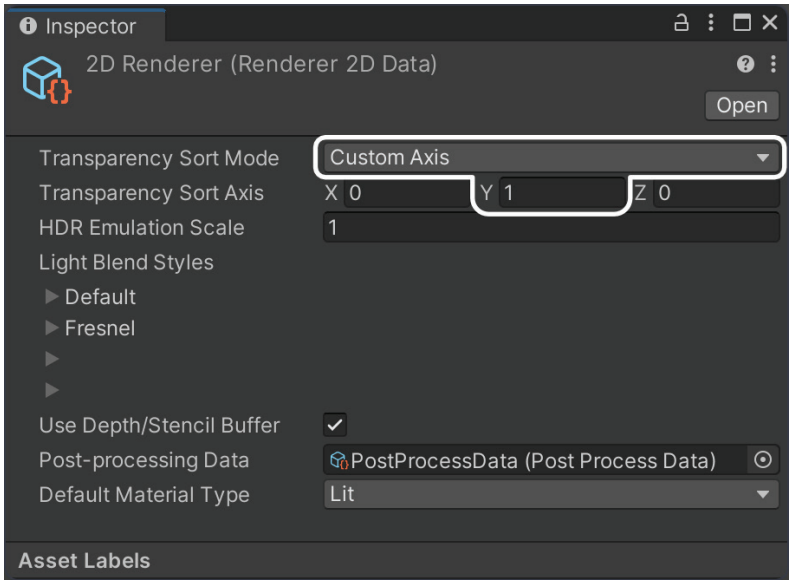
例えば、クォータービューのゲームを制作しており、2D Lights によるトーチを使いたいと考えている場合は、トーチの光を環境全体に当てるか、壁のみに当てるかを考えます。キャラクターに当てるかどうかや、トーチを光の手前に描画するかも考えます。光が当たるオブジェクト専用のソートレイヤーが必要かも考えましょう。ソートレイヤーを編集する際は、こうした点を念頭に置くことが大事です。ライティング計画の詳細については、2D Lights のセクションをご覧ください。

ソートレイヤーの数が多くなりすぎるとディテールの見落としが出てしまう場合がある点に注意してください。さらにレンダラーをソートする必要がある場合は、「Order in Layer」を使用します。これにより、同じレイヤー上にあるオブジェクトを簡単にソートできます。

**Transparency Sort Mode**

ゲーム内で擬似的な 3D ビューを使用したい場合があるかもしれません。例えば、クォータービューのトップダウンゲームで少し角度をつけるために不等角投影や斜投影を使用する場合や、ベルトスクロールゲームでキャビネット投影を使用する場合などです。その場合、レンダラーに対して、カメラからの距離（上記のソート優先度リストでは 3 番目）でカスタムソートを行う必要があります。

オブジェクトをソートするカスタム軸を選択します。トップダウンゲームやベルトスクロールゲームの場合に Y 軸でオブジェクトをソートすると、背の高いキャラクターが他のキャラクターの下にレンダリングされ、離れた場所にいるように錯覚させることができます。このオプションを編集するには、URP の設定時に作成した 2D Renderer アセットを見つけて、「Transparency Sort Mode」オプションを「Custom Axis」に変更し、「Transparency Sort Axis」の値を 0、1、0 に設定します。




スプライトを Y 軸でソートする場合、値を (X:0, Y:1) 1 に設定すると、ソート基準について下向きベクトルに従うよう指定されます（Vector2.down と同じ）。上にあるスプライトから先に描画され、下にあるスプライトが後に描画され、前面に表示されます。

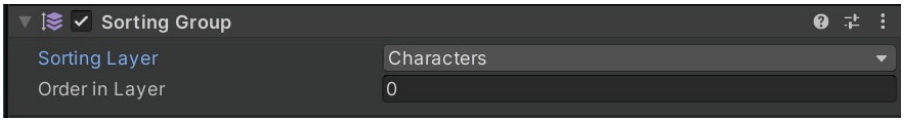
## ソーティンググループ

前の手順どおりに Y 軸でソートしても、キャラクターのパーツが正しく表示されないことがあります。

同じソートレイヤーにある 2 つのキャラクターのパーツが、2 つのキャラクターでごちゃ混ぜに表示されています。

これはバグではなく、単にソートの挙動によるものです。ソートを実行するときに、同じソートレイヤー上でどのパーツがどのキャラクターに付随するかは考慮されません。幸いなことに、ソーティンググループはソート優先順リストで 4 番目の基準です。ソーティンググループは、ソートを目的として、共通のルートを持つレンダラーをグループ化するコンポーネントです。同じソーティンググループに含まれるすべてのレンダラーは、同じソートレイヤー、レイヤーの順序、カメラからの距離を共有します。

いくつかのスプライトで構成されるキャラクターや他のオブジェクトを使用する場合は、このスクリプトを親オブジェクト（階層の一番上）に適用し、他の 2D Renderer と同様に「Sorting Layer」と「Order in Layer」を設定します。その例は、こちらの Unite Now テクニカルセッションで見ることができます。




親ゲームオブジェクトにソートグループを作成して、このオブジェクトとその子オブジェクトが 1 つの要素としてソートされるようにすることで、このオブジェクトのパーツとゲーム内の他のスプライトの間で競合が発生しないようにします。

ソートグループにより、キャラクターが正しく表示されます。

ソーティンググループはネストすることもできます。複数のスプライトで構成される武器をランダムに生成する必要がある場合は、それらをソーティンググループに含めると、キャラクターの手に武器が正しく表示されます。

## 最適化のヒント

- シーンに多くのスプライトが含まれる場合は、タイルマップの使用を検討します。多数のスプライトレンダラー（オーバーヘッドあり）を 1 つのタイルマップレンダラーに置き換えることができます。
- レベルコライダーのバッチ処理を静的としてマークします。
- シーン内で一緒に使用されることが多いスプライトはスプライトアトラスにまとめます。
- Sprite エディターの「Custom Outline」オプションを使用して、スプライトのメッシュを簡略化します。
- 「Cache Geometry」オプション（「Edit Spline」ボタンをオンにしている場合に表示されます）を有効にして、スプライトシェイプジオメトリをキャッシュします。

# 2D アニメーション

2D アニメーションは、ゲームのアート制作において特に多くの時間を要し、困難に直面しやすい部分です。魅力的なアニメーションキャラクターを作成するには、アニメーションのタイミングや勢い、動きなどについて相当な知識が求められます。全フレームを輝くものとするには長い時間を要するうえに、対象となるフレームを保存して表示させるのには大量のメモリの消費が必要となります。タイミングやキャラクターの一部を変更しようとすれば、全フレームをもう一度作画しなおすということになりかねません。

昔から、3D アニメーションは 2D に比べるとシンプルでした。3D モデルを用意して、スケルトンを追加してリグを作成し、ボーンウェイトを設定します。次に、ソフトウェアによって間が補間されるキーフレームを設定することで、その 3D モデルをアニメーション化します。調整はキーフレームの編集で行います。

幸いなことに、Unity は相当効率的に 2D アニメーションを制作するためのツールセットを備えており、3D アニメーション制作におけるシンプルなプロセスを模した様々なアプローチを利用することができます。2D Animation パッケージを使用すると、キャラクターのアートワークを Photoshop から Unity に直接インポートできます。

## デザイン、インポート、リグの作成

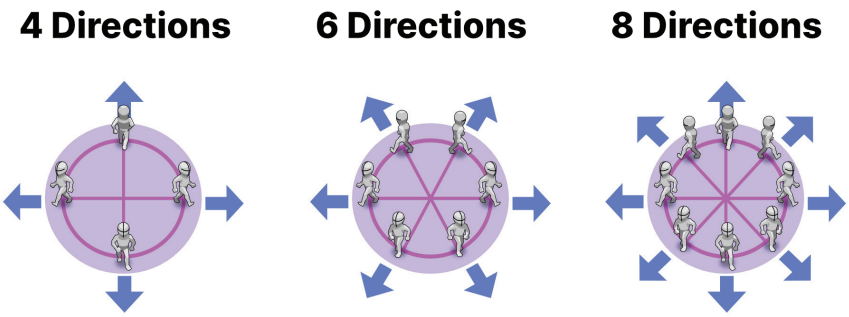
ゲームの最重要要素たるプレイヤーのキャラクターであるため、そのキャラクターをデザインするにあたっては、十分な時間をかけて考えるようにします。計画時において、考えておくべき重要な要素としては、いくつかあります。

**視点**

どのような視点を選択するかによって、ゲームのキャラクターの見た目やアニメーションの動き方が変わります。ほとんどのゲームビューでは、キャラクターは横顔を見せる形で描画することができます。わずかに回転させてスリークォータービューにすると、顔の特徴がよくわかるようになります。

等角投影または ¾ の見下ろし視点を使用すると、同じようなビューが得られますが、こちらは上から少し傾けたカメラが顔の詳細を示す形になります。この視点では、多方向に向けてキャラクターが描画されます。目指す結果や予算に合わせて、一般的に使用される 3 つのオプションから 1 つを選択できます。

- 4 方向
- 6 方向
- 8 方向




見下ろし型キャラクターアニメーションのさまざまな処理方法

いずれの場合も、右側または左側を向いたアニメーションを反転して反対側に向かせる必要があるかどうかを決めなければなりません。反転する場合は、選んだ方向性に応じて、以下の描画が必要になります。

- 3 方向：右、上、下
- 3 方向：右、右上、右下
- 5 方向：右、右上、上、右下、下

アニメーションを反転すると、それに応じてキャラクターの手も反転します。キャラクターが右を向いて、右手に剣、左手に盾を持っている場合に、そのキャラクターを左向きに反転すると、それぞれの武具を持つ手が逆になります。このトレードオフを受け入れるか、あるいはもっと時間をかけて、向きに応じたアニメーションをすべて作成するか、どちらかを選びます。

古くからある横スクロールのベルトスクロールゲームをデザインするときは、1 つの向きだけを使用することもできます。キャラクターが上下に動くと、横歩きをしているように見えます。

つきつめて言ってしまうと、テスト後にキャラクターがゲーム環境内で良く見えているかというのは、作り手の感覚次第ということになります。リアルなビューの角度や視点よりも、何より重要なのは皆さんのビジョンなのです。

**キャラクターの再スキニング**

2D Animation パッケージのもう 1 つの利点は、複数のキャラクターで同じスケルトンやアニメーションを共有できるところです。一旦ベースとなるキャラクターをデザインしてリグを設定し、アニメーション化してしまえば、そのスキンを簡単にスワップすることができるようになります。

事前に計画を立てておくことで時間の節約になります。すべてのキャラクターが同じスケルトンを共有するというコンセプトを作り出して、次にそのスケルトンがどのキャラクターにもフィットするかどうかを確認するようにします。そのためには、別のレイヤーでスケルトンを描き、すべてのキャラクターにオーバーレイします。必ず全キャラクターでレイヤー数が同数になるように注意してください。

> **ノート：**Unity 2021.1 以降のバージョンでは、新機能「Skeleton Sharing（スケルトン共有）」として、この作業は簡素化されております。

画像編集アプリケーションでスケルトンの描画を重ね合わせる

### パフォーマンス

スケルタル 2D アニメーションは、基本的にスプライトのスワップであるフレーム単位のアプローチに比べて、多くの処理能力を必要とします。パフォーマンスを上げるコツとしては、Package Manager から Unity の Burst および Collections パッケージをインストールすることがあります。これにより、アニメーション化されたスプライトを変形させる際のランタイムパフォーマンスが改善されることとなります。

次に、ターゲットプラットフォームに必要なリソースを決定し、一度に画面に表示するキャラクターの数を定めます。

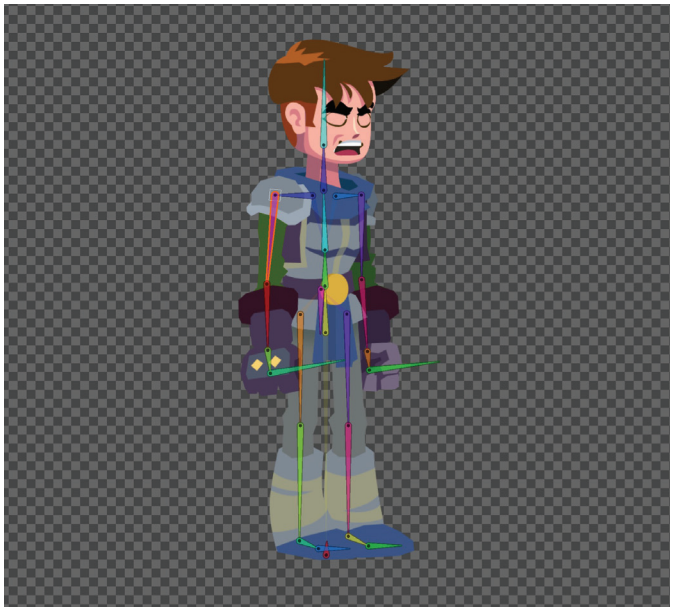
例えば、画面上に登場するキャラクターが 2、3 体ほどのアドベンチャーゲームの場合は、キャラクターのディテールの足し引きは構わずにしておけます。追加のボーンやレイヤーをふんだんに使用して、高度な IK も取り入れて、目や指など、色々動かしてみましょう。

しかし、大規模な戦闘もあり、10 体を超えるキャラクターが表示されるモバイル RPG や撃ち合うようなゲームでは、リグはシンプルにしておくことをお勧めします。例えば、腕や脚ごとにボーンを 2 つではなく 1 つだけ使用したり、異なるレイヤーで目を動かす代わりに顔全体をスプライトスワップするなどの方法をとるとよいでしょう。

ゲームアートをデザインするときは、レイヤーがいくつ必要になるかやキャラクターをどのようにアニメーション化するかについて、入念に計画を立てておくようにします。これらの要素は、以降の制作段階で変更することがほとんど不可能だからです。

### 2D アニメーションの基本原則

後でリグを設定してアニメーション化する予定のキャラクターをデザインする際に役立つルールをいくつか紹介します。



キャラクターのニュートラルポーズの例：脚と腕はどの位置でも構いませんが、曲がっていない状態にする必要があります。曲がった状態の手足をまっすぐにすると、ピクセルが伸びてしまいます。

— 腕と脚を伸ばした普通の姿勢でキャラクターを描画する。体のパーツを曲がった状態で描くと、アニメーション化するときに問題が生じる場合があります。

— ゲームの PPU が示す値を少し上回る解像度を使用する。解像度によっては、静止時に良く見えても、画像の回転や引き伸ばしによってピクセレーションが発生することがあります。

— ゲーム内で 2D ライティングを広く使用して、法線マップを最大限に活かしたい場合は、スプライトに光や影をペイントしないようにする。その代わりに、Non-Directional（非指向性）の影をペイントします。この手法は「アンビエントオクルージョン」と呼ばれます。スプライトの見た目が良くなるのはそうなのですが、日光のようなディレクショナルライトの使用は避けるようにしておきましょう。

— スプライトスワップ機能を使用してスワップしたボディパーツレイヤーは、適宜グループ化する。例えば、口の位置を伴うレイヤーはすべて、画像編集アプリケーションで「口」というグループに含めます。

以上のようなデザイン時の留意点を踏まえて、好きなグラフィックアプリケーションを起動し、キャラクターを描きましょう。スプライトスワップの詳細については、こちらのドキュメントページをご覧ください。

## Unity へのキャラクターのインポート

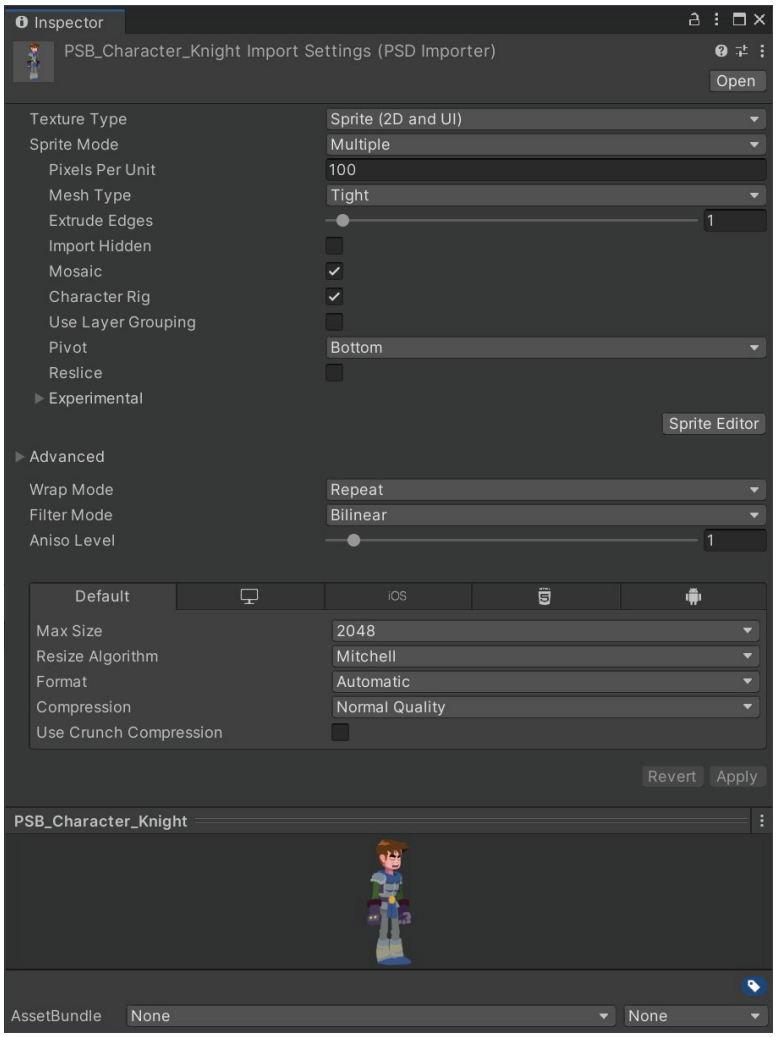
キャラクターが完成したとなれば、Unity へインポートする必要があります。手早いのは、2D PSD インポーターを使用することです。この方法では、キャラクターのすべてのレイヤーがスプライトとしてインポートされ、アプリケーションでペイントしたままの状態で保持されます。

Unity で 2D プロジェクトテンプレートを使用している場合は、インポーターを先にインストールしておく必要があります。まだの場合は、Package Manager を使ってインストールしてください。

エクスポートのプロセスでは、PSB 形式でファイルを保存する必要があります。PSB 形式は Photoshop の PSD に類似していますが、より大きなファイルサイズに対応しています。Photoshop を使用している場合は、この形式を「Save」ダイアログボックスで選択します。他のアプリケーションを使用している場合は、プロジェクトを PSD ファイルとしてエクスポートし、ファイル名の拡張子を PSB に変更してから Unity にインポートします。

Unity にインポートする方法は、他のアセットでの場合と同じです。対象のファイルを Assets フォルダーに保存するか、「Project」ウィンドウにドラッグします。

インポートされた PSB ファイルを選択すると、スプライトインポート設定に類似したオプションが表示されますが、2D アニメーションとリグ設定用の設定も含まれています。




キャラクターのインスペクター

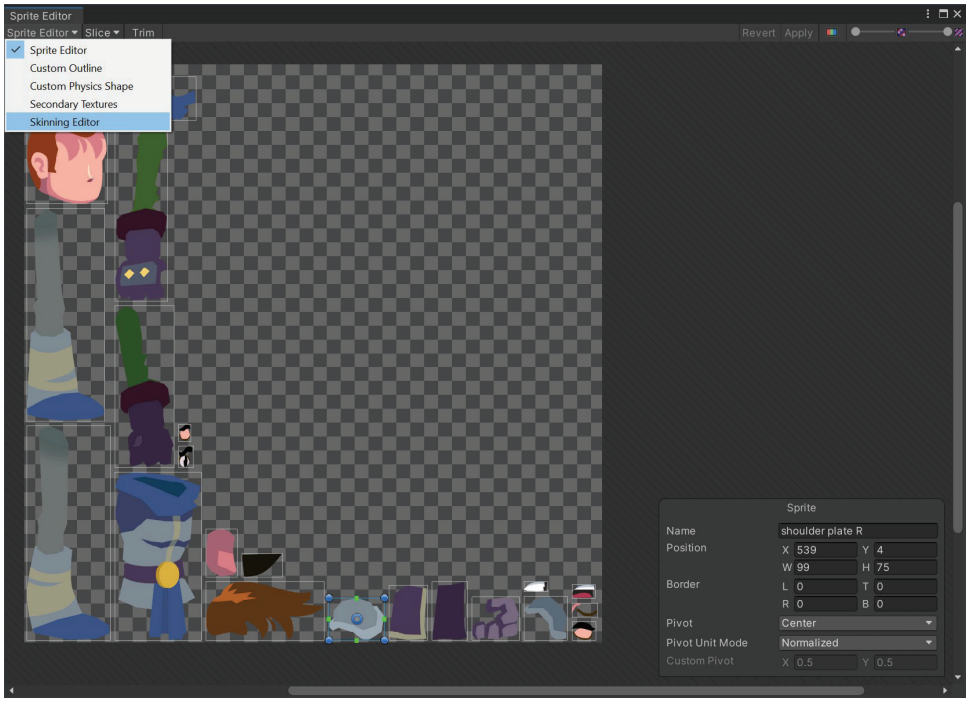
主要な設定オプションは次のとおりです。

- Import Hidden：このオプションは PSB ファイルから、非表示のレイヤーを含めすべてのレイヤーをインポートします。
- Mosaic：このオプションは「Texture Type」が「Multiple」に設定されている場合にのみ使用できます。インポートされたレイヤーからスプライトを作成し、テクスチャアトラス上に配置します。キャラクターのリグを設定する場合は、このオプションを有効にしておきます。
- Character Rig：このオプションは、PSB ファイル内のものと同じレイヤー階層と位置を使用して、キャラクターのプレハブを生成します。このオプションはオンにする必要があります。
- Use Layer Grouping：このオプションは PSB ファイルからレイヤーのグループ化を追加します。これをオンにすると、スプライトスワップされるパーツなど、キャラクターのパーツがグループ化されます。

オプションを設定したら、「Apply」をクリックします。これでキャラクタープレハブが完成し、シーンにドラックできるようになります。

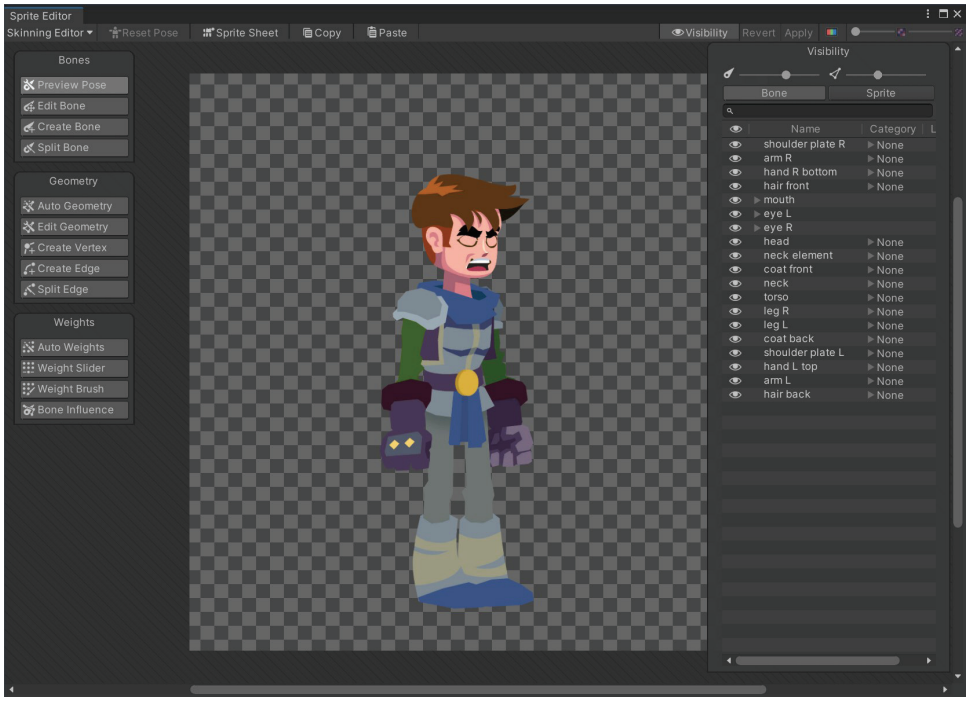
## キャラクターの作成

キャラクターのリグの設定を開始するには、インスペクターで「Sprite Editor」ボタンをクリックして、スプライトのインポート設定にアクセスします。




「Sprite Editor」ウィンドウでツールを選択する方法

「Sprite Editor」ウィンドウの左上にあるドロップダウンメニューから「Skinning Editor」を選択します。




キャラクターをアニメーション化するためのオプションを備えたスキニングエディター

このウィンドウでは、次の操作を実行できます。

— ボーンを作成および編集する

— スプライトのジオメトリを作成および編集する

— スプライトに対するボーンの影響を編集する

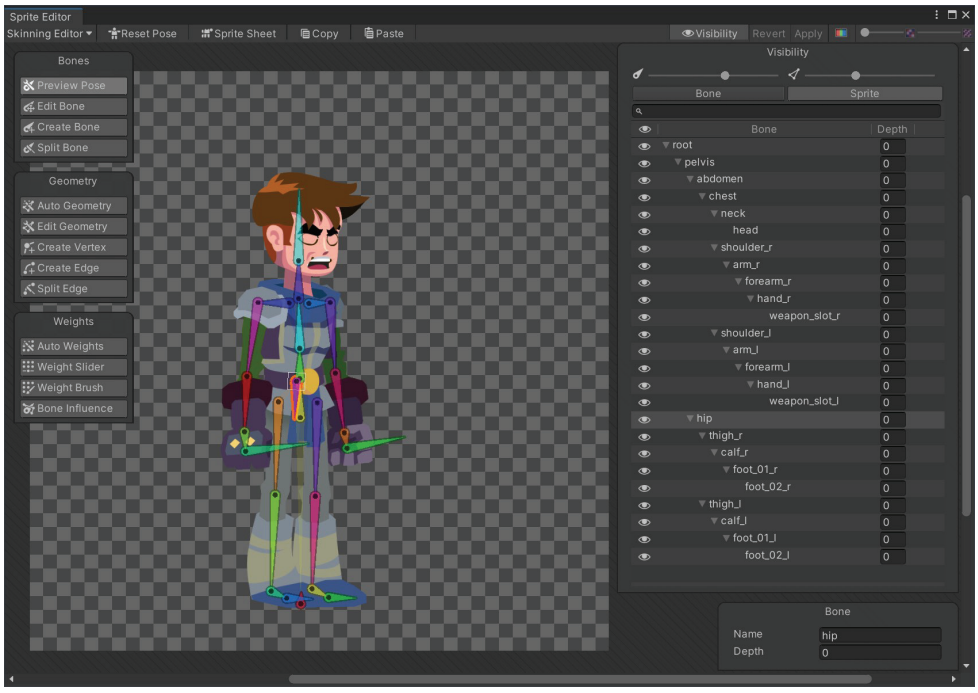
— スキンとスプライトスワップに使用するカテゴリとラベルを作成する

**スケルトンの作成**

最初にキャラクターのスケルトンを作成します。「Create Bone」ボタンを選択して、メインウィンドウ領域を左クリックします。初回のクリックでボーンの中心が作成され、2 回目のクリックでボーンの先端位置がマークされます。このツールで、ボーンどうしのチェーンやネストを設定します。

ボーンの配置を変更する場合は、右マウスボタンを押して、新しいボーンを配置する場所を左クリックします。どの既存のボーンが作成中のボーンの親になるかを制御するには、既存のボーンを左クリックして選択し、次に新しいボーンを作成します。

「Edit Bone」ボタンを使用してボーンを調整します。「Split Bone」ボタンを押すと、ボーンを 2 つに分割できます。これは腕や脚を作成するときに役立つオプションです。片脚のボーンを作る場合は、膝になる場所をクリックすると、ボーンが太ももの部分とふくらはぎの部分に分割されます。



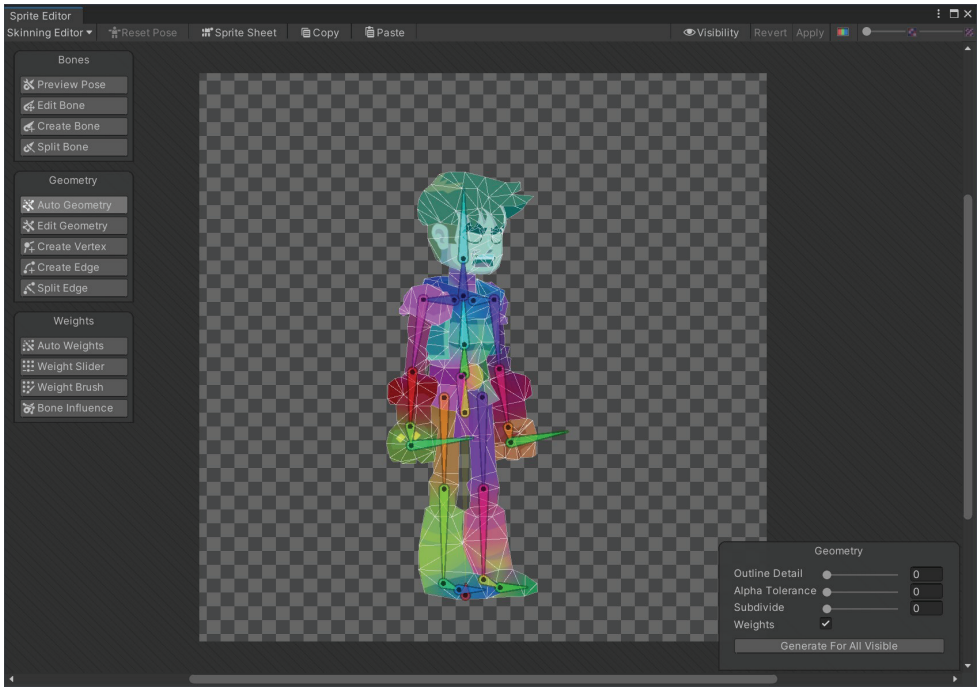

「Bone」タブを開いて、親ボーンと子ボーンを整理する

ボーンのリストビューで、ボーンの親子関係や名前を変更することもできます。このビューを開くには、右上のバーにある「Visibility」ボタンをクリックし、「Bone」タブを選択します。ボーンの親を変更するには、リストビューでボーンをドラッグします。ボーンの名前を変更するには、アクティブなボーンの名前をクリックします。ボーンに名前を付けると、それらを後で見つけやすくなります。階層が正しいことを確認するには、「Preview Pose」ボタンを選択して、いくつかのポーズを試してみます。ボーンの位置をリセットするには、ツールバーの「Reset Pose」ボタンを押します。

**スプライトジオメトリ**

ボーンにスプライトを割り当てるには、ジオメトリを作成する必要があります。まず「Auto Geometry」ボタンを押します。小さなポップアップウィンドウが開きます。ここではジオメトリをどのように作成するかを指定できます。

すべてのスライダーを 0 に合わせて、できるだけジオメトリをシンプルにしておくとよいでしょう。「Weights」オプションを有効にすると、ボーンとスプライトの関連付けが自動で行われます。「Generate For All Visible」ボタンをクリックすると、ボーンのウェイトが作成され、すべてのスプライトに設定されます。この設定を個々のスプライトに実行するには、対象のスプライトをダブルクリックします。この方法は特定のスプライトのジオメトリを調整するときに便利です。




スプライトのジオメトリを自動生成する

この生成されたジオメトリをそのまま使用するのではなく、頂点の数とジオメトリの曲がり方を十分に調整するには、メッシュを手動で編集する必要があります。以下のツールを使用します。

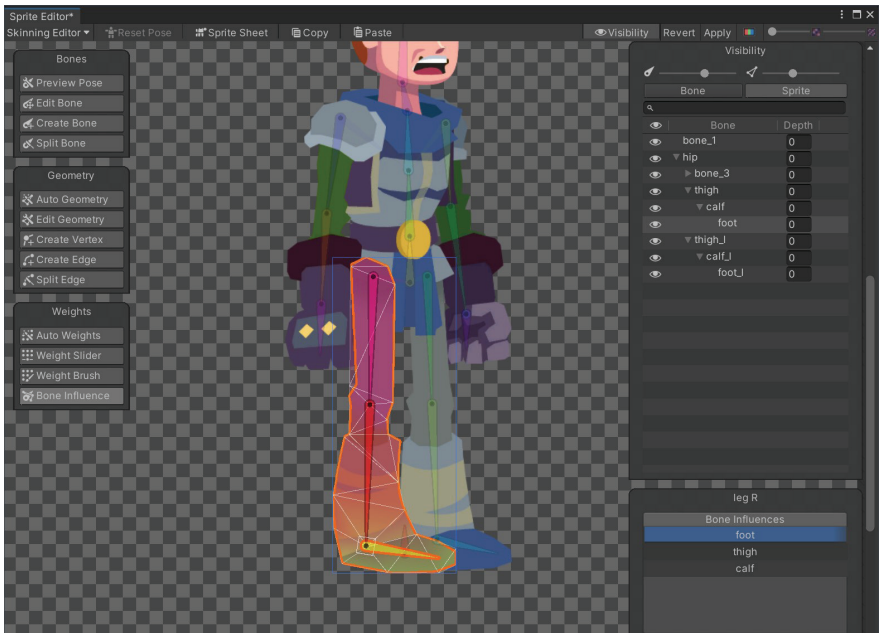
- Create Vertex：新しい頂点（または点）を作成します。また、それらの点を移動したり削除したりすることもできます。
- Create Edge：既存の 2 点の間にエッジを作成します。または、左マウスボタンを使用して新しい点を作成し、エッジを作成します。右マウスボタンで選択中の点を選択解除し、他の頂点との間にエッジを作成することもできます。
- Edit Geometry：頂点を移動します。
- Split Edge：中間地点に頂点を作成して、エッジを分割します。

使用する頂点はできるだけ少なくします。それぞれの頂点位置はボーンの回転に基づいて算出する必要があるため、頂点が少ない方がパフォーマンスの節約になります。また、頂点が少ないとウェイトが設定しやすいので、メッシュの屈曲の見栄えが良くなります。さらに、ターゲットプラットフォームごとに理想的なゲームの頂点数があるため、頂点を少なくするのはジオメトリを最適化するうえで常に効果的な考え方です。

### ウェイト

すべてのスプライトでジオメトリを整えてきれいにしたら、ウェイトの設定に進みます。ウェイトは各頂点に対するボーンの影響を 0 から 1 の範囲で定義します。0 に設定すると、ボーンはその頂点にまったく影響も及ぼさず、1 に設定すると、その頂点はボーンにくっついているかのように動きます。メッシュに適切なウェイトを設定すると、自然なカーブを作成することができます。ウェイトの設定が良くないと、スプライトが歪んで表示され、ゲームの雰囲気は台無しになります。

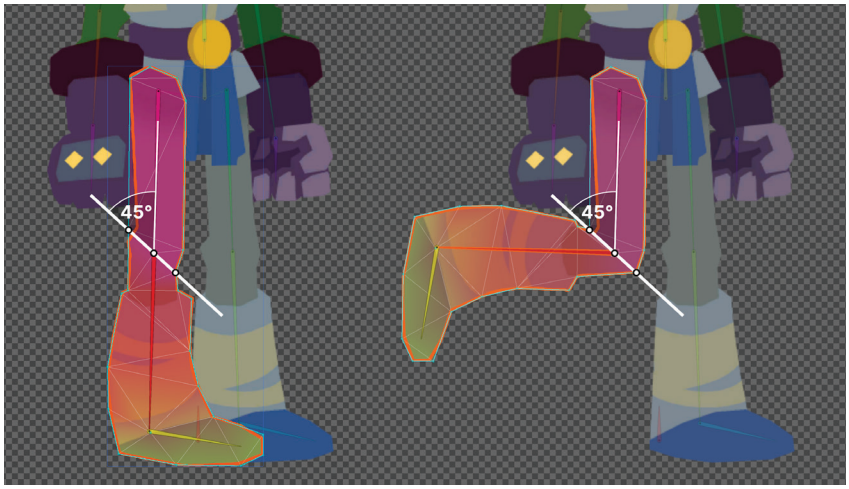
ウェイトを設定するにあたって、特定のスプライトに影響を及ぼすボーンを指定する必要があります。「Bone Influence」ボタンをクリックし、スプライトを選択します。



スプライトに対するボーンの影響を設定する

小さなポップアップウィンドウで、どのボーンが選択したスプライトに影響を及ぼすかを設定します。スプライトに影響を及ぼす必要があるボーンだけを設定し、残りは削除します。

この状態から、「Weight Brush」と「Weight Slider」を使用してウェイトを設定します。「Weight Brush」を使用して、マウスで頂点をペイントすると、選択したボーンの影響範囲がそれらの頂点にすぐに追加されます。「Weight Slider」はより正確なツールで、1 つ以上の点を選択してから、スライダーでボーンごとに影響を厳密に指定できます。ブラシは手早くウェイトを設定するときに便利で、スライダーは、肘や膝といった、ボーンが曲がる領域を細かく調整するときに有用です。




手足の曲がり具合を最も見栄え良くするための頂点の配置方法

**ヒント:** 肘や膝の部分を作っているときにベストな見た目を実現するには、2 つのボーンが接合して曲がる部分の中央を通る直線に沿って、内側と外側の頂点を整列させます。この直線は、45 度の角度でボーンを横断するようにします。これらの頂点は上に位置するボーンによってのみ影響を受けるようにするとよいでしょう。ただし、キャラクターはそれぞれ異なる存在ですから、自由にウェイトを変えてみて、最善の結果が得られるまでカスタマイズしてください。

急いでいる場合は、『*Dragon Crashers*』のプロジェクトを開き、プレハブとプレハブバリアントのセクションで、以下のキャラクタープレハブを探します。

- Prefab_Character_Base
  - *PV_Character_Witch*（自動リバインドには別の Sprite Library を使用）
  - *PV_Character_Knight*
  - *PV_Character_Wolfman*
  - *PV_Character_Skeleton*

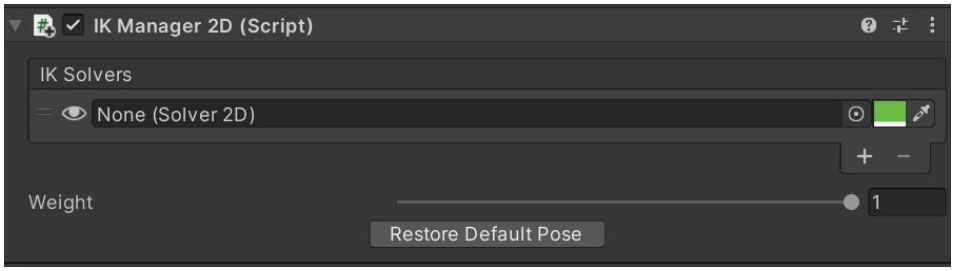
これらのプレハブは、同じ構造を持つキャラクター間で共有できるアニメーションの好例です。

## 2D インバースキネマティクス

普段は意識することがないかもしれませんが、人間の体の動きは非常に複雑です。水の入ったグラスをテーブルから持ち上げるときは、グラスがある空間の地点に手を移動させなければなりません。腕と前腕を回転させることを特に考えなくても、この一連の動作がこなせるのは、脳がバックグラウンドでこの計算を処理しているからです。

あるゲーム内において、同様の動きをアニメーションするには、腕と前腕、両方の回転を同時にアニメーション化する必要があります。両方の回転を合わせつつ自然に見える手の動きをつけるのは、一筋縄ではいきません。2D Inverse Kinematics（2D IK）は、2D Animation パッケージに含まれるツールであり、回転を計算してターゲットの位置にボーンのチェーンを動かすことができます。

最初に、階層の一番上にあるオブジェクトに IK Manager 2D を追加する必要があります。このコンポーネントは、キャラクターのすべての IK ソルバーを管理する役割を担います。

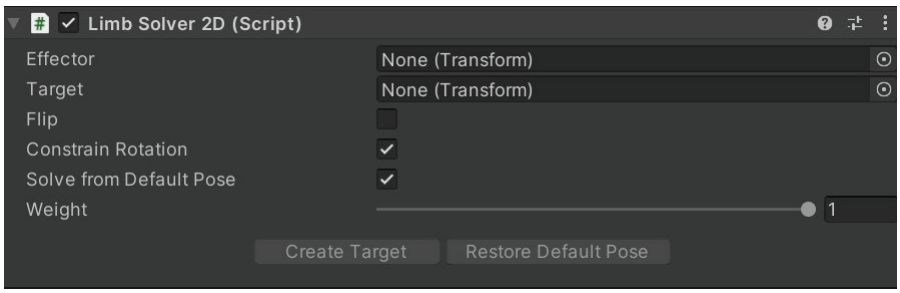



IK Manager 2D コンポーネント

「+」ボタンをクリックし、新しいソルバーを追加します。このソルバーは、ターゲットのトランスフォームに合わせてボーンの回転を計算します。次の 3 種類のソルバーが利用できます。

- Limb：腕と脚に使用する標準のソルバーです。最大 2 つのボーンおよびエフェクターを解決することができます。
- Chain（CCD）– Cyclic Coordinate Descent（循環座標降下）：アルゴリズムの実行時間が長くなるにつれて正確性が高まっていくソルバーで、長いボーンチェーンに適しています。
- Chain（FABRIK）– Forward And Backward Reaching Inverse Kinematics（前方後方到達インバースキネマティクス）：Chain（CCD）と同様に、アルゴリズムの実行時間が長くなるにつれて正確性が高まっていくソルバーです。ボーンがリアルタイムで別の位置へと操作される場合に、すぐに適応します。

ヒューマノイドキャラクターでは、Limb Solver がベストです。その理由は、2 つのボーンを持つ腕や脚に最適化されていて最も高速だからです。最初に Limb Solver を IK Manager 2D のリストに追加します。すると、新しいゲームオブジェクトが作成され、その中に Limb Solver 2D コンポーネントが含まれています。このゲームオブジェクトの名前を「Leg R LimbSolver2D」や「Arm L LimbSolver2D」といった、内容がわかる名称に変更します。



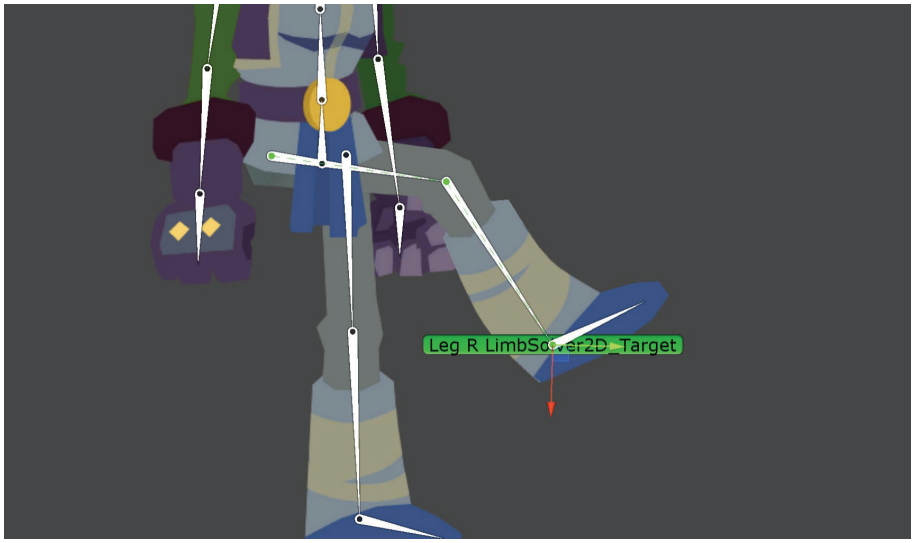

Limb Solver 2D コンポーネント

ソルバーの動作にはエフェクターとターゲットという 2 つのゲームオブジェクトトランスフォームが必要です。エフェクターは、例えば指の先端のボーンなど、チェーンの最後のボーン内に配置され、そこからターゲットの位置に到達しようと試みます。

最初にエフェクターを作成します。脚の IK を作成するときは、太ももとふくらはぎ（またはすね）の 2 つのボーンがあります。ふくらはぎのボーンを選択し、新しいの空のゲームオブジェクトを作成して、ふくらはぎのボーンの子としてエフェクターを配置します。そのゲームオブジェクトの名前を「Leg Effector」にして、エフェクターをボーンの先端に移動します。

新しく作成したエフェクターを Limb Solver 2D の「Effector」フィールドに追加し、「Create Target」ボタンを押します。Limb Solver オブジェクトの中にターゲットのゲームオブジェクトが作成されます。

ターゲットのオブジェクトが動くと、脚もついていくようになりました。




2D IK の動作：ターゲットゲームオブジェクトを動かすと、ターゲットの位置に合わせて、大腿部とふくらはぎの回転が自動的に計算されます。

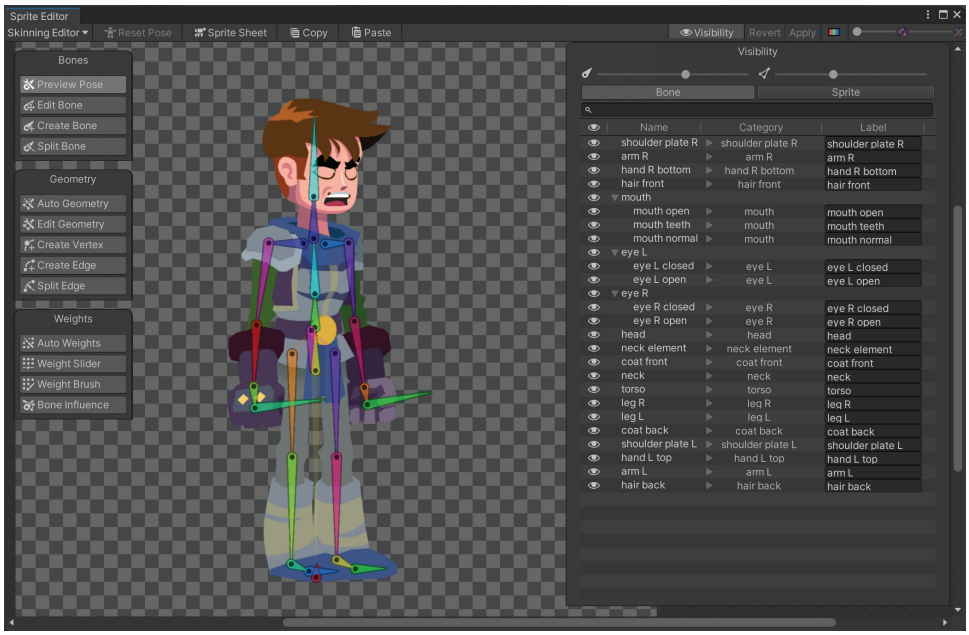
もう一方の脚にも同じ手順で IK を追加します。IK は腕や脚以外のボーンにも追加できます。例えば、頭や首に追加すると、キャラクターが周りを見渡せるようになります。

ここまでの手順を習得したら、さらに高度なユースケースがあります。例えば、IK を設定してキャラクターに銃を構えさせたり、歩行のプロシージャルアニメーションを作成したりするなどの用途です。

## スプライトスワップとスキン

ボーンの回転で動くアニメーションばかりではありません。それ以外にも、顔の表情や手のポーズを付ける必要が出てきます。そうしたときに、スプライトスワップを使用して、スプライトを別のものに差し替えることができます。

スプライトスワップを使用するには、まずキャラクターのスプライトごとにカテゴリとラベルを割り当てます。手早くスプライトを割り当てるには、スキニングエディターを使用します。「Visibility」ボタンを押して、「Sprite」タブに移動すると、すべてのスプライトがリストで表示されています。スワップするレイヤーを個別のグループに配置してから PSB にエクスポートする場合は、この時点でスプライトをグループ化しておきます。この手順が重要である理由は、グループ化されたスプライトが Sprite Resolver に追加され、スプライトのスワップが可能になるからです。

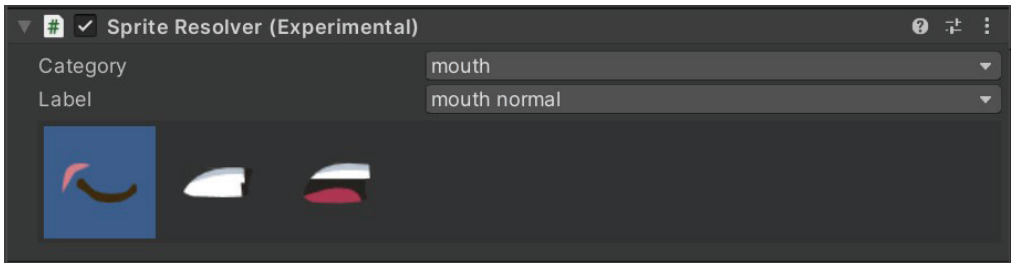



スキニングエディターで各スプライトにカテゴリーとラベルを割り当てる（Sprite Resolver で同じカテゴリのスプライトを使用したり、スワップしたりできる）

リストですべてのスプライトとグループを選択し、右クリックします。次に、「Convert Layer to Category」と「Convert Group to Category」という 2 つのオプションを順に選択します。これらのオプションは、それぞれのスプライトに適切なカテゴリとラベルを自動で割り当てます。

スワップされるスプライトは、グループ名に基づいて 1 つのカテゴリに収められます。上部の「Apply」ボタンをクリックします。キャラクターのプレハブ内に Sprite Library アセットが作成されます。ここには、それまでに作成されたカテゴリとラベルがすべて含まれています。後で必要になるので、このアセットを覚えておいてください。

グループに以前から含まれていたスプライトを確認すると、カテゴリごとに 1 つのスプライトがあり、Sprite Resolver コンポーネントが追加されていることがわかります。




Sprite Resolver コンポーネント

スワップされるスプライトは Sprite Resolver に追加されているので、それらをクリックして、スプライトスワップを実装することができます。

スプライトスワップを使用して、顔の表情や目、口をアニメーション化し、リップシンクのアニメーションを作成して、手振りの変更などを行います。スプライトスワップで、キャラクターが身に付ける帽子や鎧などの装備をスワップすることも可能です。

**スキン**

Sprite Library アセットを使用して、キャラクターのスキンを作成できます。スキンは、キャラクターのアニメーションを維持したまま見た目を変えることができるので、大幅な時間の節約になります。1 つのベースキャラクターにすべてのスクリプトを設定しておけるので、そうしておけば、どういった変更であっても他のキャラクターへそれが適用されるようになります。

以下に示すのは、キャラクターにスキンサポートを設定するワークフローです。

1. ベースキャラクターを完成させます。リグの設定と、IK と Sprite Library の設定を済ませます。そのキャラクターからプレハブを作成します。キャラクターのアニメーション化はいつでも可能です。

2. ベースキャラクターのスキニングエディターに移動して、ツールバーの「Copy」ボタンをクリックします。すると、ボーンとメッシュがウェイトとともにコピーされます。

3. このキャラクターのプレハブバリアントを作成します。このプレハブバリアントは新しい外観を持つ新規キャラクターになります。

4. 新しいキャラクターの PSB をインポートして、そのキャラクターのスキニングエディターを開きます。

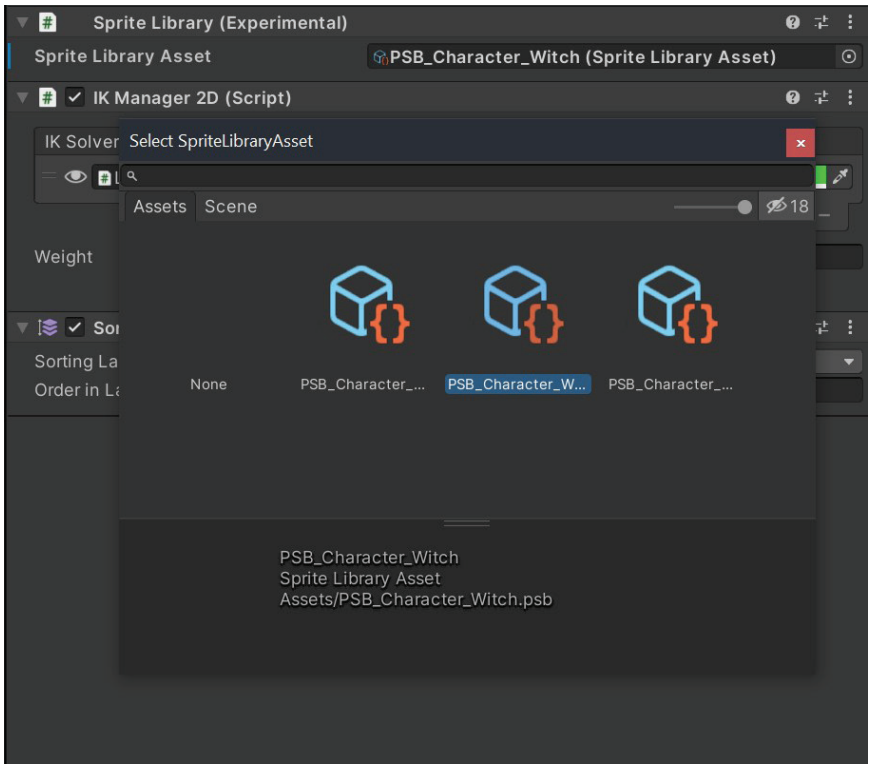
5. ツールバーの「Paste」ボタンをクリックして、ステップ 2 でコピーしたスケルトンに貼り付けます。ポップアップウィンドウで「Bones」と「Mesh」のオプションを選択していることを確認し、「Paste」ボタンをクリックして確定します。

6. 新しいスプライトに合わせてジオメトリを修正します。もう一度ウェイトを確認します。

7. 「Visibility」ボタンを押して、「Sprite」タブに移動します。リストにあるスプライトとグループをすべて選択し、それらを右クリックして、「Convert Layer to Category」と「Convert Group to Category」という 2 つのオプションを選択します。上部の「Apply」ボタンを押します。

8. ステップ 3 で作成したプレハブバリアントに移動します。Sprite Library コンポーネントに移動し、新しいキャラクター用に Sprite Library アセットをスワップします。

9. 新しいキャラクターを作成するには、ステップ 2 から 8 を実行します。




Sprite Library コンポーネントで Sprite Library アセットを選択する

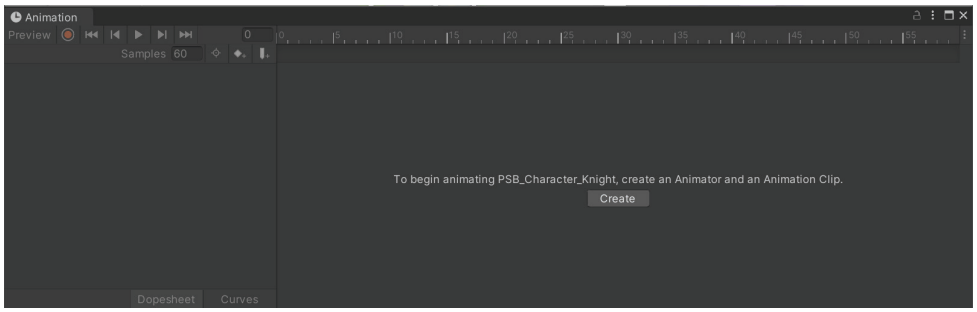
このワークフローを進めると、ベースキャラクターのプレハブができあがります。他のキャラクターはこのプレハブのバリアントになるため、ベースのプレハブと同じコンポーネントを持ちます。ベースキャラクターに IK やソーティンググループの追加や改変などの変更を加えると、他のキャラクターはその変更を継承します。

## アニメーションの基本

説得力のあるキャラクターを作りたいとなると、アニメーションは大変重要になってきます。

素晴らしいアニメーションを作るには、アニメーションの原理とそのツールについて学習し、実践していかなければなりません。基本的なアニメーションの原理の学習は手をつけないままなこともよくあるものです。しかし少し時間を割いて、キャラクターの動きを洗練されたものにすることを学んでおけば、ゲーム作りに大きな恩恵をもたらします。

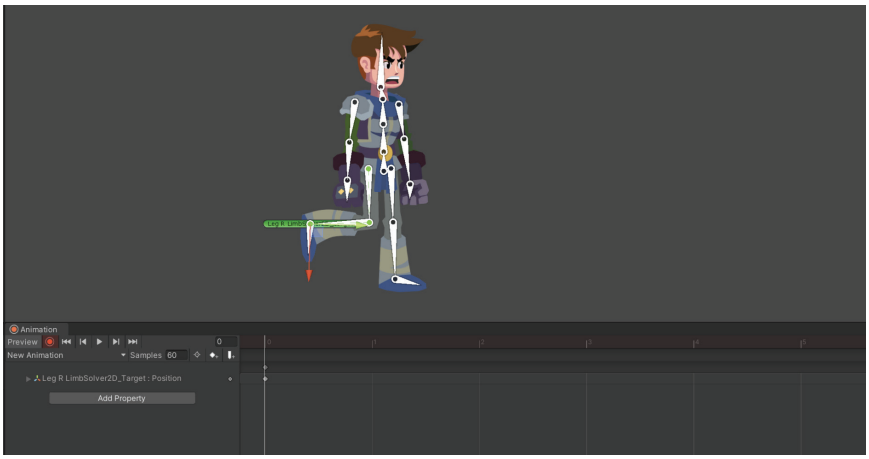
最初のアニメーションを作成するために、**「Window」>「Animation」>「Animation」**の順に移動して、「Animation」ウィンドウを開きます。




「Animation」ウィンドウ

キャラクターを選択します。「Create」ボタンをクリックすれば、はじまりのアニメーションクリップが作成されます。まずはそのクリップに名前を付けるところから始めましょう。アニメーションクリップとは、リニアで録ったもののように、オブジェクトの位置、回転、スケールなどのプロパティが時間と共にどう変化するかを記録したものです。またアニメーターコントローラーも作成されます。このコントローラーは、すべてのアニメーションクリップを管理し、どのクリップを再生し、いつアニメーションを変化させたりブレンドしたりするかといった設定を保持しています。

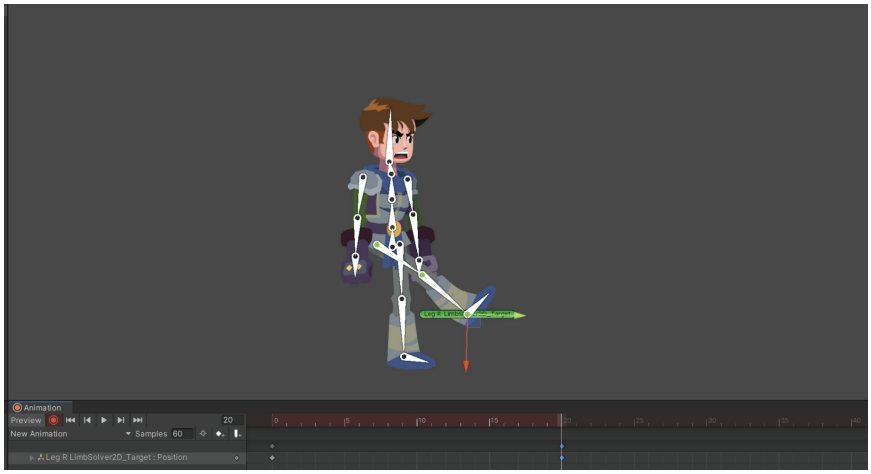
アニメーションの作成を開始するには、赤い記録ボタンをクリックします。すると、キャラクターに加えたすべての変更がアニメーションクリップに記録されます。




「Record」ボタンを押すと、変更を加えるたびにアニメーションタイムラインにキーフレームが作成されます。

アニメーションビューの右側に、現在のクリップのタイムラインがあります。このタイムラインには、アニメーション化されるプロパティごとのキーフレームが表示されます。縦に伸びている白いヘッドラインは現在のフレームを示しています。その線がフレーム 0 の位置にあることを確認します。キャラクターに好きなポーズをとらせてみてください。タイムラインに 1 つ以上のキーフレームが作成され、左側にアニメーションが作成されるプロパティの名前が表示されます。

では、ヘッドラインを数フレームだけ進めて、キャラクターのポーズを変更してみましょう。新しいキーフレームがポップアップされます。

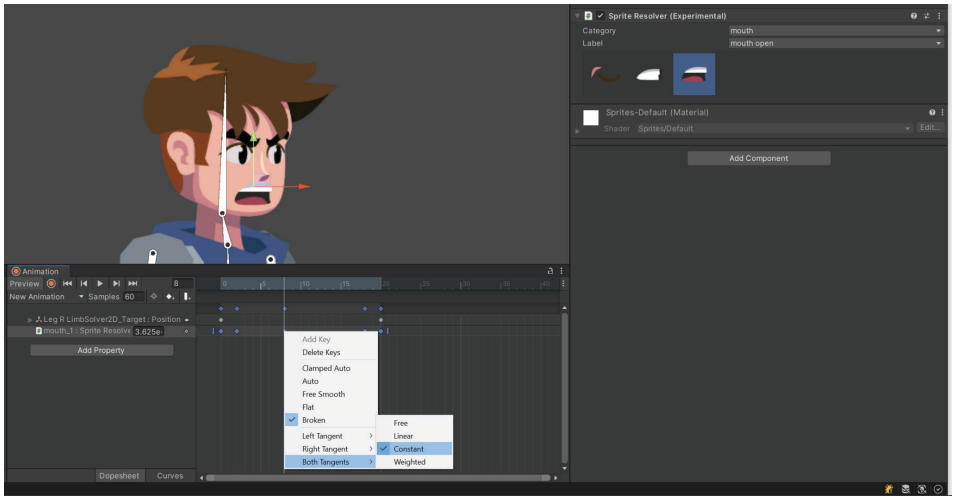



タイムラインに沿って別のキーフレームを追加すると、キーフレームを補間するアニメーションが作成されます。

再生ボタンを押すと、キャラクターの動きを確認できます。初めてのアニメーションが作成できました。

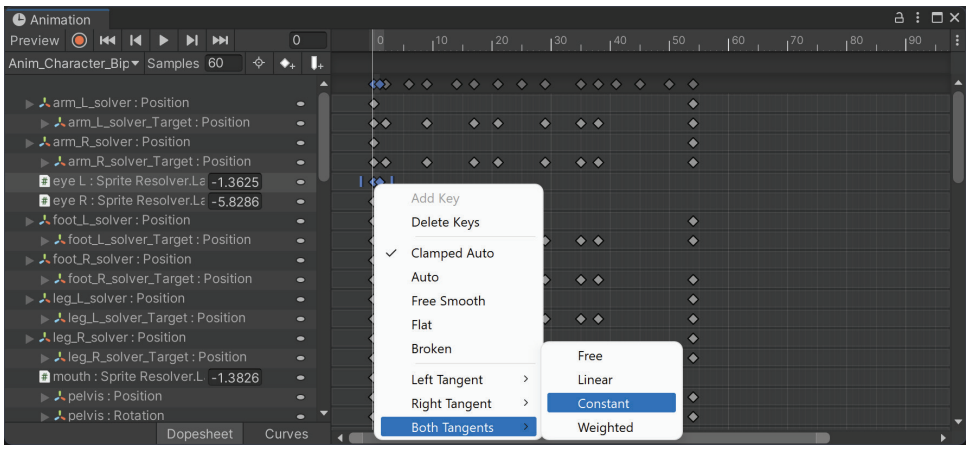
### スプライトスワップのアニメーション化 – 顔の表情

スプライトスワップのアニメーションも同様の進行となりますが、キーフレームの設定において、もう 1 つ手順が必要となります。




キーフレーム補間を無効にして Sprite Resolver をアニメーション化する方法

Sprite Resolver コンポーネントでスプライトを選択して、新しいキーフレームを設定します。デフォルトでは、キーフレーム間の値が補間されます。つまり、リストの最初のスプライトから後のフレームに含まれる最後のスプライトまで変化させる場合は、その間のスプライトはすべて表示されることとなります。Sprite Resolver では整数を使用して各スプラインのインデックスを定義するため、この場合、補間は望ましくありません。補間されないようにするには、Sprite Resolver の値を含むキーフレームをすべて選択し、右クリックして、「Both Tangents」>「Constant」を順に選択します。これでアニメーションのタイムラインは、キーフレームの後に値を補間するのではなく「保持」するようになるので、スプライトスワップのアニメーションが正しく再生されます。




Sprite Resolver のキーフレームに「Constant」接線を設定して、値が補間されないようにする

## 最適化のヒント

— 2D Animation のパフォーマンスを向上させるために、Burst パッケージをインストールする。

— スケルトンを作成するときは、必要以上のボーンやスプライトは使わないようにする。

— できるだけ頂点の数を少なくするようにして、スケルトンメッシュは簡略化する。

— アニメーターのカリングはオンにする。パフォーマンス節約のため、アニメーション化するキャラクターがオフスクリーンになるときには、ここのオプションはオフにします。

— 背景にいる鳥やハエ、小動物のような小さいオブジェクトや数が多くなるキャラクターには、スケルタルアニメーションを使わないようにする。代わりに、フレーム単位のアニメーションやシェーダーなどの別の技法を使用します。

# 2D Lights

Unity の高度な 2D 動的ライティングシステムを法線マップやマスクマップと交えて使用することで、鮮麗なシルエットライティングと細やかな部分まで明瞭に影がかけられて、キャラクターが人の目を引きやすくなります。

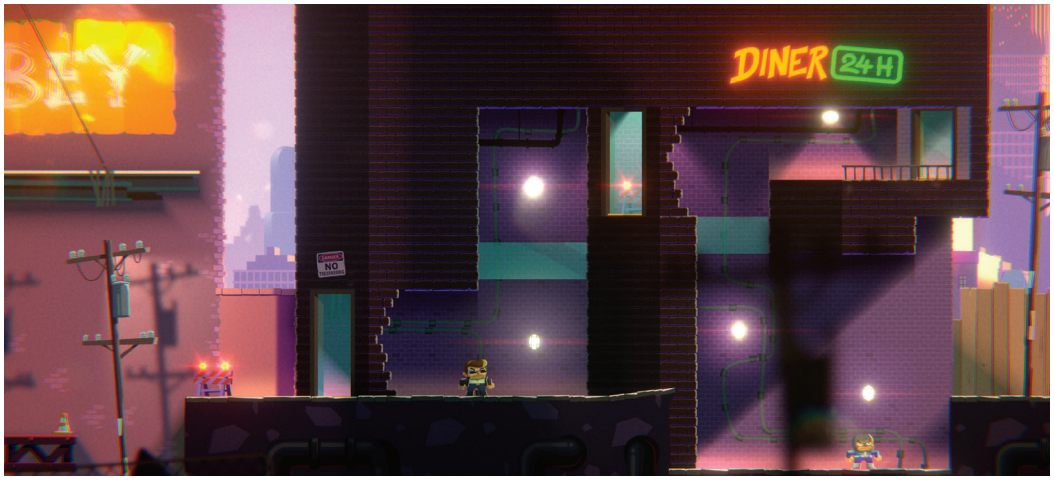
ライティング環境は、キャラクター、オブジェクト、ライティングシステムが緻密に相互で作用するものであるため、没入感とリアリティが高くなります。

2D Lights がゲームプレイに不可欠な要素になることもあります。例えば、2D Lights を使用して、プレイヤーが真っ暗な回廊の一部を照らすための懐中電灯を作成するとか、プレイヤーが監視カメラに写らないように回避しなければならない円錐形を描くといったケースがあります。

動的なライティングを使用すると、ステージの雰囲気を劇的に様変わりさせて、洞窟内のたいまつで照らされた壁のディテールを浮き上がらせたりすることができます。また、窓から差し込む光の筋で室内を舞うほこりを照らし、そのきらめきを表現したり、昼と夜のサイクルを模したアニメーションを作成したりする例も挙げられます。

以下では、作成できる 2D Lights の種類を紹介し、カスタマイズの方法を説明します。

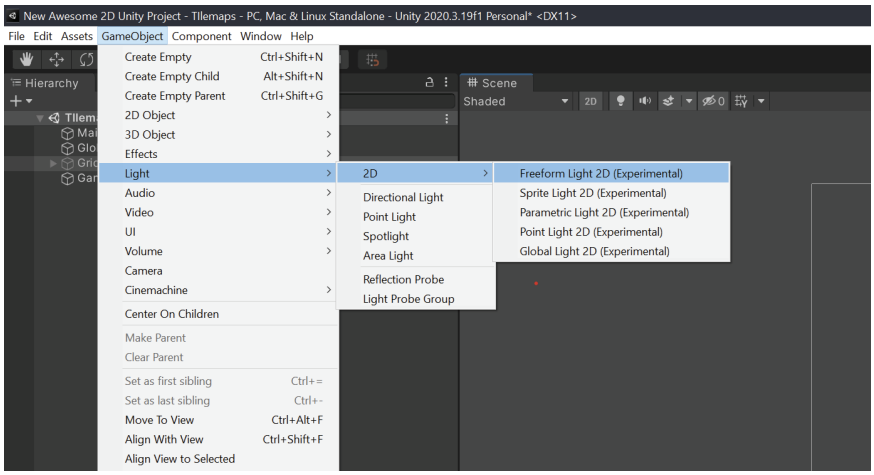
出典：『*Tails of Iron*』（Oddbug Studio 制作）




出典：Jarek Majewski が開発中のゲーム、『*Ultimate Action Hero*』

## 種類とユースケース

新しい 2D ライトを追加するには、**「GameObject」>「Light」>「2D」**の順に選択して、ライトの種類を選択します。

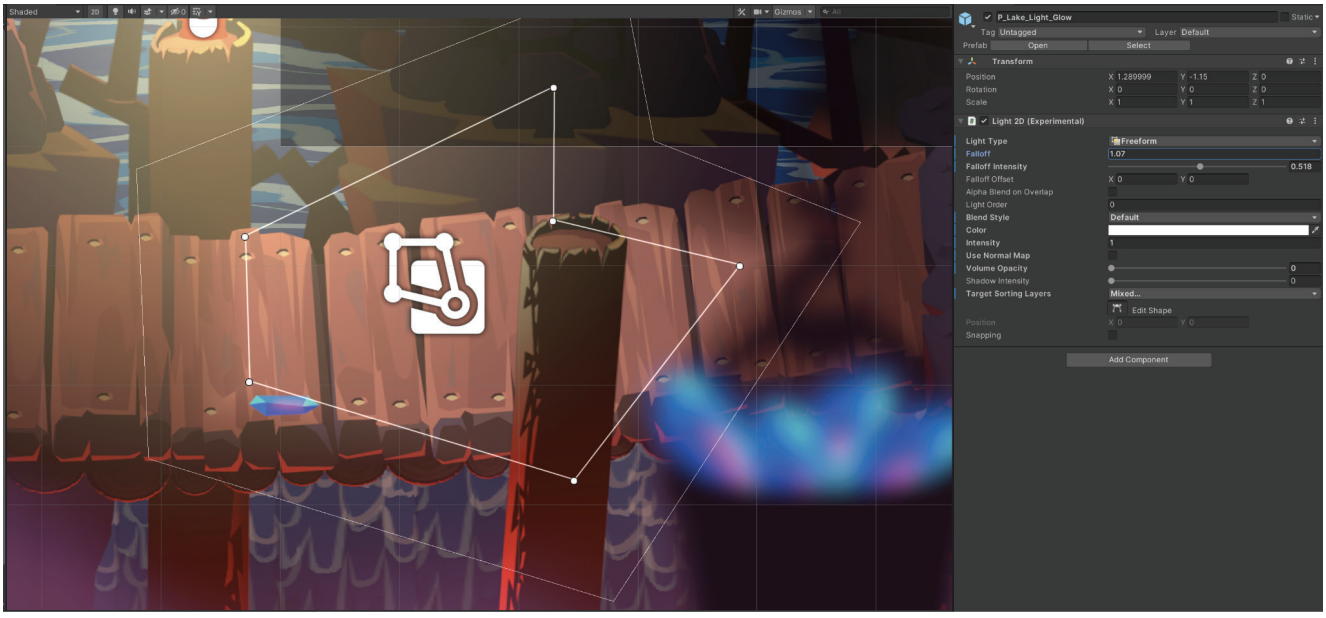



2D ライトをシーンに追加する

2D Lights には、5 種類の形状があります。

### Freeform

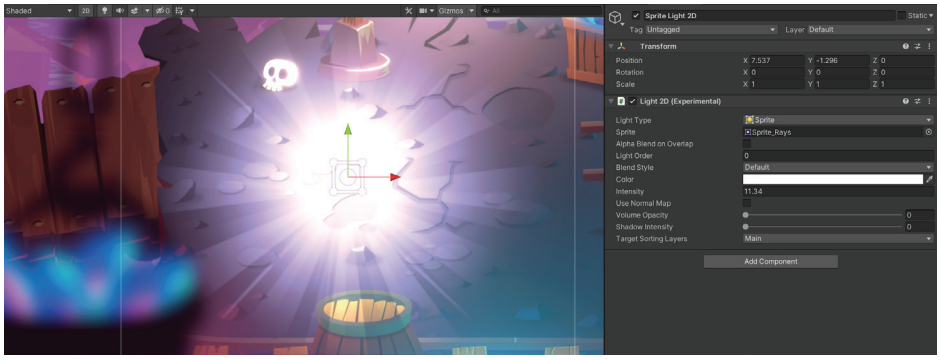
この種類のライトは多角形で Sprite Shape と同じように編集することができます。環境内の大きなパーツ（溶岩の海など）に効果的なライティングを施したり、各種の形状の光（天井の穴から降り注ぐ光「ゴッドレイ」など）を再現したり、窓の形に沿って光が当たっているところを表現したりする場合に良いツールです。




フリーフォーム 2D ライトを編集する

### Sprite

この形状を選択すると、スプライトをライトのテクスチャとして使用できます。他の種類では実現できない特定の形状のライトが必要な場合に役立ちます。良い使用例をいくつか挙げると、レンズフレア、グレア、ライトクッキーなどのほか、ミラーボールのような光の投影、幼児向けのランプが壁に投げかける星形の光などのテクスチャを作ることが可能です。




スプライトライトの動作

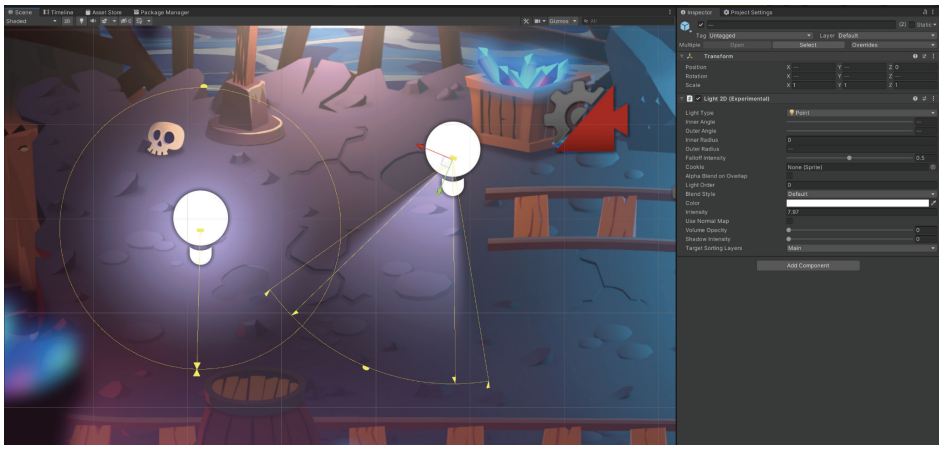
### Parametric

この種のライトは、n 角形の光を形成します。鉱物で構成された環境やスタイル化された環境に使用できる、特殊なライトです。この種類のライトは、将来の Unity バージョンで Freeform ライトオプションのデフォルト形状になる予定です。


3 角形、5 角形、8 角形のパラメトリックライト

### Point/Spot

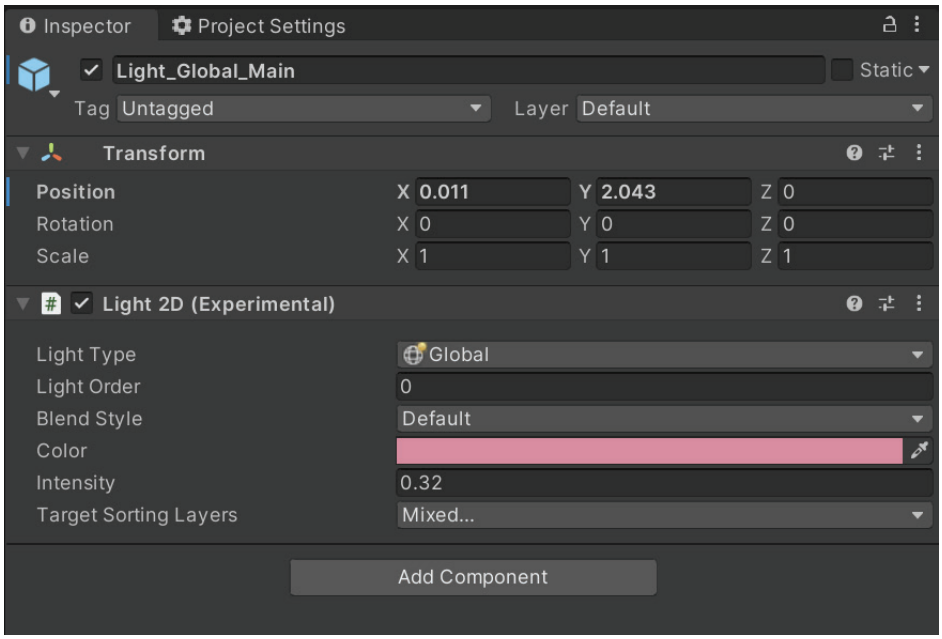
円または円の一部の形をしたライトです。このオプションは、特定の地点を照らすスポットライトに適しています。具体的にはたいまつ、ろうそく、車のライト、懐中電灯、ボリュメトリックライトなどの光です。



円形のポイントライトと円弧のポイントライト

### Global

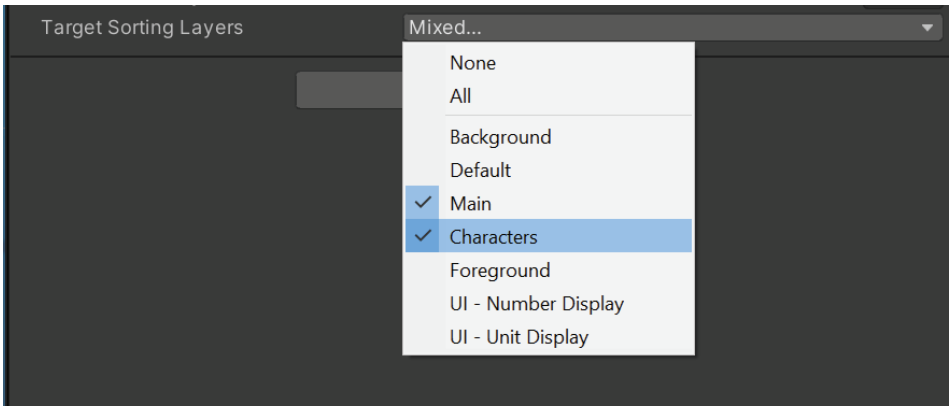
グローバルライトは特定の形状を持たずに、ターゲットのソートレイヤー上にあるすべてのオブジェクトを照らします。グローバルライトは、1 つのブレンドスタイル（ライトとスプライトが相互に作用する方式）につき 1 つだけ、また 1 つのソートレイヤーにつき 1 つだけ使用できます。ベースの環境光を追加するために、最初に使用します。




グローバルライトのオプション

## 2D Lights の使用方法

2D Lights はスプライトレンダラー、スプライトシェイプレンダラー、タイルマップレンダラーで機能します。ワークフローに合わせるために、ソートレイヤーを使用します。それぞれのライトは、1 つ以上のソートレイヤーに適用できます。「Target Sorting Layers」ドロップダウンリストで、適用するレイヤーを選択します。



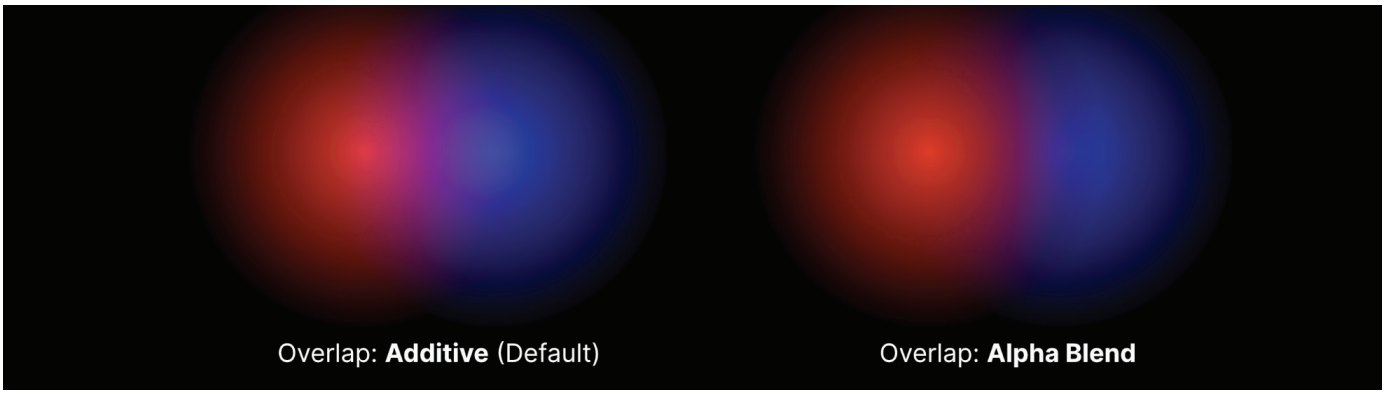

2D ライトの影響を受けるソートレイヤーを選択する

『*Tails of Iron*』（Oddbug Studio 制作）の 2D Lights の設定

同じソートレイヤーのライトのレンダリング順序を制御するには、「Light Order」オプションを使用します。

「Blend Style」オプションで、このライトで使用されるブレンドスタイルを選択できます。2D Renderer アセットでは、さまざまなブレンドスタイルをカスタマイズできます。各タイプのライトで使用される共通のプロパティについては、こちらをご覧ください。

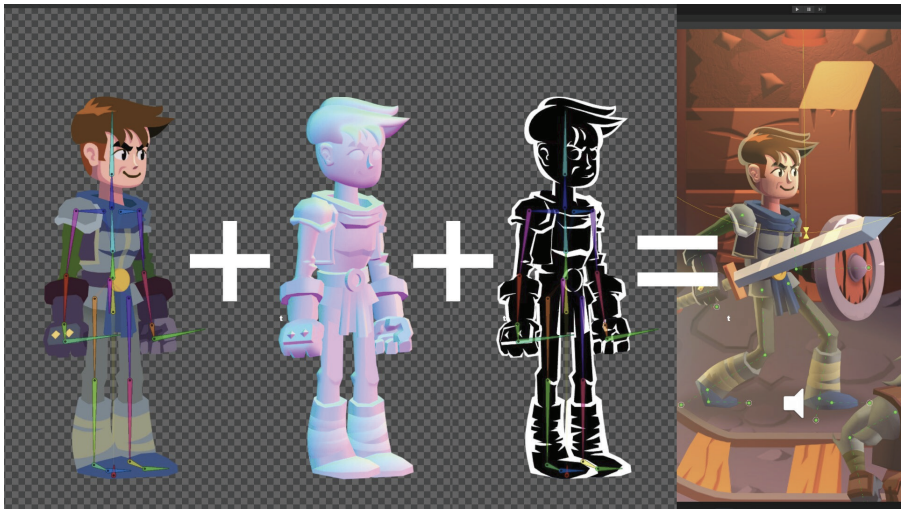
「Alpha Blend on Overlap」オプションは、強度を加算するのではなく、重なり合ったライトをブレンドします。このオプションでは 2D ライトを影に変換できます。ライトの強度を 1 より小さくして、明るい青色を選択するだけです。キャラクターにアタッチして、微細な影を投げかけることもできます。




2D Lights のオーバーラップモード

「Alpha Blend on Overlap」を使用して 2D ライトを影に変える

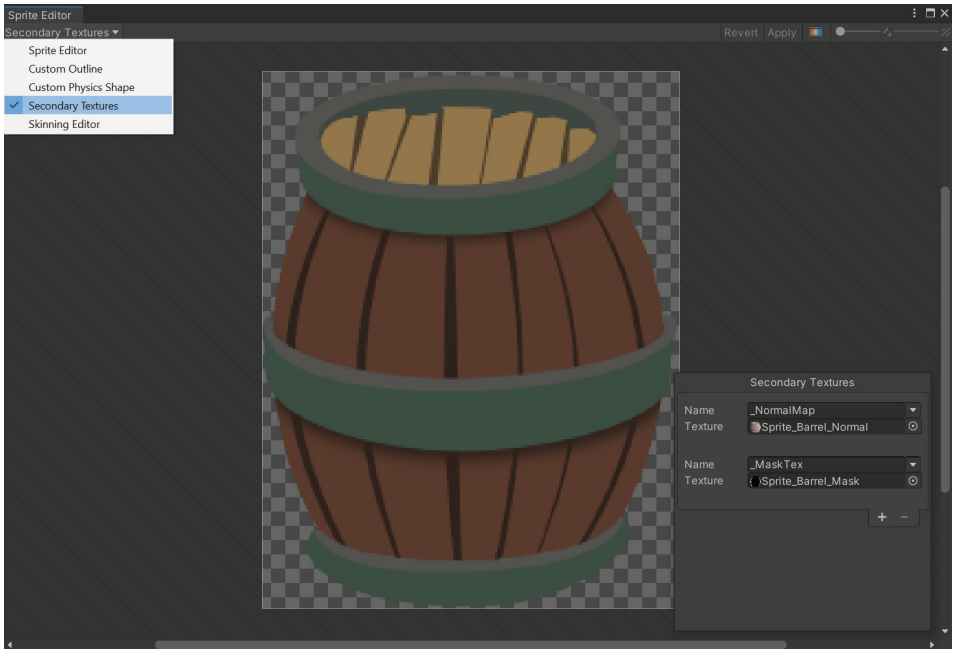
## 補助的なテクスチャ（スプライトマップ）



スプライトに Secondary Textures（補助的なテクスチャ）を追加する

法線マップとマスクマップは、2D Lights の使用時に必須の要素というわけではありませんが、これらのマップを使用して、予算や時間、アートに関するリソースを費やせば、ゲームの外観を今までとはまったく違うレベルに引き上げることができます。キャラクターと環境がより一層細やかに光を反映し、それらの形状がさらに三次元的に際立って見えるようになります。

どのスプライトアセットにも、任意の補助的なテクスチャを割り当てることができます。これらのテクスチャはマテリアルで使用できます。補助的なテクスチャを変更するには、**「Sprite Editor」>「Secondary Textures」**の順に選択します。デフォルトでは、法線マップとマスクマップという 2 種類のテクスチャを割り当てることができます。法線マップにはオブジェクトの全ピクセルの角度が含まれています。またマスクマップはリムライティングなどに利用できます。独自のテクスチャを追加し、それらをシェーダーグラフで名前で参照できます。

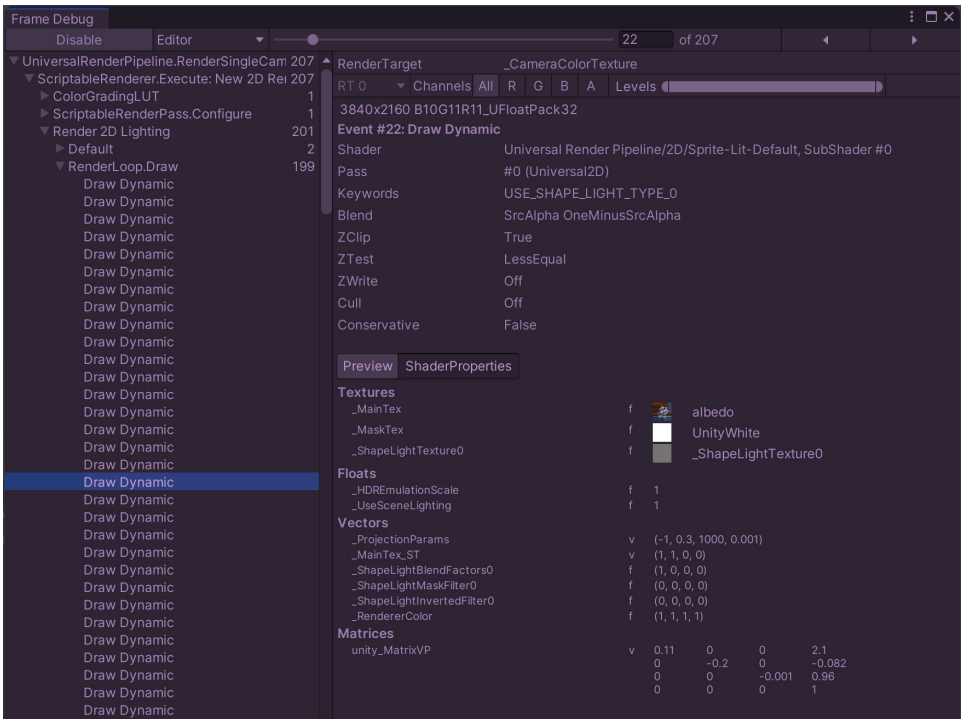



スプライトに Secondary Textures（補助的なテクスチャ）を追加する

いくつかの理由から、テクスチャはマテリアルレベルではなくアセットレベルで割り当てます。

メリットの 1 つは、スプライトレンダラーやスプライトを使用するその他のレンダラーが、たとえスプライトと補助的なテクスチャが異なる場合でも、マテリアルを共有できるという点です。つまり、バッチ処理で効率的にレンダリングを実行できます。

スプライトシェーダーを使用する場合は、法線マップとマスクマップをマテリアルに追加する必要があります。どのスプライトにも、スプライト、法線マップ、マスクマップを設定した独自のマテリアルが必要になります。そのため、すぐにプロジェクトに何百個ものマテリアルが存在するようになってしまいます。しかし、すべてのスプライトに追加のテクスチャを設定すれば、1 つのスプライトマテリアルを使うだけで済みます。マテリアルが 1 つということはドローコールも 1 回でよく、数百ものコールは不要となります。

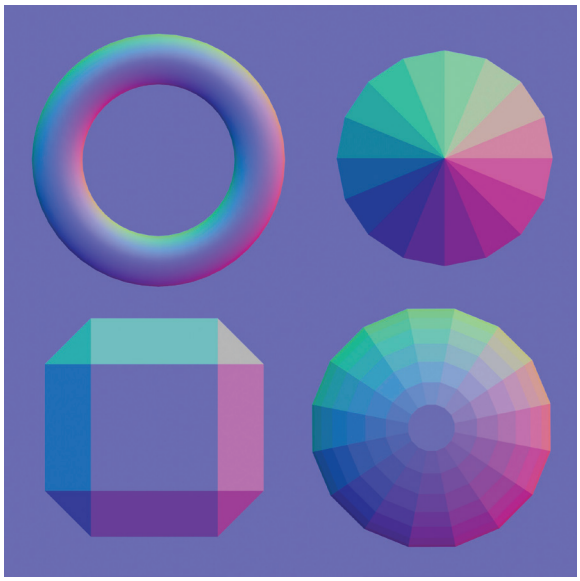



フレームデバッガーを使用すると、フレームをレンダリングするために実行されるすべてのステップを観察できます。バッチ処理できないマテリアルが多数あると、パフォーマンスが低下します。

## 法線マップ

よくできた法線マップは、スプライトを 3D に見せる効果を生むこともあれば壊すこともあります。法線マップのすべてのピクセルは、メインテクスチャの角度に関するデータを保持します。また赤、緑、青（RGB）の各チャンネルは、X、Y、Z 座標の角度データを保持します。法線マップを使用するライトにはいずれも方向があります。そのため法線マップを伴うテクスチャ上のピクセルは、この方向とピクセルの方向に基づいてシェーディングされます。これは現実世界と同じように作用します。つまり、ピクセルがライトの方向に向いているとそのピクセルはライティングされ、ライトの方向に向いていないとライティングされません。

RGB 値が法線マップの角度にどのような影響を及ぼすかを見てみましょう。



法線マップ

上の図は法線マップで、中のピクセルはカメラに面しています。RGB 値はそれぞれ 127、127、255 です。各カラーチャンネルは 0 から 255 までの値をとるため、127 は真ん中付近の値です。サーフェスを左に向ける（-90 度）には、R のカラー値を 0 に設定する必要があります。サーフェスを右に向けるには、R を 255 に設定します。まっすぐ上下させるには、G のチャンネルを上の場合は 255、下の場合は 0 に設定します。

### 2D ライティング用のスプライトの準備

ベースとなるカラースプライトを書き留めます。ゲーム内で 2D ライティングを広く使用して、法線マップを最大限に活かそうと考えている場合は、スプライトに光や影をペイントしないようにします。

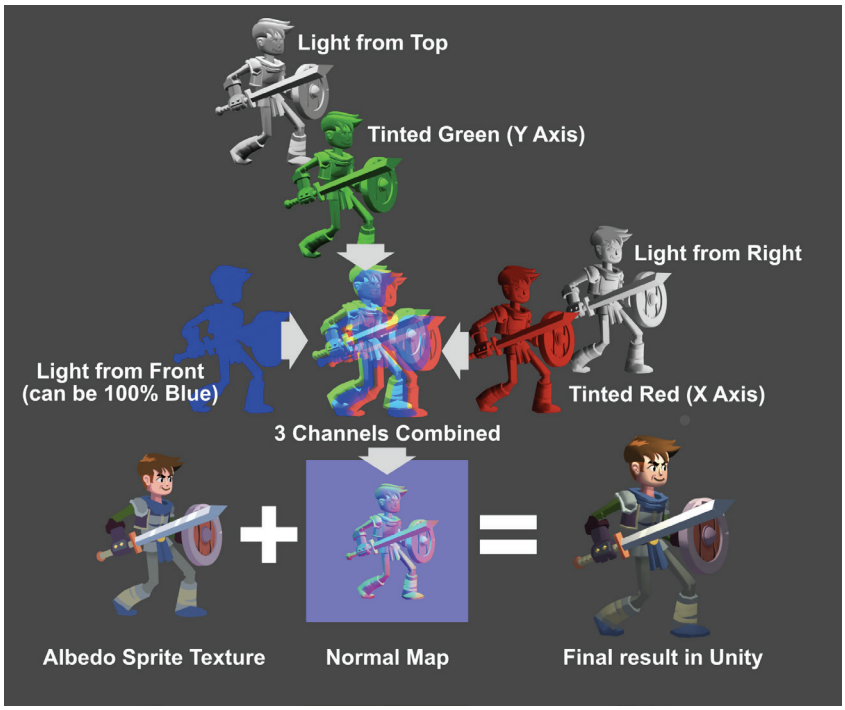
2D ライティングは影がペイントされているスプライトに適用しても、良い見た目になりません。また結局は、法線マップでライティングをペイントすることになるので、二度手間ということにもなってしまいます。代わりに方向性のない影をペイントすると、日光などのディレクショナルライトを当てない限り、スプライトの外観は良くなります。


2D Lights をオフにすると、スプライトに色情報（アルベド）は含まれますが、光や影の情報は含まれないため、平坦な見た目になります。

### 法線マップのペイント手法

法線マップをペイントする手法の 1 つは、複数の角度から照らしたスプライトの図を作成し、それらを組み合わせて 1 つのテクスチャを合成することです。すると、スプライトは 1 つのライトでは R チャンネルで右側から照らされ、また別のライトでは G チャンネルで上方から照らされます。B チャンネルでは、スプライトは正面から照らされますが、簡素にするために、2D スプライトで法線マップを使用する場合は、このチャンネルを省略してもかまいません。なぜなら、2D で正面からライティングしても全体のシェーディングは大きく変わらないからです。

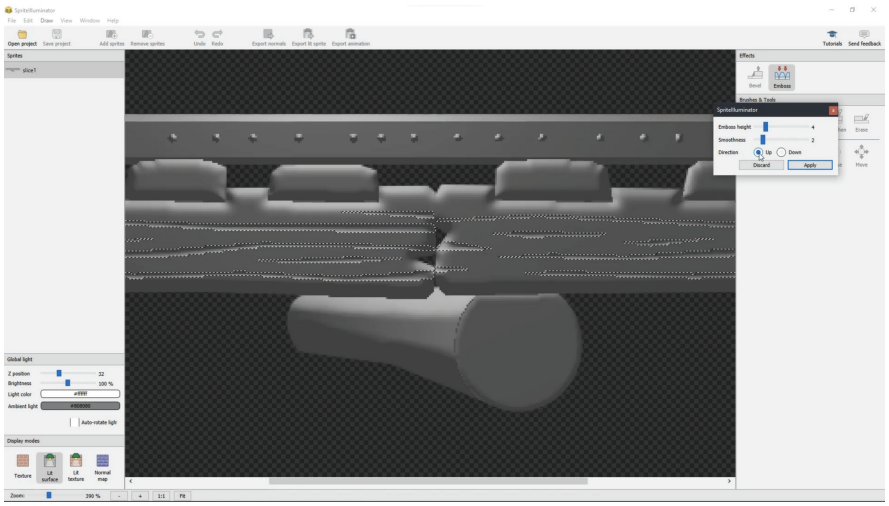
ただし、この方法は最低でも X 軸と Y 軸の 2 回はシェーディングをペイントする必要があるため、時間がかかることがあります。




影付きの 2 つのグレースケール（上から照らされたものと右から照らされたもの）画像を組み合わせて法線マップを作成する方法

法線マップジェネレーターのアプリケーションを使用して、ペイントする方法もあります。ジェネレーターアプリケーションでスプライトを開いて、何回かクリックするだけで法線マップを生成できます。ジェネレーターアプリケーションはスプライトの角度を考慮しないので、スプライト全体をこの方法で処理することは避けてください。同様にオブジェクトも認識しません。その代わりに、ジェネレーターはスプライトの色を参照することで、もしくは画像編集アプリケーションのベベルやエンボス、あるいは浮き彫りに類似した汎用フィルターを追加することで、形状を推定します。例えば、顔の角度を認識することはできませんが、どの部分で角度が変化しているかを推定しようと試みます。限界はありますが、ジェネレーターは鎖、ケーブル、龍の尻尾のような斜めの面を含むスプライトの法線マップを生成する場合に有用であり、レンガ、石、木などのサーフェス法線の生成にも活用できます。

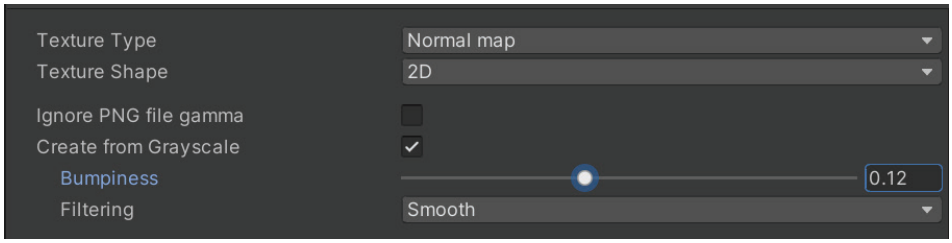
法線マップジェネレーターにセクションをインポートし、値を調整してからエクスポートして、その後に必要なパーツやディテールを自分で追加します。



Sprite Illuminator の Emboss ツールを使用して法線マップに割れ目を生成する

最後に取り上げるのは、グレースケールのハイトマップから法線マップを生成する Unity の機能を利用する手法です。これは黒い部分はサーフェスの高さが最小であることを表し、白は最大であることを表すテクスチャです。画像を法線マップとしてインポートし、「Create from Grayscale」オプションをオンにする必要があります。この手法は作業しているエンジン内にいながら、手軽に法線マップを生成できて便利です。

この手法を使用する場合は、インスペクターに「Bumpiness」スライダーが表示されます。Unity はピクセルの明度を使用して、高さの違いを法線マップの角度に変換します。このスライダーはそれらの角度の険しさを制御します。「Bumpiness」の値が低いと、ハイトマップでコントラストが強くても、緩やかな角度や凹凸に変換されます。




「Create from Grayscale」オプションを使用して、テクスチャを法線マップに変換する

**法線マップをスプライトにペイントする方法**

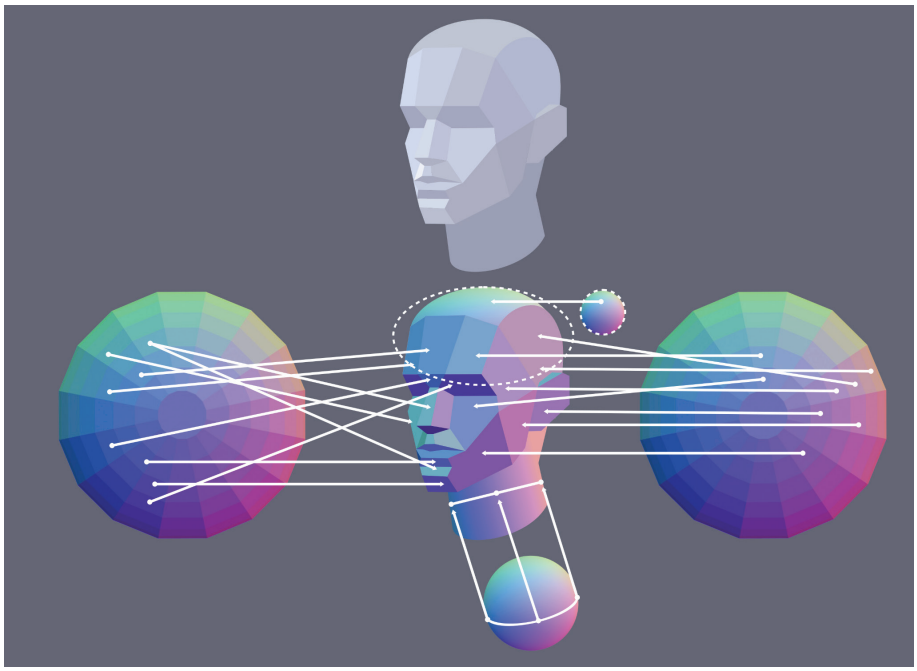
法線マップをペイントするには、どんな角度がどの色で表されるかを理解することから始めます。

最初に、法線マップパレットを入手して、サーフェス角度を表す色を採取できるようにします。インターネットでパレットを探してもいいですし、または単純に法線マップの章の画像にある色のパレットを使用してもかまいません。ペイントのワークフローがどのようなものかにかかわらず、パレットをコピーしてお使いのペイントアプリケーションにコピーしてください。そしてカラーピッカーを使って、法線マップにペイントする色を選択します。

角度を表す色は 100% 正確でなくても構いません。2、3 度ずれていても違いはありません。ただし、スプライトの全体的な形状はそれらしいものになっている必要があります。コンテキストに合わない色を使うと、ライトを当てたときに全体の形状が崩れてしまいます。

法線マップをペイントするには優れた空間想像力が必要であるため、最初は難しい可能性があります。頭部のベース平面のようなシンプルなものから始めるとよいでしょう。この人間の頭部の簡素化されたモデルは、ポリゴン数が少ない外観になっています。この用意された例は、こちらで確認できます。

法線マップをペイントするときは、スプライトを構成する基本的な 3D 形状を想像してみてから、個々のパーツの角度を視覚化してみましょう。その角度がわかれば、パレットのスプライトのどの部分から色をサンプリングすればよいのかわかります。



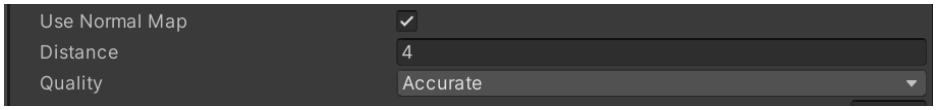
法線マップパレットから角度をサンプリングする方法

この例では平面に対して作業を行っていますが、より柔らかいブラシでペイントする場合もプロセスは同様です。ハードエッジをブレンドすると、外観をもっと自然に表現できます。

便利な方法を 2 つご紹介します。球形がある場合は、パレットから法線球体を貼り付けることができます。円筒形がある場合は、その球体の一部を取り出して貼り付け、それを引き伸ばしたり、グラデーションにしたりすることができます。

法線マップの一部をコピーして貼り付け、回転させると、シェーディングが崩れることに注意してください。ただし、これを利用することもできます。例えば、凹状の球形が必要な場合は、球体を 180 度回転させるだけで穴を作ることができます。

1 つ以上のスプライトに法線マップを表示するには、ライトの法線マップ機能を有効にする必要があります。




2D ライトの法線マップサポート設定

法線マップを生成するときは、一番使いやすい方法を選んでください。ほとんどの場合、ゲームのために多数のアセットを作成することになるので、最も目立つくオブジェクトに注力し、ゲームの他の部分は簡素化します。

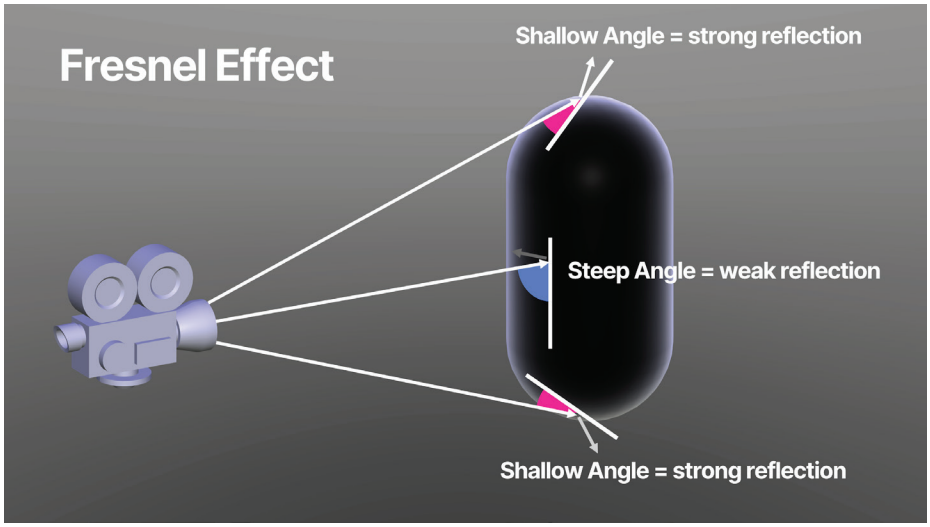
エンジン内の白いスプライト上での法線マップの外観

以下のツールが役立ちます。

- NormalPainter
- Krita の Tangent Normal Brush
- SpriteIlluminator
- Laigter
- Sprite Lamp

## マスクマップを使用したリムライティングの追加

リムライティングは、キャラクターの輪郭を強調するために使用されるエフェクトです。オブジェクトの背後からの光と、光散乱の自然な特性のシミュレーションを行います。これはフレネル効果と呼ばれます。現実世界では、オブジェクトの表面に光が当たる角度が浅いほど、そのオブジェクトから反射される光が多くなります。

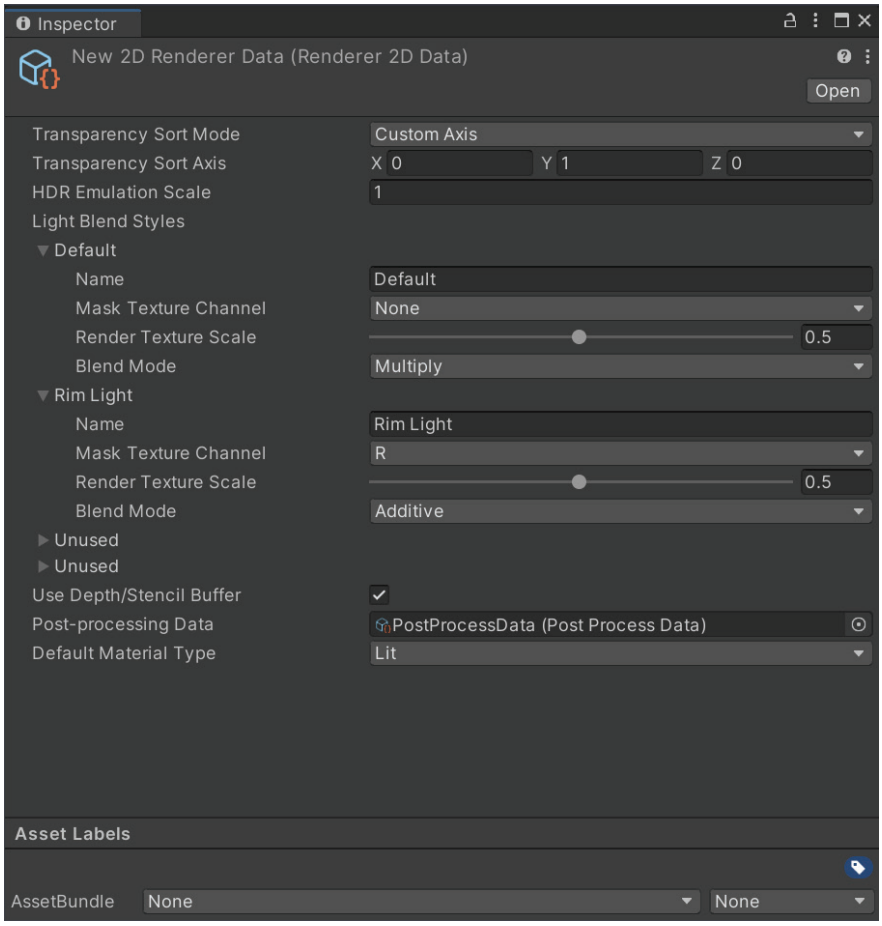



フレネル効果の図

そのため、湖を遠くから眺めると水面が反射して見えますが、水面に足を入れて上から見下ろすと、水中にある足が見えます。オブジェクトは端の方が反射することが多いのも、同様の理由からです。

2D グラフィックスでは、「マスクマップ」と呼ばれる追加のテクスチャと、「ブレンドスタイル」と呼ばれる特殊なライトを使用することで、この効果のシミュレーションを行うことができます。

ブレンドスタイルによって、特定のライトがシーン内のスプライトとどのように相互作用するかが決定されます。「Light Blend Styles」という設定が 2D Renderer アセットにあります。




2D Renderer Data アセットの「Light Blend Styles」のオプションを設定する

「Light Bend Styles」には、異なるブレンドスタイルの 4 つのオプションがあります。1 つ目のオプションはデフォルトなのでそのままにしておき、2 つ目のオプションを操作します。

「Rim Light」や「Fresnel」など、わかりやすい名前を付けます。「Mask Texture Channel」を「R」に設定して、ライトでマスクマップテクスチャの赤色が使用されるようにします。最後に、「Blend Mode」を「Additive」に設定します。これにより、既存のライティングの上にライトが追加され、明度が上がります。

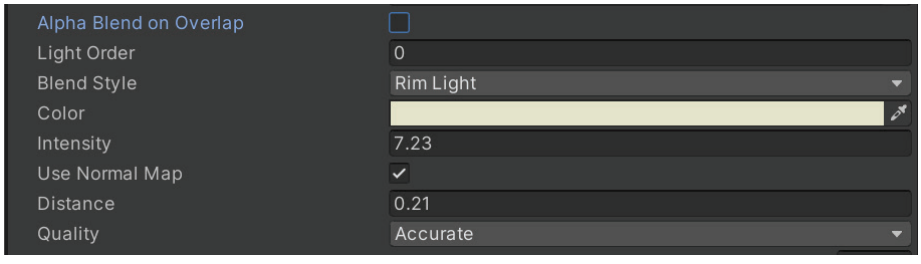
### マスクマップのペイント

フレネルライトではマスクマップの赤チャンネルを使用しますが、わかりやすくするために白黒でペイントしましょう。光を反射する部分は白になり、反射しない部分は黒になります。フレネル効果はオブジェクトのエッジに影響を与えるので、ベーススプライトをコピーして黒でペイントし、エッジを白でハイライトします。これで、背後から明るい光が当たったオブジェクトのように見えてきます。作業をスピードアップするために、オブジェクトに Inner Glow エフェクトを追加し、その上にディテールをペイントします。残念ながら、このプロセスを迅速化できるアプリケーションはないため、ここではペインティングスキルに頼るしかありません。

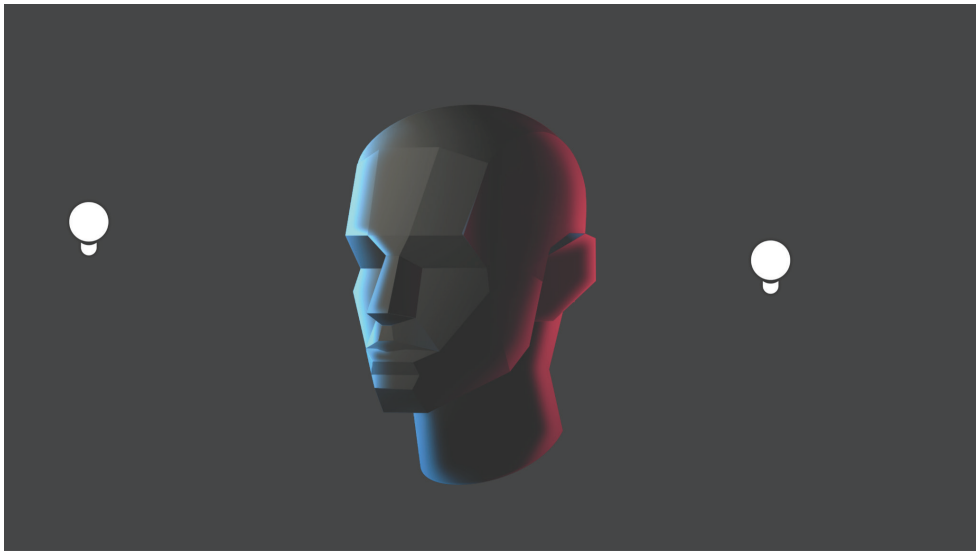
ベース平面の頭部スプライト用のマスクマップ

### フレネルライトの設定

ライトがマスクマップに影響を与えるようにするには、ブレンドスタイルを先ほど作成したものに変更します。




ライトの「Blend Style」を「Rim Light」に変更する

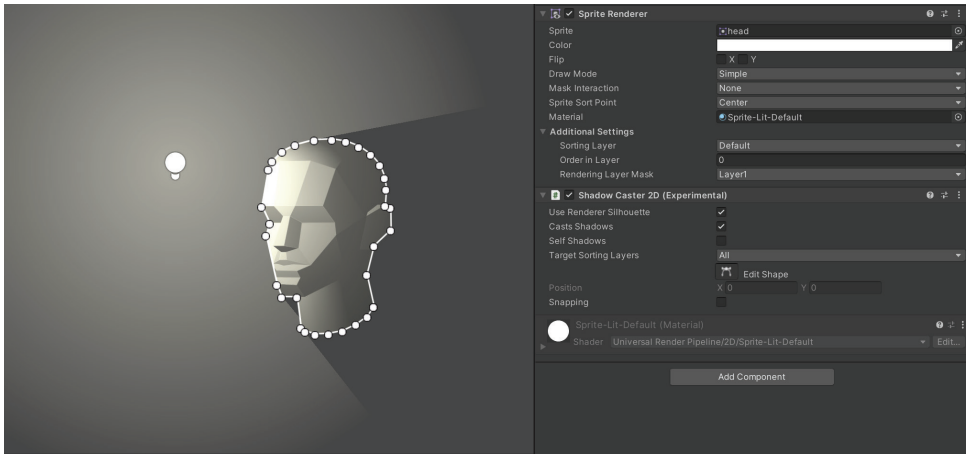


Shadow Caster 2D コンポーネントで形状を編集する

フレネルライトは、「Use Normal Map」オプションを有効にし、「Distance」を低い値に設定することで最も効果を発揮します。これにより、オブジェクトの反対側がハイライトされなくなります。

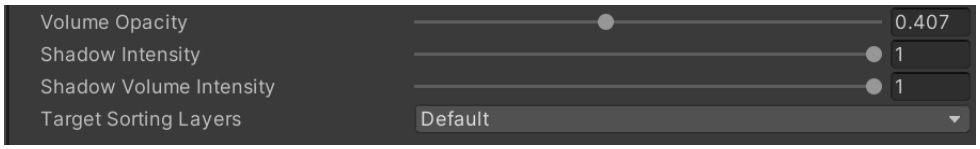
## 2D Lights による 2D の影の追加

2D Lights では、シーンに影を落とすこともできます。ゲームオブジェクトが 2D ライトから影を投影するには、オブジェクトに Shadow Caster 2D コンポーネントを追加します。




Unity エディターのスプライトのリムライティングエフェクト

Shadow Caster のシェイプエディターを使用して、オブジェクトのシルエットに合うようにジオメトリを編集します。また、ライトには、影を投影するためのオプションが設定されている必要があります。




影を使う場合に使用する 2D Lights のオプション

影を表示するには、「Shadow Intensity」オプションを使用します。この設定は、影の中にある他のオブジェクトに対するライトの不透明度を決定します。1 に設定すると、ライトは影の領域内のものを照らさなくなります。

「Volume Opacity」オプションを使用して、光の影響を受ける領域を可視化することもできます。「Shadow Volume Intensity」は、影の領域のライトボリュームの不透明度を制御します。


『*Among Us*』(InnerSloth 制作)。影はゲームの外観を向上させるだけでなく、ゲームプレイの非常に重要な部分になる場合もあります。

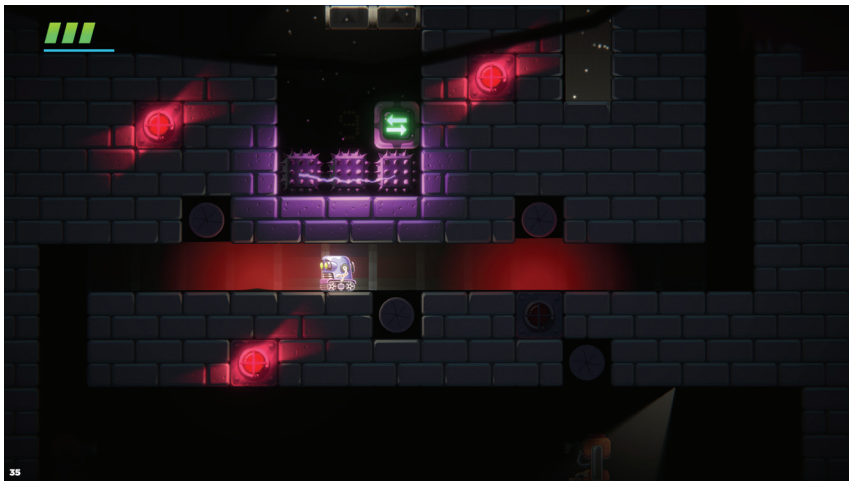
## さまざまなレンダラーでの 2D Lights の使用方法

2D Lights は、スプライトの法線マップやマスクマップと一緒に使用できますが、さまざまなレンダラーで使用することもできます。その方法を見てみましょう。

### 2D タイルマップ

タイルマップで 2D Lights を使用するには、Sprite エディターの Secondary Textures モジュールを使用して、タイルとして使用するスプライトに法線マップテクスチャとマスクマップテクスチャを割り当てます。2D Lights システムで、その補助的なテクスチャが自動的に使用されます。

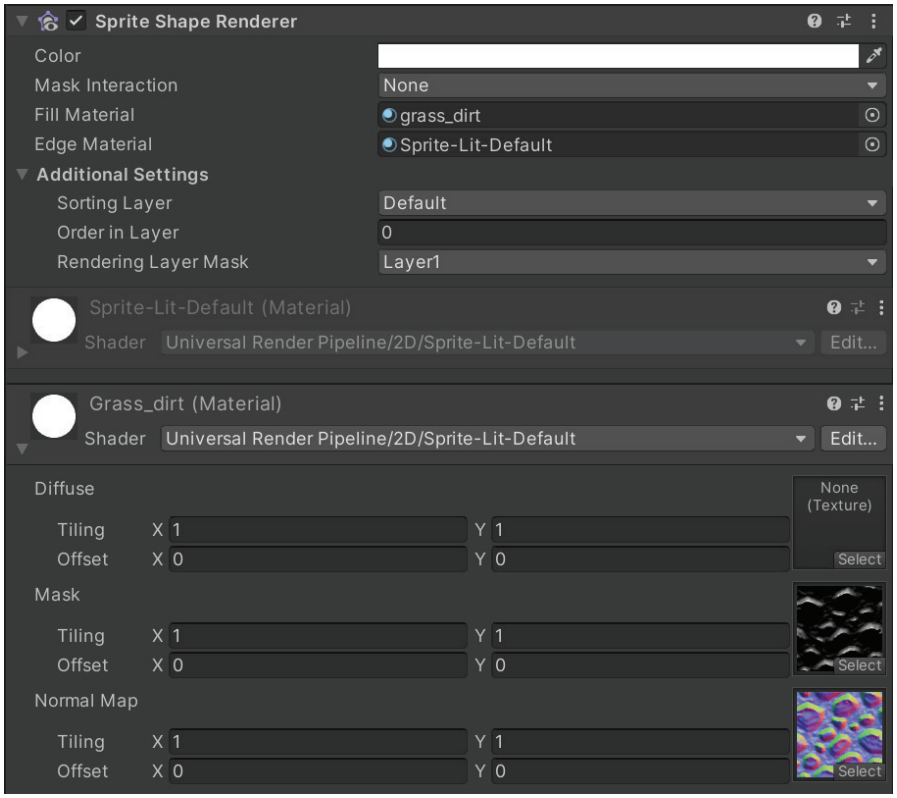

*セカンダリマップを使用したタイルマップ*



2D タイルマップに法線マップとマスクマップを適用した 2D Lights

### 2D Sprite Shape

スプライトシェイプレンダラーでは、オブジェクトごとに 2 種類のマテリアルを使用します。1 つは塗りつぶし用、もう 1 つはエッジ用です。エッジマテリアルでは、スプライトアセットのセカンダリマップを使用します。塗りつぶし領域では、スプライトの代わりに、「Wrap Mode」を「Repeat」に設定したテクスチャをタイリングに使用します。補助的なテクスチャを設定するために、マテリアルプロパティブロックを有効にした新しい塗りつぶしマテリアルを作成する必要があります。




法線マップを使用した Sprite Shape ゲームオブジェクトの例

法線マップを使用した Sprite Shape ゲームオブジェクトの例

### アニメーション化されたキャラクター用の 2D PSD インポーター

2D PSD インポーターでの補助的なテクスチャを設定しても同様に機能しますが、法線マップに関しては、わずかな違いが 1 つあります。

2D PSD インポーターでインポートされたキャラクターの法線マップとマスクマップを作成する最も早い方法は、ベースキャラクターの .PSB ファイルを使って作業を開始することです。

1. アニメーション化されたキャラクターに法線マップを追加するには、以下の手順を実行します。

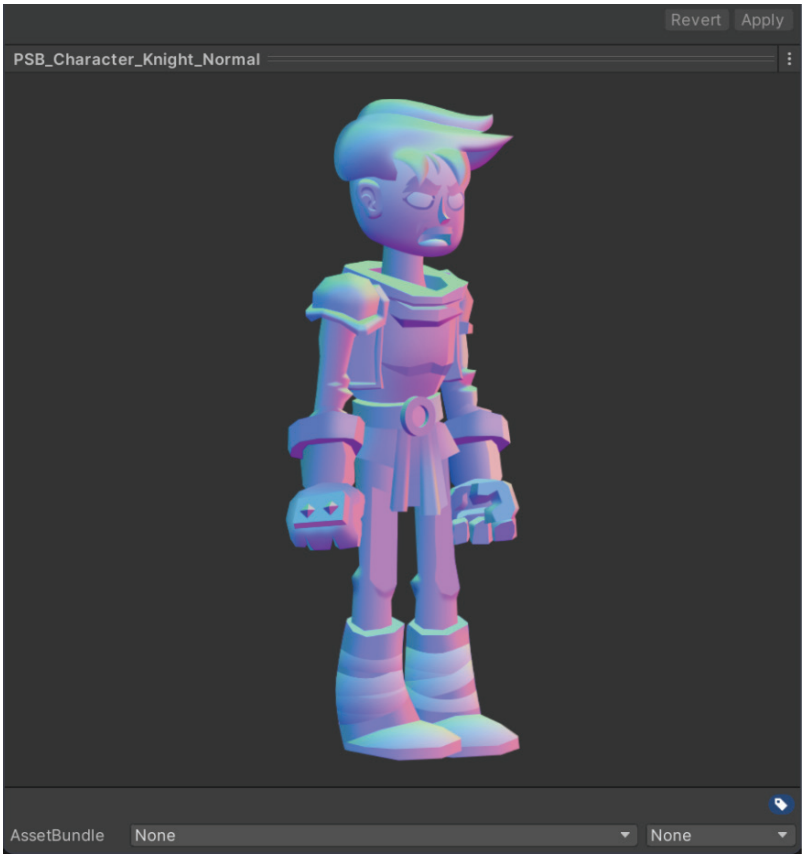
2. ベースキャラクターの .PSB ファイルを複製します。複製したファイルの名前にサフィックスを追加して、ファイル名を変更します（例：_normal）。

3. このファイルを任意の画像エディターで開き、各レイヤーに法線マップを作成します。ファイルを保存します。

4. 法線マップを含む PSB を Unity にインポートする場合、「Texture Type」を「Sprite」に設定し、「Advanced」設定に移動して「sRGB (Color Texture)」オプションをオフにする必要があります。法線マップには色の sRGB データは含まれておらず、角度値のみが含まれています。

5. ベースキャラクターの補助的なテクスチャとして、この PSB ファイルを割り当てます。

マスクマップを作成するには、このプロセスを繰り返し、複製したファイルに別のサフィックスを付けて、手順 3 をスキップします。




アニメーション化されたキャラクターに法線マップを追加する

## 最適化のヒント

- 1 つのシーンで使用するライトが多すぎないようにします。
- 使用するソートレイヤーとライトブレンドスタイルをできるだけ少なくします。
- 2D Renderer アセットの「Light Blend Styles」の「Render Texture Scale」に小さい値を設定します。
- 法線マップを使用するときは、法線ライティングのライトの「Quality」を「Accurate」ではなく「Fast」に設定します。
- 2D Shadows は少数のライトにのみ使用します。

# 高度なビジュアルエフェクト

ビジュアルエフェクト（VFX）は見栄えの良いゲームの最後の仕上げとなるものであり、ゲーマーのプレイ体験に不可欠です。VFX は、環境の危険や回復ゾーンなどのゲームイベントを伝えます。また、激しい大爆発で終わるアクションシーンのように、うまくいったアクションに対してプレイヤーに視覚的に報酬を与えます。

Unity には、ゲームに VFX を追加するための多くのオプションが用意されています。他のソフトウェアで作成されたフリップブックアニメーションの追加、パーティクルのアニメーション化、アニメーション化されたシェーダーの追加を行うことができます。

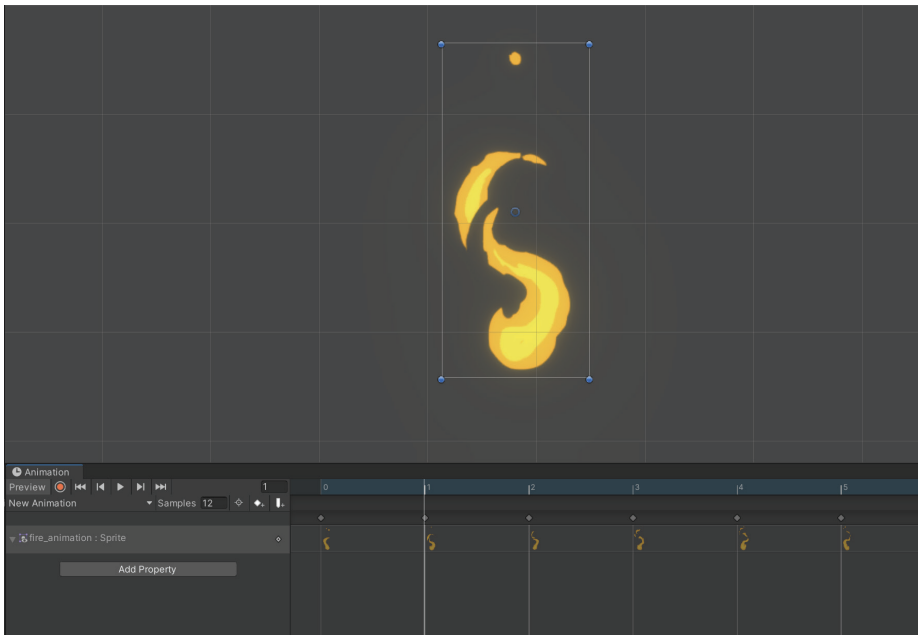
## フリップブックアニメーション

フリップブック（パラパラマンガ方式）アニメーションを使用して、Unity でエフェクトをすばやく追加できます。フレームをスプライトとしてインポートし、「Animation」ウィンドウで Sprite プロパティを時間の経過に合わせてアニメーション化するだけです。また、全フレームスプライトを選択してシーンへドラッグすることで、自動的にアニメーション化させることもできます。

フレームのインポートは高速ですが、アニメーション化する各フレームの描画はそうではありません。このプロセスには身を削るほどの大変な時間と技量が求められることすらあるのです。代わりに他のアプリケーションからフレームとして VFX をエクスポートすると、時間を節約できます。

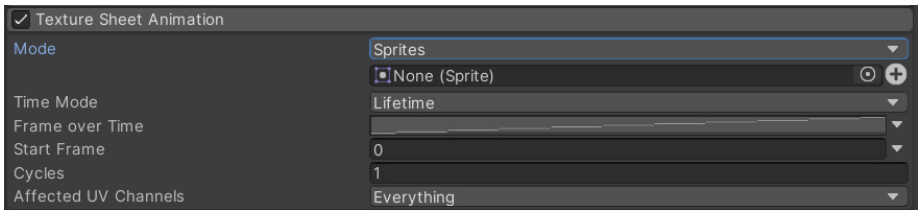
もう一つ負担の少ない代替案としては、「Project」ウィンドウから全アニメーションフレームアセットを選択して、それらを「Hierarchy」ビューまたはシーンへドラッグしてしまうことです。これにより、先ほど選択した画像で構成されたシーケンスを使用したアニメーションを含む、新規ゲームオブジェクトが作成されます。

フリップブックアニメーションの手法は大変に手間がかかるものであるため、画面上に多数のエフェクトインスタンスを表示する必要があるとなった場合に使用するようにしましょう。高解像度の多数のフレームとなると、より多くのメモリが必要となることに注意してください。



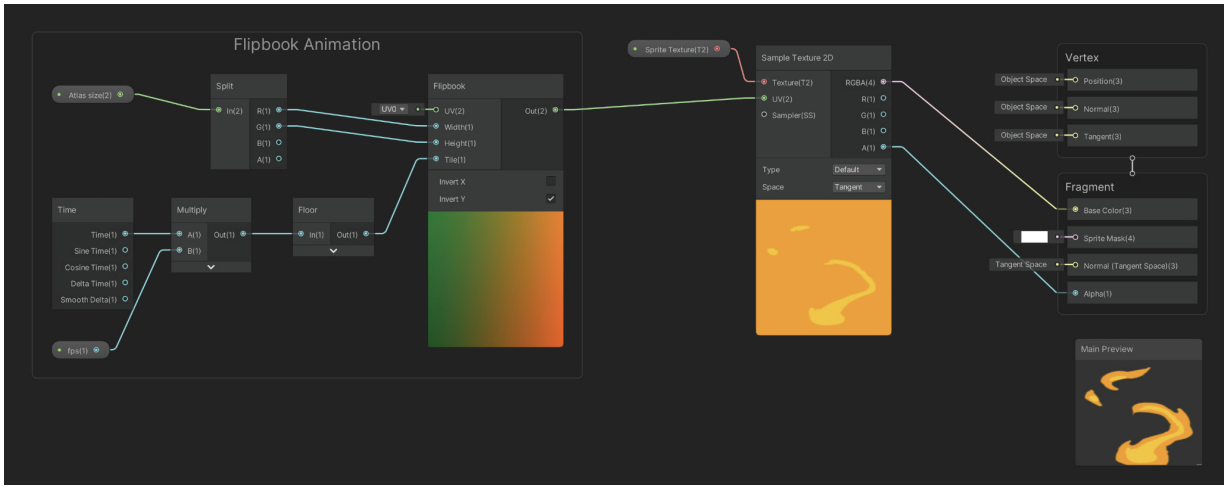

炎のフリップブックアニメーション

フリップブックアニメーションは、パーティクルシステムで使用することも可能です。パーティクルシステムでは、「Texture Sheet Animation（テクスチャシートアニメーション）」と呼ばれています。




パーティクルシステムでフリップブックアニメーションを設定する

フリップブックアニメーションは、シェーダーグラフで使用することもできます。UV 座標のテクスチャを時間経過で変わるアニメーション化にすることで、フレームが動いているように見えます。

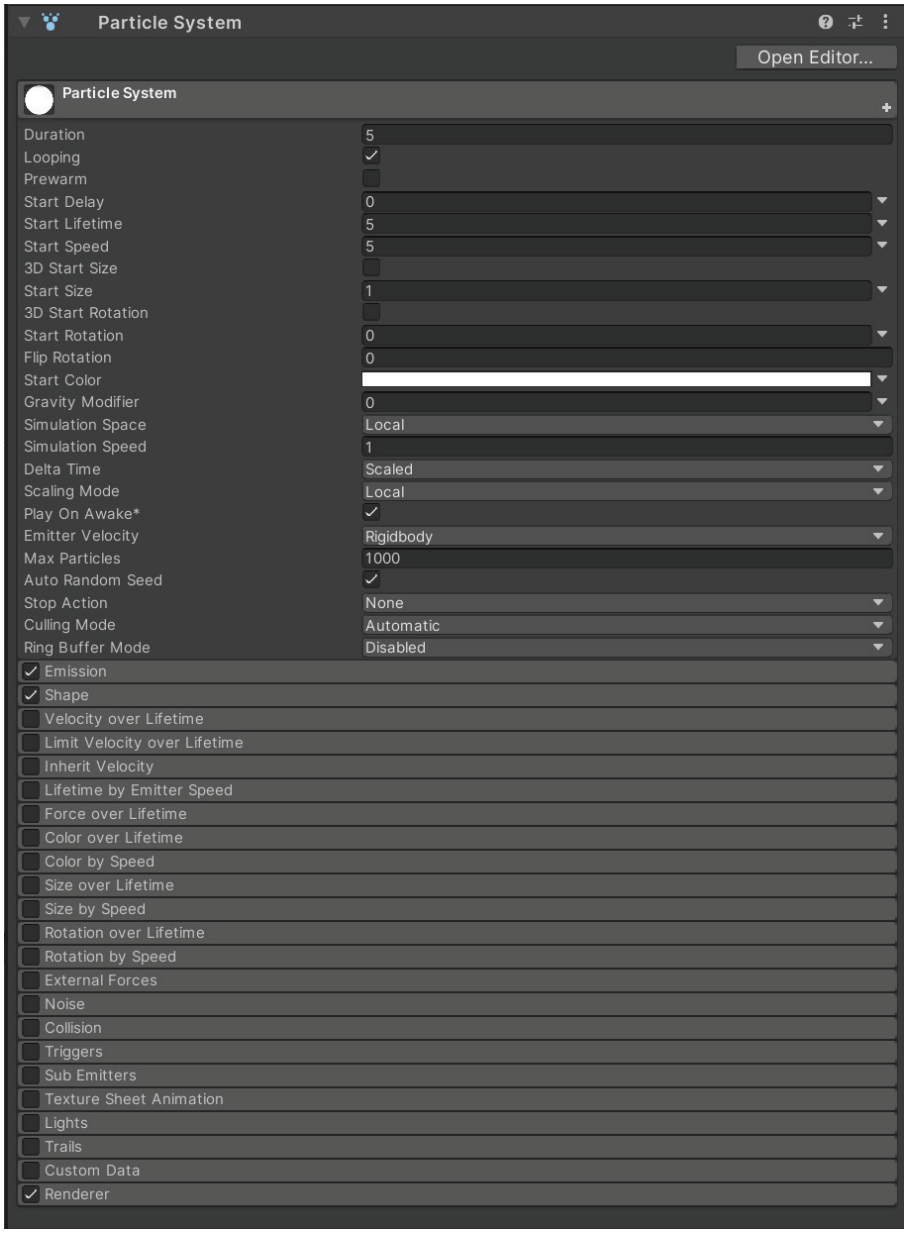



シェーダーグラフのフリップブックアニメーション

## パーティクル

パーティクルシステムを使用すると、1 つのビジュアルエフェクトを実現するために、多数の小さな画像やメッシュを表示したり動かしたりできます。サイズ、速度、色、回転などのパーティクルプロパティは、特定の定義済みのルールとランダム化を使用して、経時的にアニメーション化することができます。これにより、炎、爆発、煙、魔法などの動的エフェクトを作成できます。

**「GameObject」>「Effects」>「Particle System」**メニューオプションを選択して、新しいパーティクルシステムを作成します。




パーティクルシステムのプロパティ

### メインモジュールのプロパティ

パーティクルシステムには多くのモジュールがありますが、必須のものをいくつか見ていきましょう。

最初のモジュールには、基本的なパーティクルプロパティが含まれています。

— Duration：システムの実行継続時間

— Looping：システム全体が永続的にループするかどうか

— Speed：パーティクルの初期速度

— Start Size：パーティクルの初期サイズ

— Color：パーティクルの初期の色

— Gravity：パーティクルにかかる重力

#### Emission

Emission は放出されるパーティクルの速度とタイミングを制御し、Rate は単位時間あたりに放出されるパーティクルの数を制御します。スポーンする数やタイミングを指定して、パーティクルのバーストを作成することもできます。

#### Shape

このモジュールでは、パーティクルを放出できるボリュームまたはサーフェスと、パーティクルが移動する方向を定義します。Shape プロパティでは、放出ボリュームの形状を定義します。

#### Color over Lifetime

グラデーションの計算を行い、パーティクルの生存期間中の色と不透明度を制御することができます。グラデーションの左側が開始時の色で、右側がパーティクルの生存期間の終了時の色です。

#### Size over Lifetime

この設定では、カーブの水平軸に沿ったパーティクルのサイズを定義します。

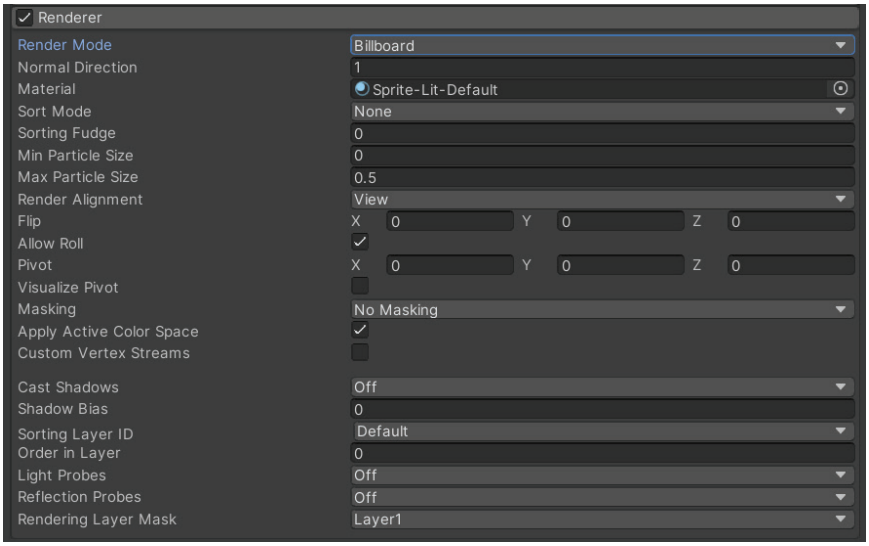
#### Rotation over Lifetime

Rotation over Lifetime では、パーティクルの回転速度を指定します。

#### Noise

このモジュールは、パーティクルの動きにランダム性を追加します。魔法、SF エネルギー、排気炎、塵など、凝った目を引くエフェクトを非常に簡単に追加できます。

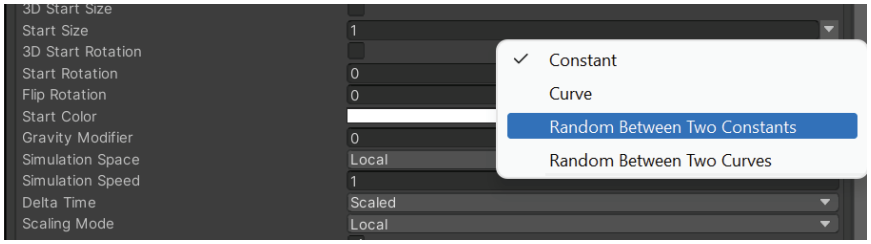
### Renderer




Renderer モジュール

上記の Renderer モジュールでは、パーティクルの見た目を変更できます。多くの設定がありますが、変更する最も重要な設定は「Material」設定です。この設定で、テクスチャを含む新しいマテリアルを追加できます。

### ランダム化されたパーティクル




Random パーティクルプロパティ

ほとんどのパーティクルプロパティは、予測不能でありながら自然な様子を実現するために、ランダム化することができます。

パーティクルプロパティの右側にあるドロップダウンメニューで、さまざまなランダム化オプションを選択できます。

パーティクルシステムは相互にネストすることもできます。ネストすると複数のパーティクルシステムが同時に再生されるので、非常に複雑なエフェクトを実現できます。例えば、爆発を作成する場合、煙のシステム、小さな火花のシステム、飛び散る破片のシステムを組み合わせることができます。

パーティクルシステムは、ゲームのほぼすべての VFX を作成できる非常に強力なツールです。HDR Unlit シェーダーとブルームポストプロセスを使用して、エフェクトを光らせることもできます。すべてのプロパティを試してみてください。どんなに単純なパーティクルでもゲームに磨きがかかります。

ゲーム制作プロセスを大幅にスピードアップするために、Asset Store で既製のパーティクル VFX をチェックしましょう。

## シェーダーグラフ

VFX を作成するもう 1 つの方法は、シェーダーを使用することです。シェーダーは、グラフィックスプロセッサーに対する一連の命令です。例えば、ピクセルの色や頂点位置を計算できます。

シェーダーグラフを使用すると、シェーダーを視覚的に構築できます。コードを記述する代わりに、グラフフレームワーク内にノードを作成して接続します。シェーダーグラフでは、変更が即座に反映されてフィードバックが得られるため、シェーダーの作成に慣れていないユーザーでも簡単に操作できます。

シェーダーグラフでは以下が可能です。

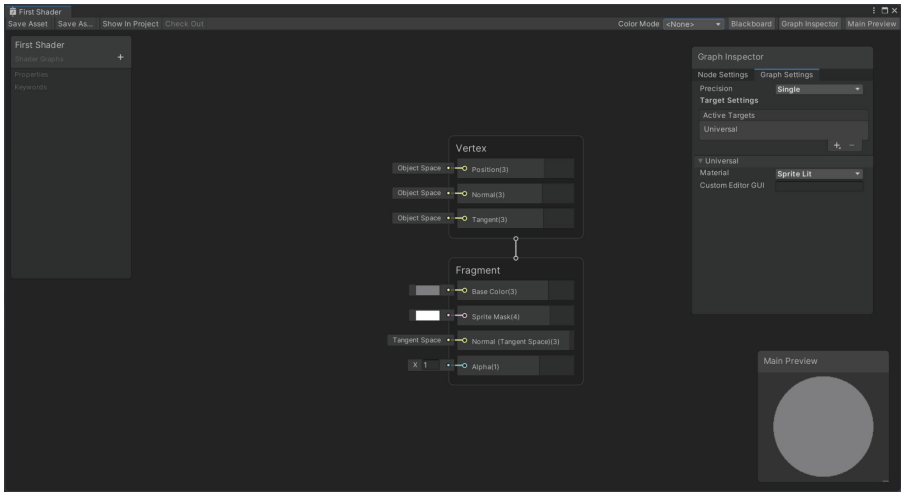
- UV のワープおよびアニメーション化を実行する
- サーフェスの外観を手続き的に変更する
- 画像編集アプリケーションと同様の広告画像フィルター
- ワールド空間での位置、法線、カメラからの距離などの情報に基づいてオブジェクトのサーフェスを変更する
- マテリアルのインスペクターにプロパティを公開して、シーンのコンテキストでシェーダーの外観を調整する

### シェーダーグラフの使用

2D 用シェーダーグラフの使用を開始するには、「Project」ウィンドウに移動し、右クリックして、**「Create」>「Shader」>「Universal Render Pipeline」>「Sprite Lit Shader Graph」**（または**「Sprite Unlit Shader Graph」**）の順に選択します。

スプライトに対してライトの影響を与えるには、「Sprite Lit Shader」を使用します。これに該当するのは、ほぼすべてのキャラクターやオブジェクトです。炎、光源、電気、呪文、ホログラム、さらに幽霊のような特殊なキャラクターなど、光が当たらないサーフェスまたはエミッシブサーフェスに対しては「Sprite Unlit」を使用します。

シェーダーの作成を開始するには、まずファイルに名前を付けます。これで初期シェーダーが作成されます。次に、ファイルをダブルクリックして編集モードに入ります。




シェーダーグラフウィンドウ

これにより、シェーダーグラフウィンドウが開きます。ワークスペースと呼ばれる中央部分が Master Stack です。これはシェーダーグラフの最終的な出力であり、シェーダーの最終的な見た目を決定します。Master Stack は、シェーダーごとに 1 つだけ存在します。

Master Stack には、頂点関数とフラグメント（またはピクセル）関数に対応する 2 つのコンテキストが含まれています。

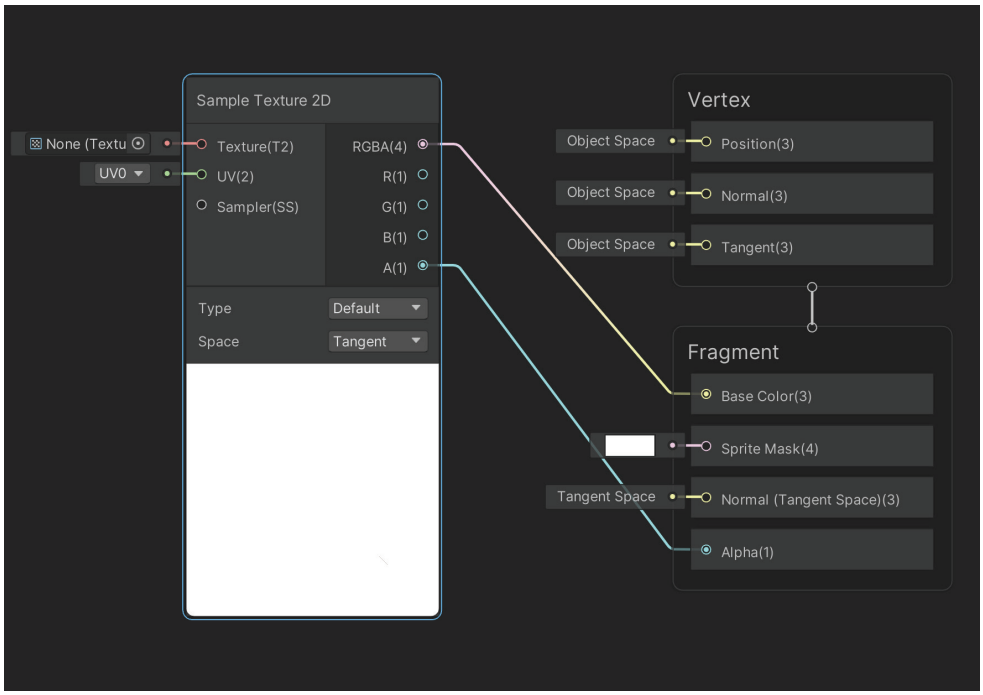
左側にはブラックボードが表示されています。ここに含まれるプロパティは、インスペクターとキーワードに表示されます。

右上の Graph Inspector には 2 つのタブがあります。「Node settings」タブでは選択したノードの設定を変更でき、「Graph Settings」タブではグラフ全体の設定を変更できます。Graph Inspector で、Precision（精度）モードを切り替えることができます。ローエンドデバイスでは、Half に設定することをお勧めします。シェーダーのマテリアルの lit と unlit を切り替えることもできます。

下部の「Main Preview」では、最終的なシェーダーを確認できます。

シェーダーグラフの各ノードは構成要素であり、入力、エフェクト、相互作用を保持します。グラフは左から右に進むので、ノードの左側に入力スロット、右側に出力スロットがあります。

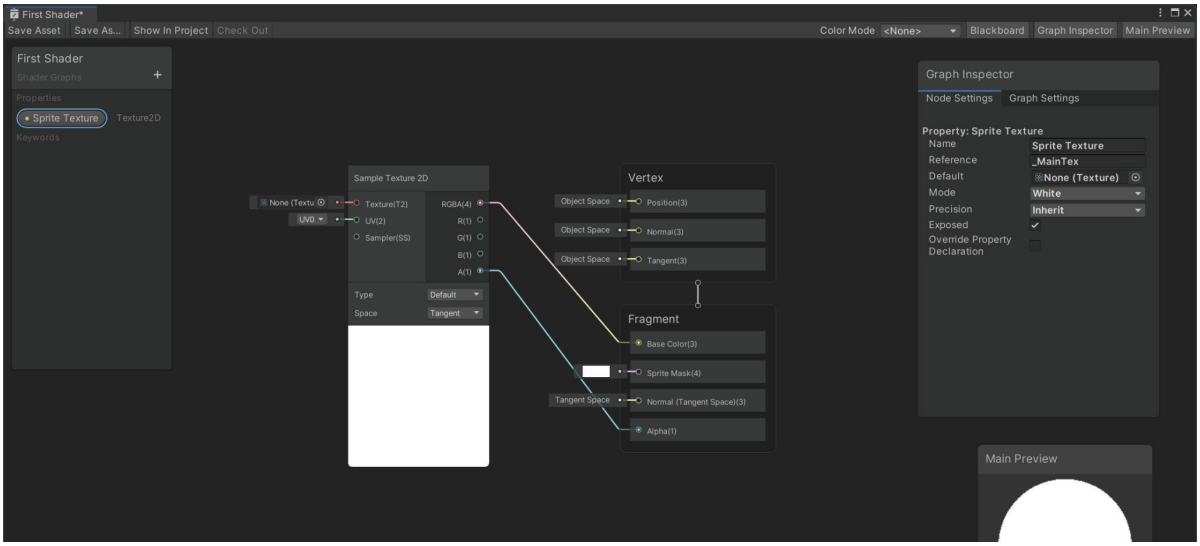
新しいノードを追加するには、ワークスペース上の任意の場所で右クリックし、コンテキストメニューから**「Create Node」**を選択します。タイプ別にグループ化されたすべてのノードのリストが表示されます。**「Input」>「Texture」>「Sample Texture 2D」**の順に選択します。




Texture 2D ノードの例

Sample Texture 2D ノードの RGBA(4) スロットを、Master Stack の入力スロットの Base Color(3) に接続します。A(1) スロットを Alpha(1) に接続します。これで、Master Stack の Fragment コンテキストでは、Sample Texture 2D ノードに指定したテクスチャの色とアルファが使用されるようになります。

シェーダーの外部からこのテクスチャを参照するには、プロパティが必要です。



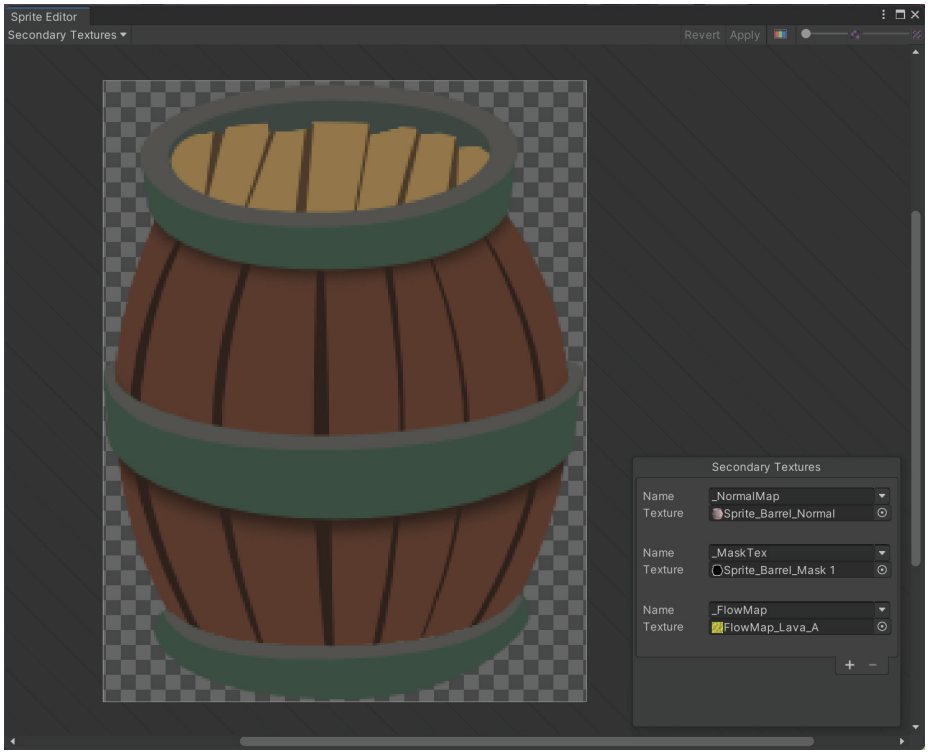

公開済みのプロパティは Inspector ビューに表示されます。ここで、グラフに使用されるパラメーターを変更できます。

ブラックボードにあるプラス（+）ボタンをクリックし、「Texture 2D」を選択します。新しく作成したプロパティを選択すると、Graph Inspector にオプションが表示されます。プロパティの名前を変更することもできますが、より重要なのは「Reference」オプションです。これを「_MainTex」に変更します。

このテクスチャでは、すべての Sprite Renderer に設定されているスプライトを使用します。Sprite Renderer は、現在のマテリアルのシェーダーの _MainTex プロパティをチェックし、そのスプライトテクスチャをシェーダーのテクスチャとして設定します。最後に、このプロパティをブラックボードからワークスペースにドラッグし、Sample Texture 2D ノードの Texture(T2) 入力スロットに接続します。ツールバーの「Save Asset」ボタンを押すと、シェーダーがコンパイルされます。

シェーダーグラフは、Unity の他のシェーダーと同様に使用できます。「Project」ウィンドウ内で右クリックして、**「Create」>「Material」**の順に選択し、作成したシェーダーをリストから選択します。シェーダーのすべてのプロパティをカスタマイズできます。

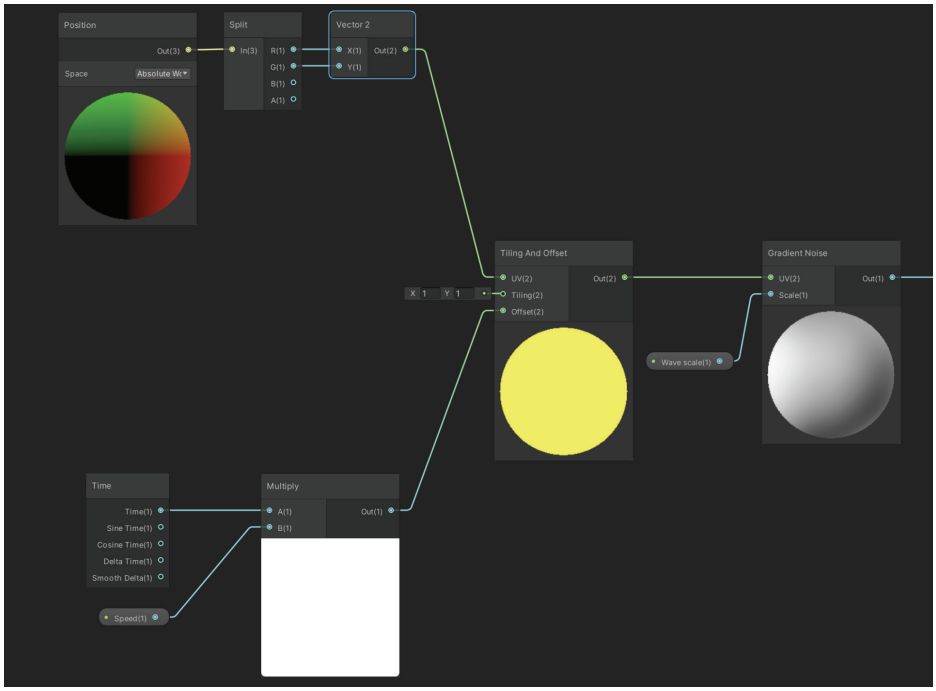
作成したマテリアルをスプライトに適用して、シェーダーをテストできます。もちろん、これはテクスチャだけを使用した最も単純なシェーダーです。自由に実験して、残りのノードを試してみてください。あるいは、このシェーダーに法線マップとマスクマップ（スプライトマスク）を追加してみても構いません。ヒントとして、法線マップを正しく表示するには、Sample Texture 2D の「Type」を「Normal」に設定する必要があります。また、「Sprite Editor」の「Secondary Textures」ウィンドウで、法線マップテクスチャとマスクマップテクスチャの参照名を確認できます。それでは頑張ってください！




樽のスプライトの「Secondary Textures」。テクスチャの名前を確認し、シェーダーグラフで名前で参照できる独自のテクスチャを追加できます。

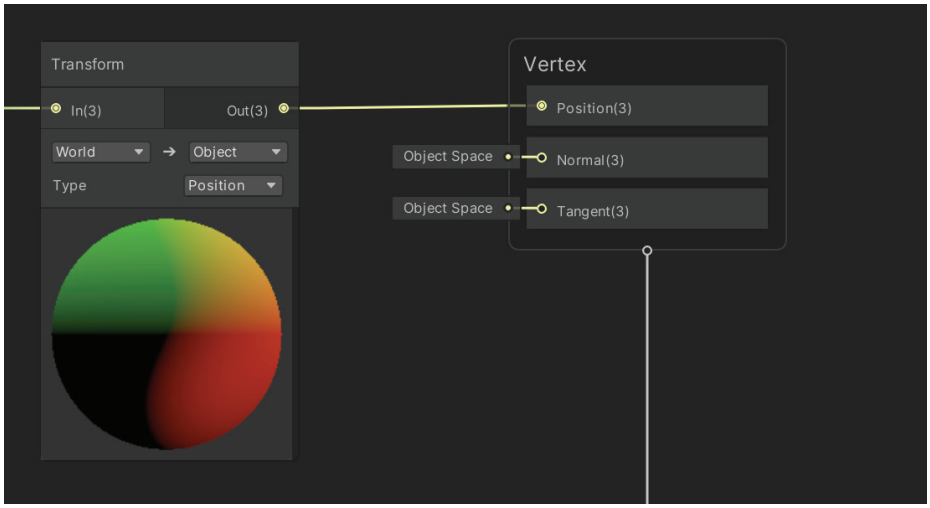
## 頂点ディスプレイスメント

頂点ディスプレイスメントシェーダーは、メッシュのジオメトリの頂点の位置に影響を与えます。さまざまな方法で頂点を変位させることができますが、最も簡単な方法は動くノイズを適用することです。このシェーダーは、植生、ぶら下がったつるやロープ、旗など、揺れるオブジェクトに適しています。このシェーダーグラフファイルの例は、*Dragon Crashers* プロジェクトにあります。ファイル名は、*ShaderGraph_Sprite_Lit_Waving* です。



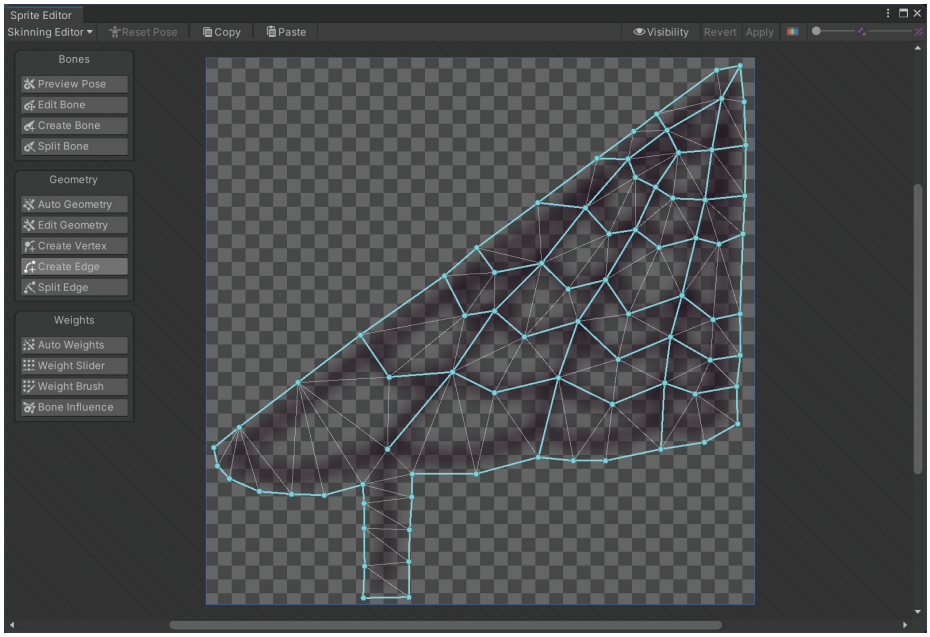

Waving シェーダーの主要部分

波形アニメーションを実現するために、Gradient Noise が波形マスクとして使用されています。その UV オフセットは、Speed という名前の Float を乗算した Time ノードによってアニメーション化されます。これらはすべて、Vertex コンテキストの頂点の位置に影響します。

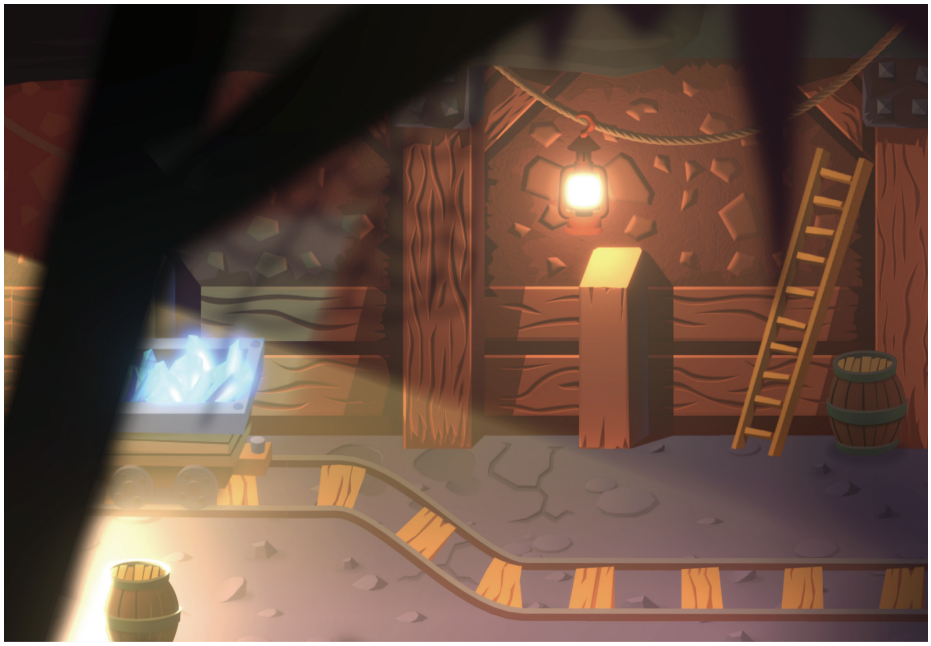



シェーダーグラフの Vertex コンテキストで頂点位置を変更し、Fragment という Master Stack でピクセル情報を変更します。

Waving シェーダーの影響を受けるスプライトには、十分な頂点数が必要です。頂点数が不十分な場合、アニメーションが粗く見えます。頂点を編集するには、スキニングエディターの Geometry ツールを使用します。




頂点を編集する

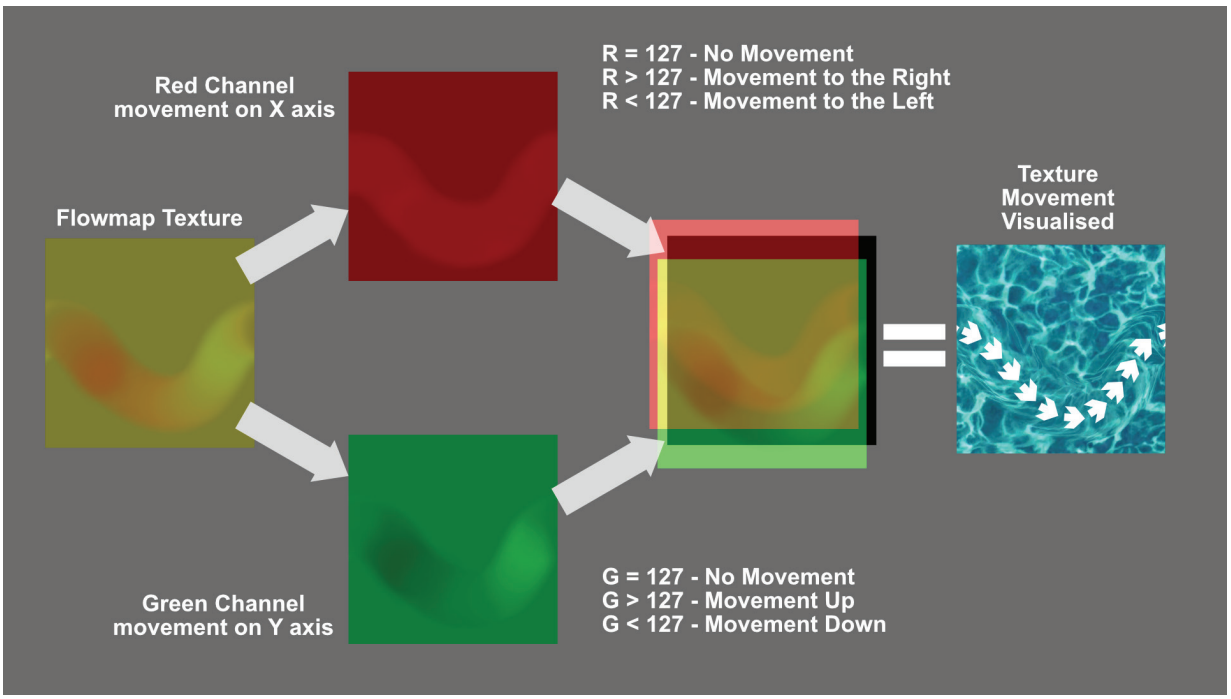


Cobweb シェーダーの動作（クリックするとアニメーションが表示されます）。

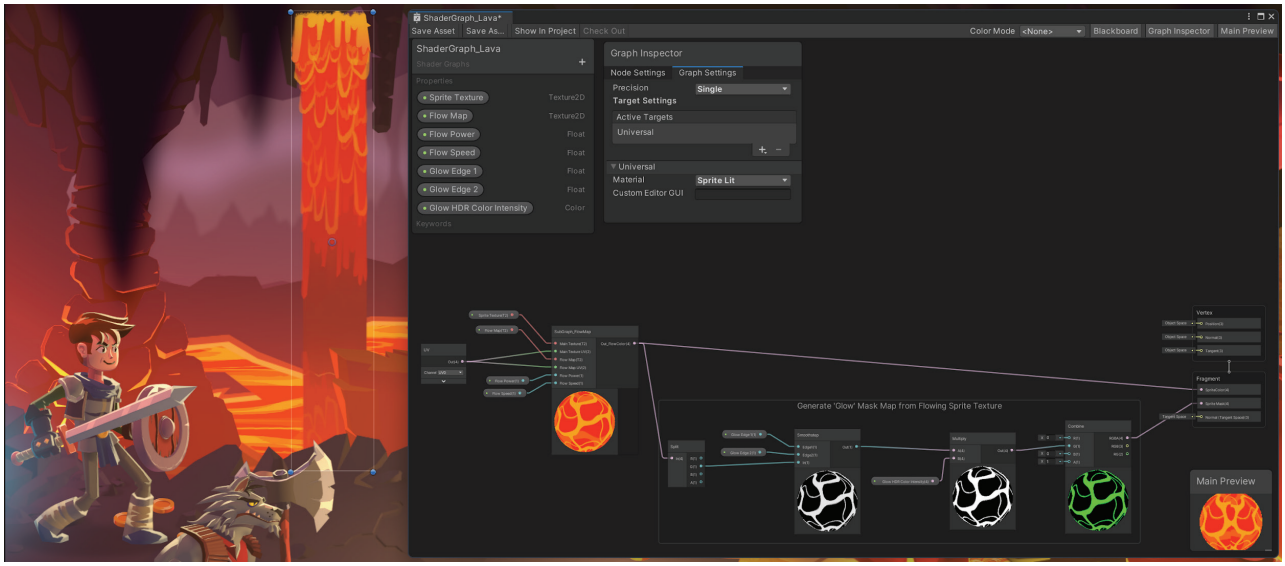
## フローマップ

フローマップは方向情報を保存するテクスチャです。『Dragon Crashers』デモに、ShaderGraph_Lava という名前のフローマップシェーダーがあります。このシェーダーでは、フローマップテクスチャを使用して、メインテクスチャの UV 座標の方向を制御します。赤色と緑色を使って XY 方向を指定することで、ピクセルがフレームごとに移動し、メインテクスチャのピクセルが「流れる」ように見えます。

SubGraph FlowMap を開いて、このエフェクトを実現する方法を学習してください。



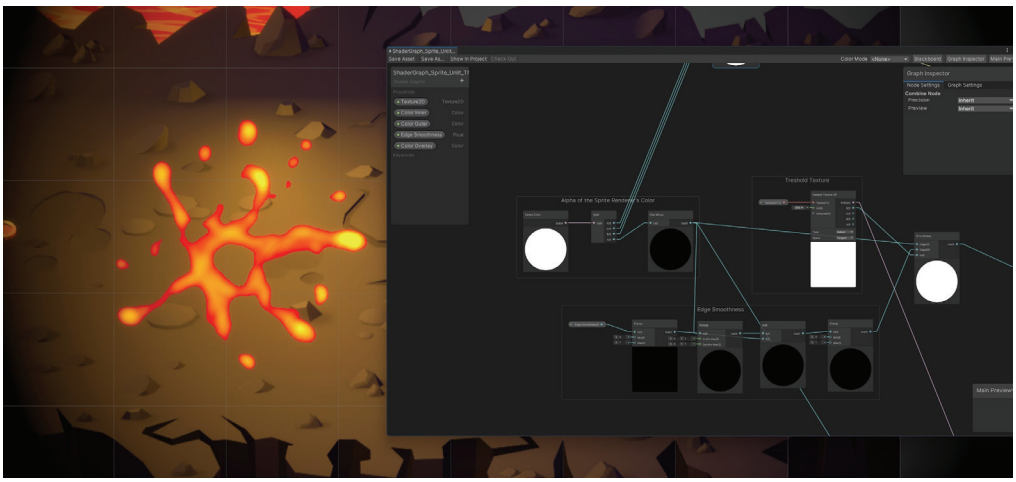

フローマップテクスチャの色の説明：赤色は X 軸のピクセルの動きを制御し、緑色は Y 軸のピクセルの動きを決定します。
このシェーダーは、矢印で表示された方向にメインテクスチャのピクセルを移動させます。




手描きのフローマップが、プロジェクトのアートディレクションに合わせて溶岩の流れの粘性をマンガのように表現する効果を与えています。フローマップの作成に使用できるツールとして、Flow Map Generator / Visualizer と Flow Map Painter があります。

## 流体プロップアニメーションのしきい値アニメーション

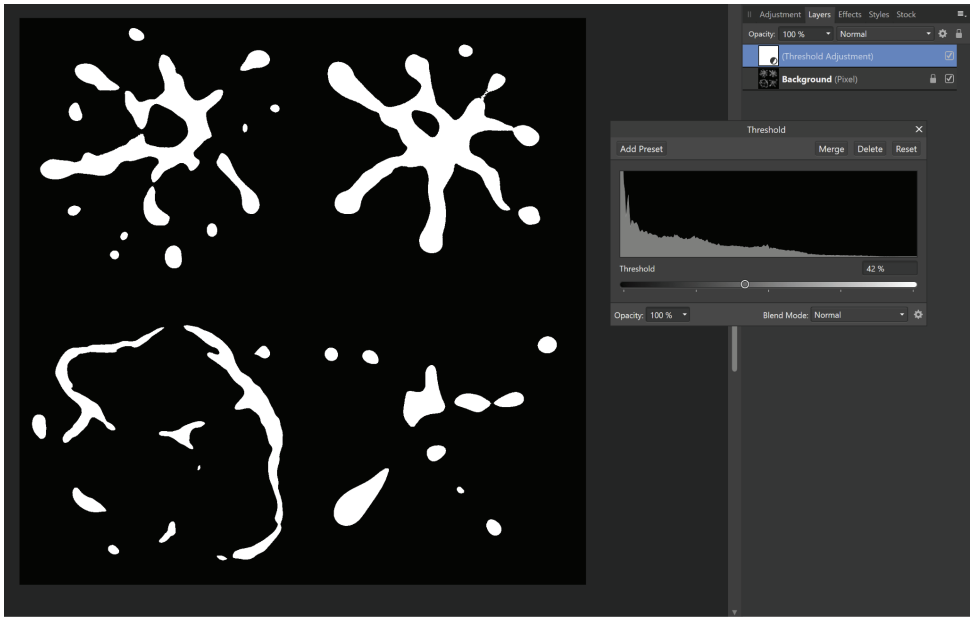
1 つのテクスチャから滑らかなアニメーションを作成する、「アニメーションアルファクリッピング」と呼ばれるシェーダーアニメーション手法もあります。この手法では、各フレーム内で特定の範囲のピクセルを、ピクセルのアルファ値に基づいて表示します。テクスチャはシングルチャンネルであるため、このエフェクトは容易に実現できます。『*Dragon Crashers*』に、*ShaderGraph_Sprite_Unlit_ThresholdAnim* という名前のサンプルがあります。



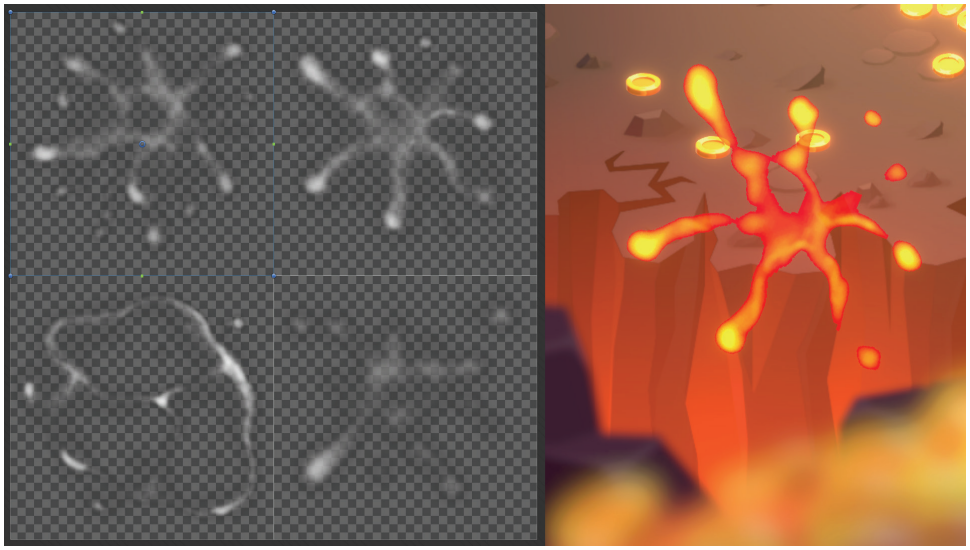
アニメーション化されたアルファクリッピング

このエフェクトを実現するには、Smoothstep ノードを使用します。テクスチャの赤チャンネルが In(1) スロットに接続され、頂点色のアルファによって Smoothstep エフェクトのエッジと滑らかさの評価が制御されます。頂点色は Sprite Renderer の Color プロパティで制御されるので、スプライトの不透明度を変更したときにテクスチャがアニメーションします。そのため、このシェーダーはパーティクルに対しても使用できます。

Affinity Photo を使用する場合は、このシェーダーのテクスチャを白黒でペイントし、Threshold Adjustment を使用して、値スライダーを動かすとアニメーションをプレビューできます。




Affinity Photo でしきい値アニメーションをプレビューする

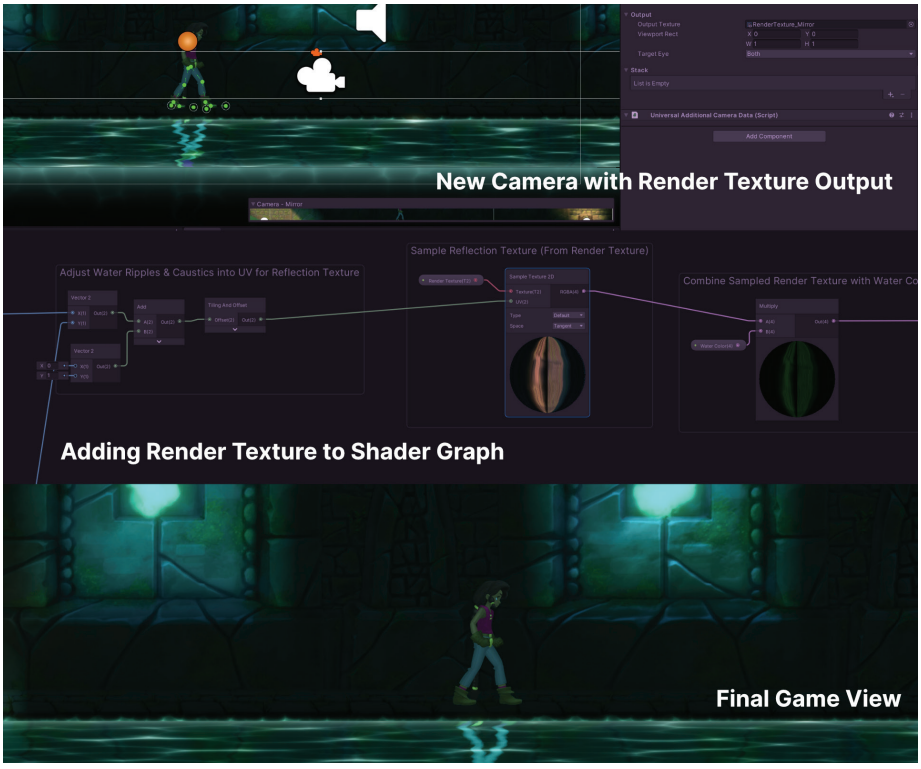


溶岩のエフェクトに使用されているテクスチャとゲーム内での見た目（こちらでアニメーションを確認できます）

このエフェクトによって、フリップブックアニメーションを使用せずに流体や飛び散りを表現できます。このアニメーションの詳細については、Surface Tension コミュニティの投稿をご覧ください。

### 反射と屈折

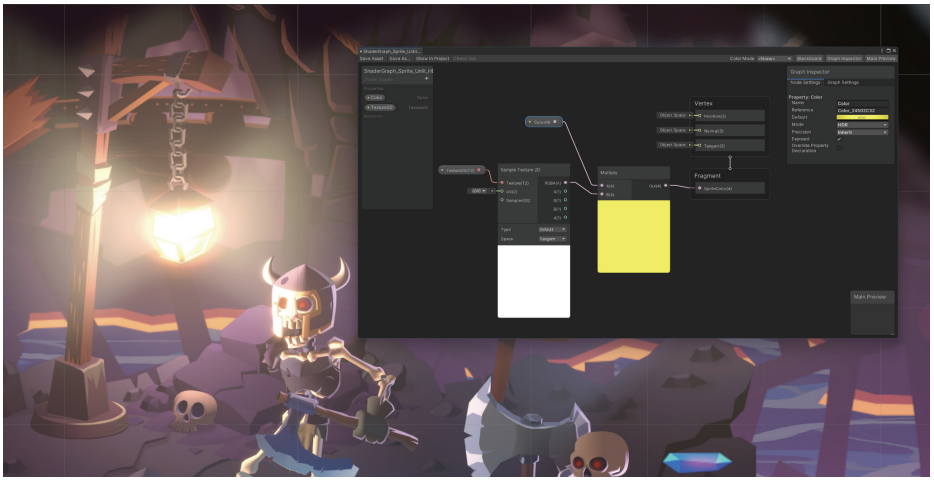
水、氷、ガラス、熱気による光の反射と屈折の表現は、よく使用される 2D 手法です。このエフェクトを実現するには、反射または屈折する部分を別のカメラでレンダーテクスチャに出力します。このテクスチャをシェーダーで使用し、必要に応じて歪ませ、調整して、最終的に画面に出力します。




『*Lost Crypt*』の「ShaderGraph Water Unlit」というデモシェーダーファイルを使用して作成された、水の反射の設定プロセス

## ライトの周囲への光の追加

まばゆい光でスプライトを光らせるには、HDR 拡張スプライトシェーダーとブルームポストプロセスエフェクトを組み合わせます。



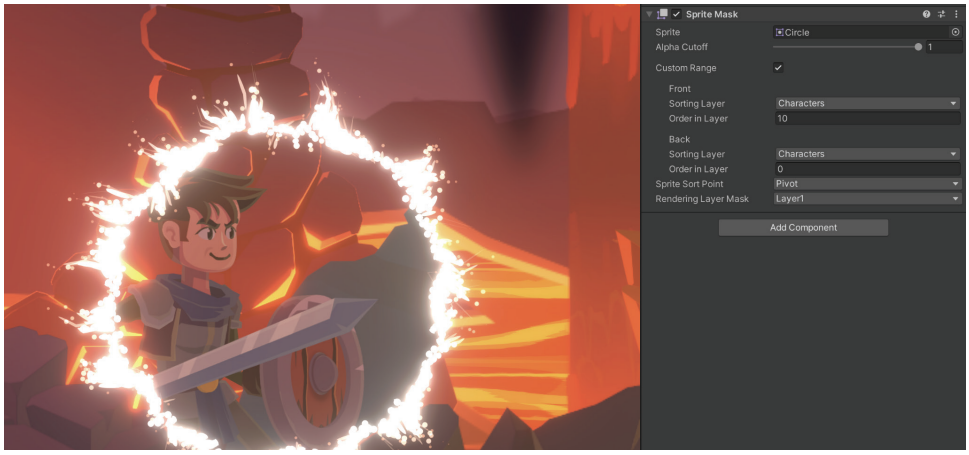
『*Dragon Crashers*』の HDR シェーダー（ShaderGraph_Sprite_Unlit_HDRTint）

このまばゆい光のエフェクトを作成するには、HDR カラーモードを使用する Color ノードによってテクスチャを乗算します。「Material」で、HDR 値を 1 よりも高い値に設定し、メインカメラのブルームポストプロセスエフェクトを有効にします。

## スプライトマスキング

スプライトマスクは、スプライトまたはスプライトのグループの一部を非表示にしたり、表示したりするために使用されます。例えば、画像の一部を非表示にしてポータルのようなエフェクトを作成したり、3D エフェクトを使ったトレーディングカードを作成したりできます。

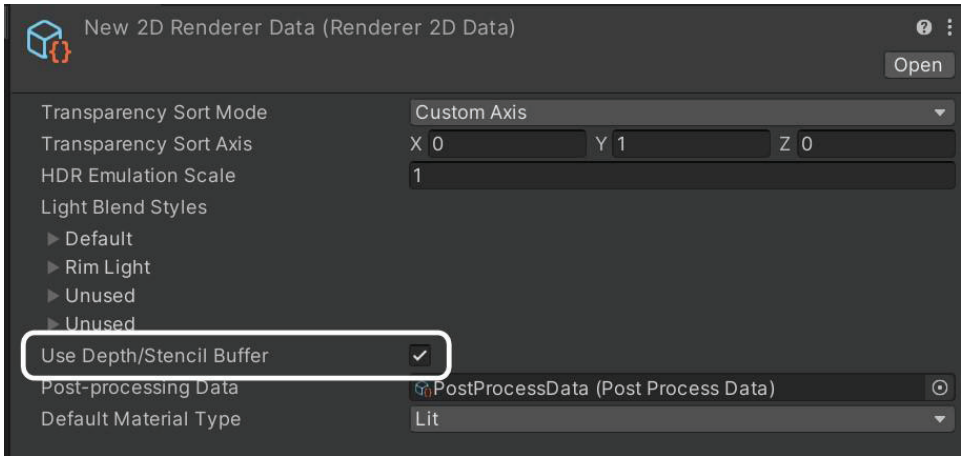
スプライトマスクを作成するには、メニューに移動し、**「GameObject」>「2D Object」>「Sprite Mask」**の順に選択します。次に、マスクの形状として使用するスプライトを選択します。マスクのスプライト自体は表示されないことに注意してください。マスクが機能するソートレイヤーを設定します。




スプライトマスクとパーティクルシステムを使って作成されたポータルエフェクト

マスク画像として使用されるスプライト自体を、アニメーションツールによってアニメーション化できます。これにより、面白い効果を生み出すことができます。例えば、ポータルを出現させ、それを大きくしたり、形状を変えたりすることができます。

マスクのスプライトがスプライトマスクの影響を受けるようにするには、Sprite Renderer で「Mask Interaction」が設定されている必要があります。URP の 2D Renderer で、「Use Depth/Stencil Buffer」が有効になっている必要があります。




マスキングに使用される「Use Depth/Stencil Buffer」オプション

**2D スケルタルアニメーションとフリップブックアニメーションの違い**

<table>
<tr><th colspan="2">フリップブック</th><th colspan="2">2D スケルタル</th></tr>
<tr><th>長所</th><th>短所</th><th>長所</th><th>短所</th></tr>
<tr>
<td>• アーティスティックな外観<br>• あらゆるものをアニメーション化できる（キャラクターの循環が簡単）<br>• ランタイムコストがほとんどかからない</td>
<td>• 作成にコストがかかる（時間、スキル）<br>• 後で編集するのが難しい<br>• 大量のメモリを消費する<br>• このアニメーションでは常に 1 秒あたりのフレーム数が作成時の設定に制限されるため、アニメーションのスピードやフレームレートの変更が必要な場合、手動で編集する必要がある</td>
<td>• 簡単に設定してアニメーション化できる<br>• 後で簡単に編集できる<br>• テクスチャサイズが小さい<br>• スケルトンごとに複数のキャラクターやスキンをサポートできる<br>• 服や武器の追加、IK のアニメーション化など、ランタイムに編集できる<br>• アニメーションが常に滑らかになる - 補間<br>• アニメーションをブレンドして、スムーズな遷移を作成できる</td>
<td>• ある程度のランタイムコストがかかる<br>• Y 軸を中心とした回転などのアニメーションが難しくなる<br>• ピクセルアートではうまく機能しない</td>
</tr>
<tr>
<td colspan="2">以下の場合に使用：<br>• 美しい見た目のスタイルや特定のスタイルを必要としている<br>• ピクセルアートアニメーションに重点を置いている<br>• 数フレームのシンプルなアニメーションにしたい</td>
<td colspan="2">以下の場合に使用：<br>• 高いアニメーションスキルがない<br>• アニメーションを迅速に作成したい<br>• 複数のスキンやキャラクターを必要としている<br>• ランタイムの制御を向上させたい<br>• テクスチャ空間を節約したい</td>
</tr>
<tr><td colspan="4">両方の手法を組み合わせることができます。手足の動きはボーンで制御する一方で、顔の表情はスプライトスワップで制御できます。</td></tr>
</table>

**シェーダーアニメーション、VFX 用アニメーションクリップ、パーティクルシステムの違い**

| シェーダー | | アニメーションクリップ | | パーティクルシステム | |
|---|---|---|---|---|---|
| **長所** | **短所** | **長所** | **短所** | **長所** | **短所** |
| • 他の手法では不可能な効果を実現<br>• 手続き型の効果<br>• 制作時間の短縮<br>• テクスチャサイズが小さい | • ランタイムの計算が難しい | • 視覚的に高品質<br>• アニメーションとそのフレームをアーティスティックに制御 | • テクスチャサイズ<br>• 個々のフレームの作成に時間がかかる<br>• 後でカスタマイズするのが難しい | • 作りやすく、後で簡単に編集できる<br>• ランダム化して独自の見た目を定義できる | • 複雑な効果<br>• 大量のパーティクルが必要なため、CPU に負荷がかかる可能性がある |
| **以下の場合に使用：**<br>• 数フレームのシンプルなアニメーションにしたい。フレームごとに実行したときに、他の手法では多くの容量やメモリを必要とする効果に適している<br>• カスタマイズが必要な場合に最適<br>• 効果をマテリアルに適用する場合に最適 | | **以下の場合に使用：**<br>• 自然なアニメーションやマンガのようなアニメーションに適している（昆虫、鳥、煙の効果など）<br>• ランタイムコストが非常に低い | | **以下の場合に使用：**<br>• 炎、流体、魔法など、少しランダム化された効果に適している | |

**タイルマップ、スプライト、スプライトシェイプの違い**

| タイルマップ | | スプライト | | スプライトシェイプ | |
|---|---|---|---|---|---|
| **長所** | **短所** | **長所** | **短所** | **長所** | **短所** |
| • ペイント、タイル、ルールタイルの高速作成プロセス<br>• 正確<br>• ゲームの最適化に適している | • グリッドにコンストレインされている場合に、有機的な形状を作成するのが難しい | • オブジェクトを手動で配置する場合に最適 | • 複数のスプライトをペイントするのが難しい<br>• 大量に使用した場合に最適化されない | • 多角形も有機的な形状も簡単に編集できる | • ピクセルパーフェクトを実現できない<br>• 配置が無造作<br>• テクスチャが繰り返しになる可能性がある |
| **以下の場合に使用：**<br>• レベル要素の正確な配置に適している<br>• グリッドは経路検索に適している<br>• 汎用性が高い | | **以下の場合に使用：**<br>• タイルマップやスプライトシェイプを補完するものとして、追加要素に適している | | **以下の場合に使用：**<br>• 有機的な形状に最適<br>• レベルの高速編集 | |

### 最適化のヒント

— フリップブックアニメーションを使用する場合は、フレーム数 / テクスチャサイズを抑えるようにします。

— オブジェクトプールを使用すると、VFX アセットのインスタンス化と破壊を回避できます。ゲームオブジェクトをアクティベート / 非アクティベートするだけです。

— シェーダーグラフでは、可能な限り半精度を使用します。

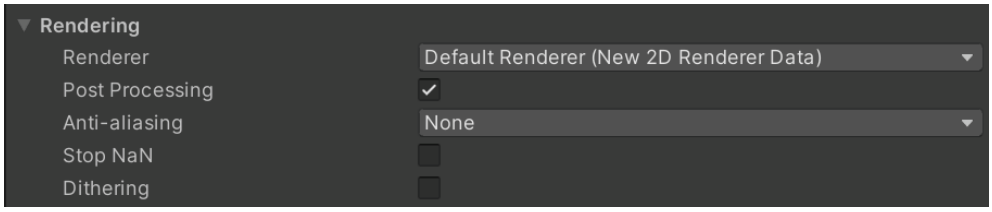
# ポストプロセス

## 概要

ポストプロセスとは、最終的な画像フレームにエフェクトを追加することで、ゲーム画面のクォリティを高めることです。そのようなエフェクトによって物理的なビデオカメラの映像のシミュレーションを行ったり、そのようなエフェクトを完全にスタイル化したりできます。URP にはポストプロセス機能が含まれているので、別のパッケージをインストールする必要はなく、エフェクトの設定は簡単です。

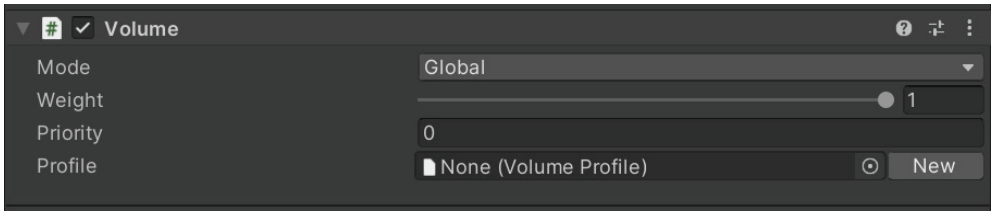

左側はポストプロセスを使用していないカメラ出力の画像。右側はポストプロセスを追加した同じ画像。違いを示すために効果を誇張しています。特にモバイルゲームでは、ポストプロセスエフェクトを慎重に適用します。

まず、メインカメラを選択し、「Post Processing」オプションをオンにします。




メインカメラのポストプロセスを有効にする

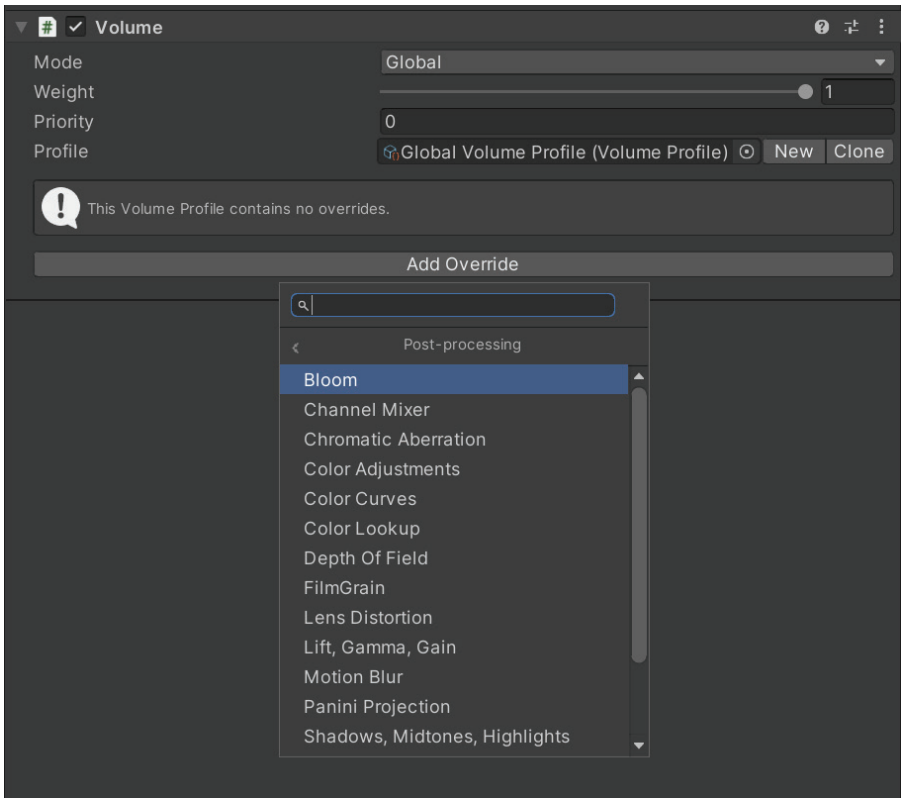
ポストプロセスでは、ボリュームシステムを使用します。新しいポストプロセスボリュームを追加するには、**「GameObject」>「Volume」>「Global Volume」**の順に選択します。




ボリュームプロファイルを定義する

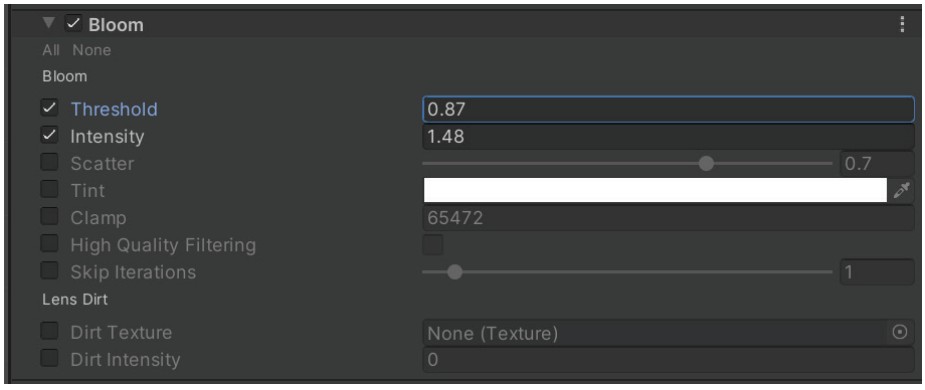
次に、「New」ボタンをクリックして、ボリュームプロファイルを作成します。ボリュームプロファイルにポストプロセスエフェクトが保存されます。ボリュームプロファイルは、ボリューム間やシーン間で簡単に共有できます。

ポストプロセスエフェクトを追加するには、「Add Override」ボタンをクリックし、リストから目的のエフェクトを選択します。




「Bloom」ポストプロセスエフェクトを選択する

まず、「Bloom」を選んでみましょう。

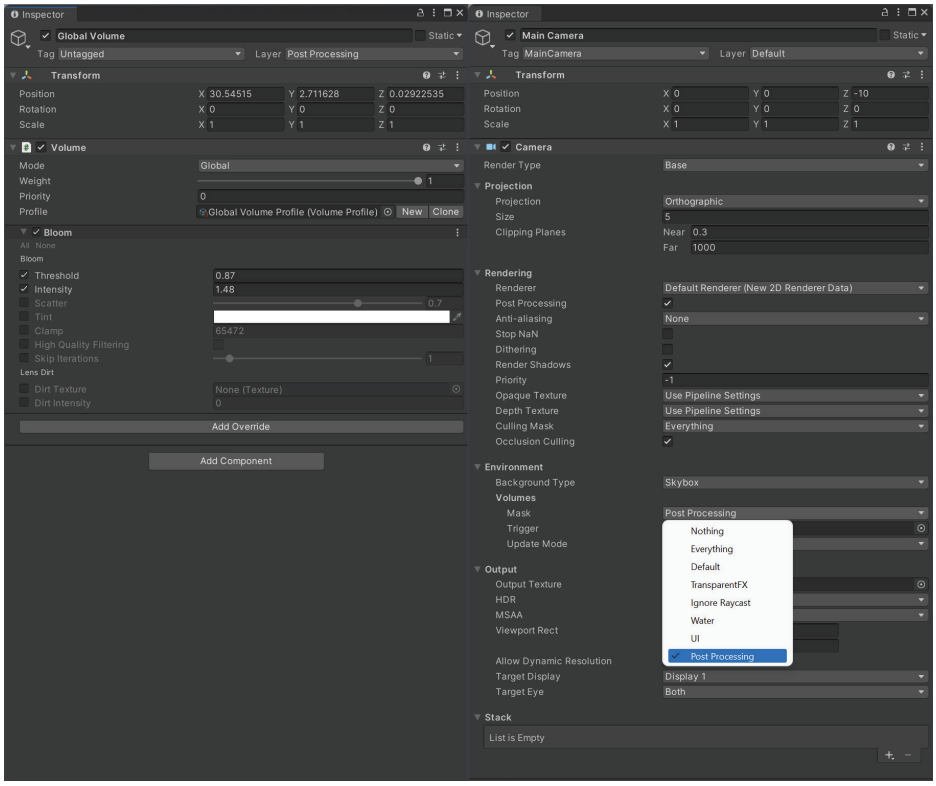



ブルームエフェクトのプロパティを定義する

エフェクトのプロパティを調整するには、隣接するチェックボックスをオンにします。

グローバルボリュームのゲームオブジェクトレイヤーが、メインカメラの「Mask」オプションで選択されているレイヤーと一致していることを確認してください。

すべてが正しく機能していれば、ゲームビューにブルームエフェクトが表示されます。



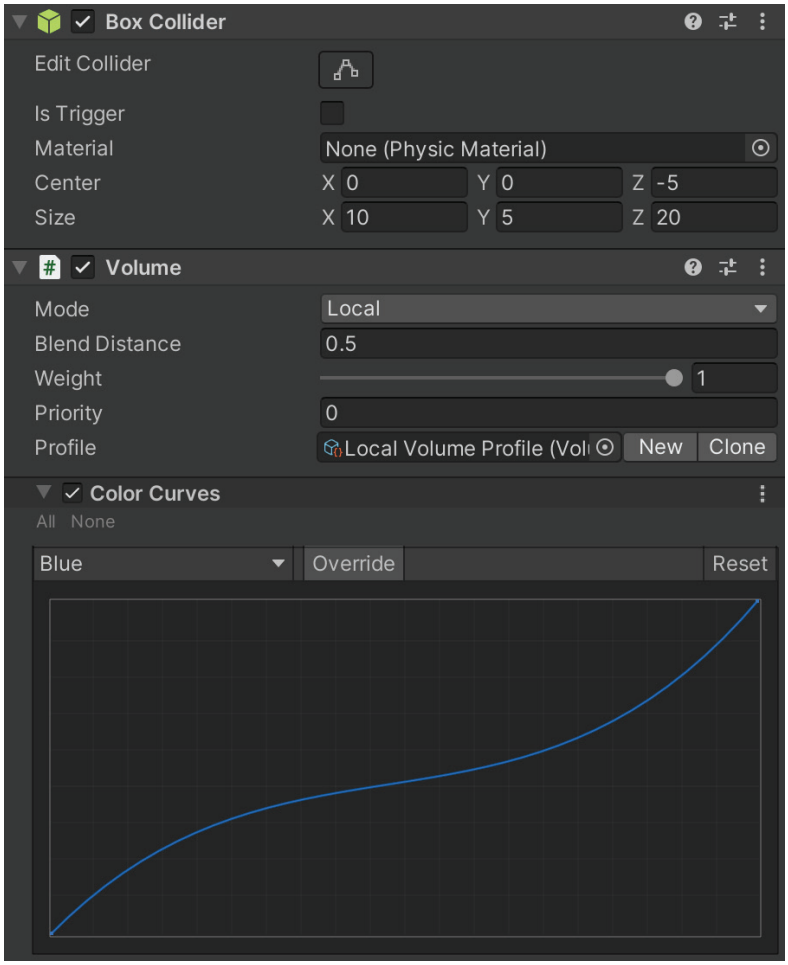

カメラの「Post Processing」チェックボックスが有効になっており、カメラの「Volumes」の「Mask」ドロップリストでボリュームのゲームオブジェクトのレイヤーが選択されていることを確認する

グローバルボリュームを使用しているため、シーンでボリュームオブジェクトがアクティブになっているときは、このエフェクトが常に表示されます。

### ローカルボリューム

ポストプロセスエフェクトはレベル全体に追加できますが、プレイヤーが建物に入ったときなどに、レベルの一部にエフェクトを設定することも可能です。

そのためには、異なるポストプロセスプロファイルをアタッチした別のボリュームオブジェクトを作成する必要があります。




ボリュームオブジェクトに 3D Box Collider を追加する

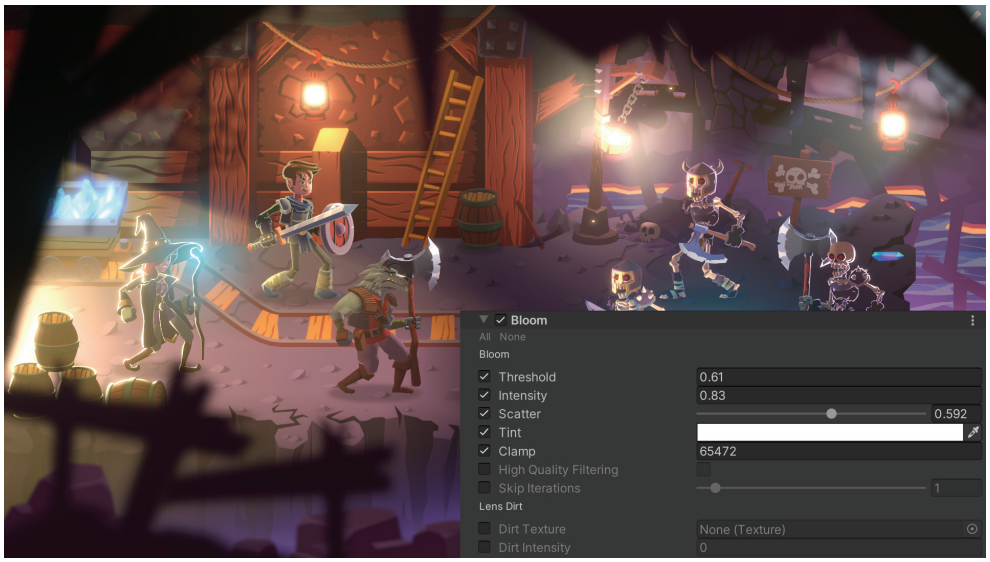
「Volume」の「Mode」を「Local」に設定し、3D Box Collider をボリュームオブジェクトに追加します。カメラが Box Collider の境界内に入ったときにポストプロセスが適用されるため、Z 軸方向のサイズが、カメラオブジェクトが十分に収まる大きさであることを確認します。デフォルトでは、コライダーが小さすぎるので、シーンビューでサイズを再確認してください。

「Blend Distance」の値を増やすと、カメラがボリュームゾーンに入ったときにエフェクトが突然有効になるのを防ぐことができます。

## 2D グラフィックス用のエフェクトの概要

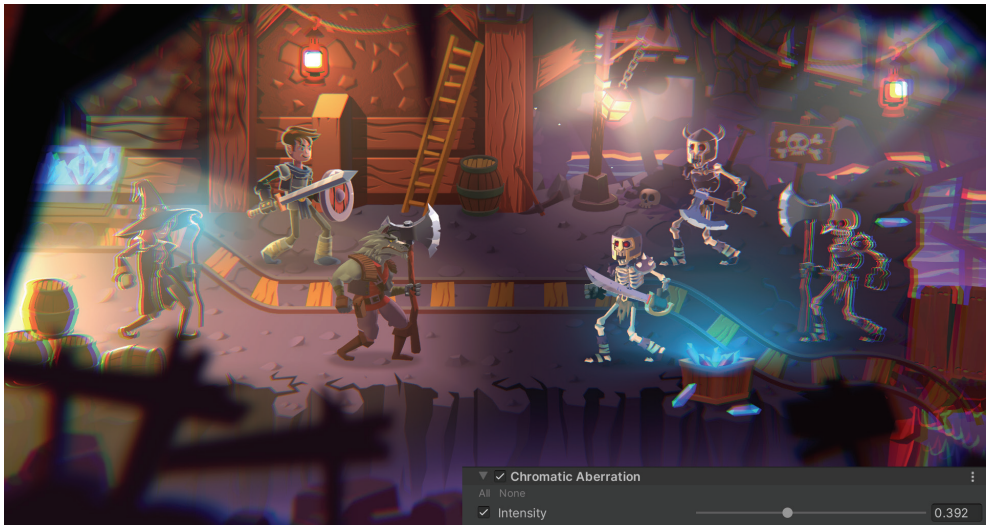
URP のポストプロセスには多くのエフェクトが組み込まれており、ほぼすべてのエフェクトを 2D ゲームで使用できます。以下に各エフェクトの概要を簡単に示します。

### ブルーム




ブルームエフェクトは、画像の最も明るい領域の周囲に光を作り出し、部分的に明るくなったような印象を与えて、レンズによる光の取り込みのシミュレーションを行います。見栄えをさらに良くするには、カメラレンズの欠点のシミュレーションを行う Lens Dirt テクスチャを追加してみます。最高のパフォーマンスを得るために、特にモバイルでは、「High Quality Filtering」を無効にします。

### 色収差

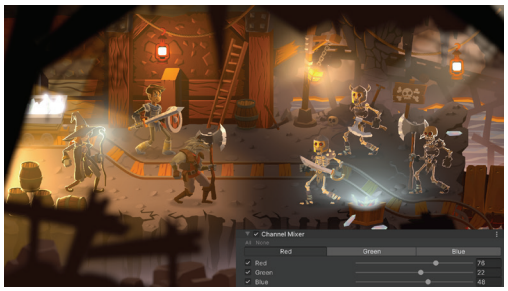



このエフェクトは、画像のエッジに色の歪みを作り出します。これは、中心から離れた部分で色が分かれるレンズのシミュレーションを行います。

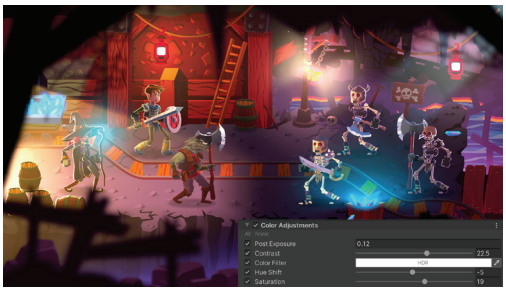
### カラーグレーディング

これは 1 つのエフェクトではなく、エフェクトのグループです。カラーグレーディングの目的は、画像の色を変更してシーンの雰囲気をスタイライズしたり、変えたりすることです。カラーグレーディングでは、色温度を調整して、環境の時間帯や温度のシミュレーションを行うことができます。このエフェクトを利用して、ゲームに独特の雰囲気を付け加えることができます。

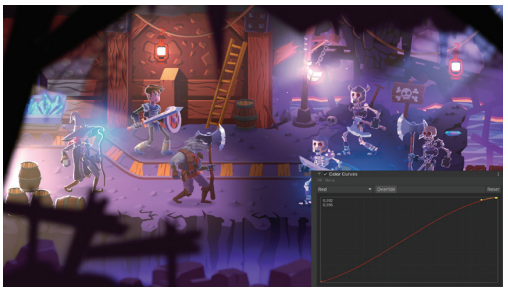
URP では、以下のエフェクトを使用してカラーグレーディングを制御しています。



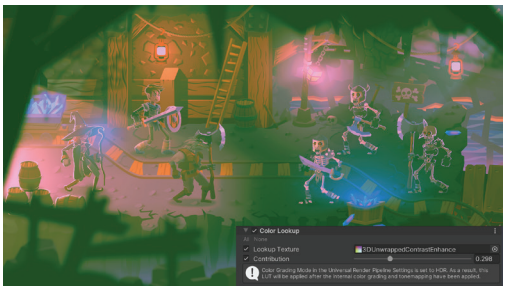

チャンネルミキサー：最終的な画像に対する各入力カラーチャンネルの影響度を変更する



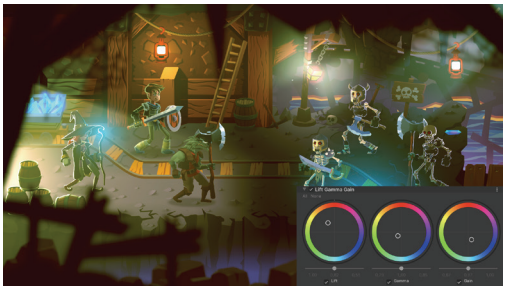

色調整：最終的にレンダリングされる画像の全体的なトーン、明度、コントラストを調整する



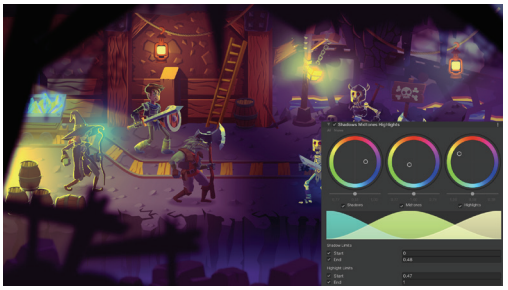

カラーカーブ：画像の色と輝度の正確な調整を可能にする




カラールックアップ：1 つのテクスチャを使用したカラーグレーディングの効率的な方法

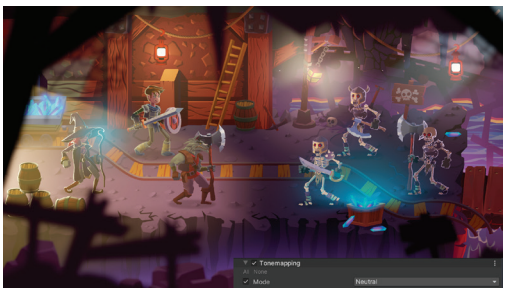



リフト、ガンマ、ゲイン：3 方向のカラーグレーディングを実行する

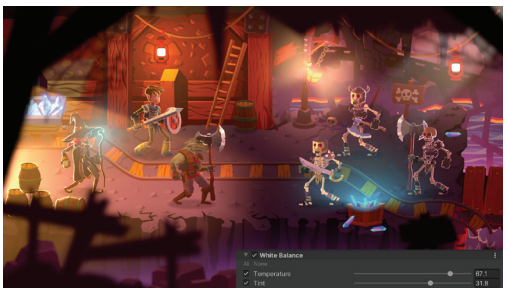



シャドウ、中間調、ハイライト：画像の影、中間調、ハイライトを個別に制御する


スプリットトーン処理：輝度値に基づいて画像のさまざまな領域を色付けし、影、中間調、ハイライトに異なる色をわずかに加える

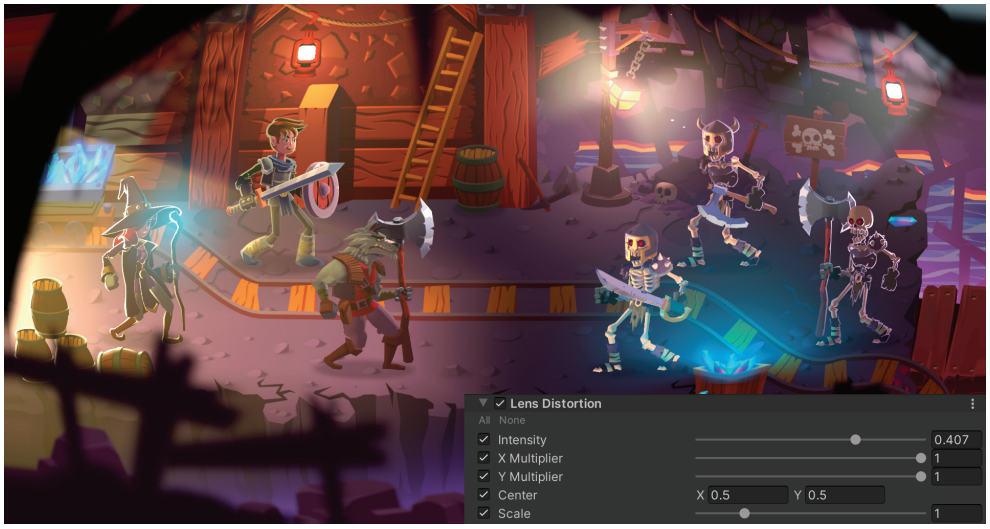



トーンマッピング：画像の HDR 値を新しい範囲の値に再マッピングできる




ホワイトバランス：色温度のバランスを取るだけでなく、画像を冷たい色調または暖かい色調に変えることもできる

## Lens Distortion（レンズディストーション）




これは、現実世界の広角レンズでよく見られる樽型歪曲収差の効果を生み出します。画像上の直線を内側に湾曲させ、エッジに近いほどこれが顕著になります。このエフェクトを使用して、古いブラウン管テレビの湾曲を再現したり、魚眼レンズ効果を追加したりできます。

## Vignette（ビネット）


ビネットはエッジに向かって画像を暗くします。これは、画像の中央に注目させる、現実世界のレンズの別の特性です。

### Film Grain（フィルムグレイン）

フィルムグレインは画像に微妙なノイズを加えて、ネガフィルムの乳剤構造と同様の効果を生み出し、アナログ映画で見られる粒子を表現します。

### Panini Projection（パニーニ投影）

パニーニ投影は、有効視野（FOV）が非常に広い透視図のレンダリングに役立つように設計されています。2D で使用すると、円筒形の歪みがある画像をスタイライズすることもできます。

## 最適化のヒント

— 使用するポストプロセスエフェクトをできるだけ少なくします。

— できる限り、負荷の少ないエフェクト（ブルーム、色収差、カラーグレーディング、レンズディストーション、ビネットなど）を使用します。

— フィルムグレインとパニーニ投影は負荷が高いエフェクトです。

— ブルームの場合は「High Quality Filtering」を無効にします。

# まとめ

この長いガイドも、これで終わりとなります。Unity で美しい 2D ゲームを制作する取り組みを開始する際に、本書でご紹介した手順や知識が一助となれば幸いです。

長年にわたって、ゲーム開発で苦悩するたくさんの人たちを見てきました。そういった人たちは、「もう少し勉強してから」、あるいは「チュートリアルで学んでから」でないと、実際の開発には手を付けられないと思っているようです。しかし、最良の学びは実践から得られます。ですから、今、始めましょう。

もっとも、最初の一歩を踏み出した後で何かの壁に直面する可能性も十分にあります。チュートリアルを試したり、技術記事で詳細を調べたり、開発者仲間からアドバイスをもらったりする必要があるかもしれませんが、また元の道に戻ることができるでしょう。

ゲーム開発には、実践と学習の両方が必要です。常にこの 2 つのバランスを取ってください。

この 1 冊きりの e ブックで 2D ゲーム開発の全領域を網羅して説明することは不可能です。しかし、プロジェクトを開始し、これらのツールの知識を深める際に、こちらの e ブックを手元に置いておくとよい参考になります。他にも、Unity のドキュメント、Unity Learn、Unity ブログなど、開発者やアーティスト向けの追加リソースが多数用意されています。

グッドラック！

– Jarek Majewski、Unity 2D チーム


Unity®