

→ EBOOK



Unity で高度なビジュアル エフェクトを作成するため の決定版ガイド



Sakura Rabbit 氏により Unity で制作されたシーン

Contents

はじめに.....	7
寄稿者	8
主著者	8
Unity の貢献者	8
外部の貢献者	8
リアルタイム VFX を始める.....	9
ビジュアルワークフロー.....	11
コアグラフィックスパッケージ.....	12
その他のサンプルコンテンツ.....	12
ビルトインパーティクルシステム	13
VFX Graph 入門	16
VFX Graph アセットとコンポーネント	17
VFX Graph ウィンドウ	19
グラフロジック	21
システム、コンテキスト、ブロック	22
プロパティとオペレーター	27
Blackboard	30
グループノードと Sticky Note	32
サブグラフ	33
属性	35
Blackboard の属性.....	39
イベント	42
イベント属性	44
Unity 6 での UI 改善	45
ノード検索.....	45

アクティベーションポート	46
新しい VFX Toolbar	47
キーボードショートカット	47
VFX サンプルコンテンツを詳しく見る.....	48
その他のリソース	55
その他の参考資料	55
ビジュアルエフェクトの例	56
VFX Graph の Learning Templates	56
ギズモの視覚化.....	57
グラフの基礎	58
Contexts & Data Flow	58
Spawn Context.....	59
Capacity Count	59
Multiple Outputs.....	60
Bounds	61
パーティクルの方向と回転	62
テクスチャリングとフリップブック	63
パーティクルのピボット.....	65
メッシュとテクスチャのサンプリング	66
衝突とインタラクティブ性.....	69
デカルパーティクル.....	72
パーティクルストリップ	73
VFX Graph のサンプル (HDRP)	79
Smoke Portal.....	80
リアルタイムスモークのための 6 ウェイライティング	84
GooBall	87
物理演算ベースのエフェクト	92

リボンパック	95
Meteorite サンプル	98
メッシュサンプリングエフェクト	102
Skinned Mesh サンプリング	105
その他の例	107
レンダーパイプラインと VFX Graph	108
URP 用の VFX Graph	109
Lit 出力	109
2D Renderer と Shader Graph のサポート	110
デカール	111
6 ウェイスモークライティング	111
モーションベクトル	111
カメラバッファ	111
HDRP 用の VFX Graph	112
レイトレーシングのサポート	112
ボリュメトリックフォグの出力	113
Shader Graph の統合	114
ビルトイン出力	114
シェーダーエフェクト	115
Shader Graph と VFX Graph	122
Shader Graph Feature Examples.	123

インタラクティブ性	124
イベントバインダー	124
OnPlay および OnStop イベント	125
Mouse イベントバインダー	126
Rigidbody Collision イベントバインダー	127
Trigger イベントバインダー	128
Visibility イベントバインダー	129
タイムライン	130
イベント属性	132
プロパティバインダー	133
Output イベント	136
パイプラインツール	139
Point Cache	139
Point Cache ベイクツール	140
Point Cache の使用	141
符号付き距離フィールド	142
SDF の使用	142
SDF ベイクツール	143
ベクトル場	144
TFlow (Asset Store)	145
VFXToolbox	145
Image Sequencer	145
デジタルコンテンツ制作ツール	146
SideFX Houdini	146
Autodesk Maya	146
Blender	147
Adobe Photoshop	147

最適化	148
VFX Graph の Profiling パネルと Debug パネル	149
Graph Debug Information パネル	149
Particle System Info パネル	150
Context Debug パネル	150
Unity Profiler と Frame Debugger	151
VFX Control の Debug モード	153
バウンド	155
メッシュ LOD	156
Mesh Count	158
パーティクルレンダリング	159
高度な機能	160
Graphics Buffer のサポート	160
カスタム HLSL	162
追加リソース	164
動画チュートリアル	165
GitHub 上の VFX プロジェクト	166

はじめに

指先から火の玉を放ったり、ワームホールを通り抜けたり、そんなことも、ゲームのビジュアルエフェクト (VFX) なら可能です。ビジュアルエフェクトは、ゲームの雰囲気を高め、ストーリーを伝えるのに役立つだけでなく、プレイヤーの心をつかむディテールによって想像上の世界に命を吹き込みます。

Unity は、VFX Graph などのツールを通してリアルタイムグラフィックスの限界を押し広げています。このノードベースのエディターを使用すれば、テクニカルアーティストや VFX アーティストは、シンプルな一般的なパーティクルの動作から、パーティクル、ライン、リボン、トレイル、メッシュなどを含む複雑なシミュレーションまで、ダイナミックなビジュアルエフェクトをデザインできます。

本書は、VFX Graph をゲームアプリケーションに組み込むことを検討しているアーティストやクリエイター向けの包括的なガイドです。VFX Graph とその関連ツールを使用して Unity でリアルタイムのビジュアルエフェクトを構築する具体的な方法について説明します。

このガイドでは、個人開発者や 100 人規模のチームの経験を踏まえて、AAA 品質のツールをより使いやすいものにするための多くの例を紹介しています。このことにより、誰もがゲームデザインを楽しめます。

重要事項：この改定版には、Unity 6 以降で利用できる新機能とクオリティオブライフ (QOL) の改善が盛り込まれています。Unity Hub から Unity 6 をインストールし、このガイドに沿って進めてください。なお、Unity 6 以降のリリースでは、命名規則も変更されています。詳細については、[命名規則に関する記事](#)を参照してください。



VFX Graph サンプルの Smoke Portal

寄稿者

主著者

Wilmer Lin 氏は、映画や TV 業界で 15 年以上の経験がある、3D とビジュアルエフェクトのアーティストで、現在は独立系ゲーム開発者および教育者として活躍しています。Wilmer 氏の主な担当作品には、『X-Men: フューチャー & パスト』、『インクレディブル・ハルク』、『ナルニア国物語 / 第 1 章: ライオンと魔女』があります。

Unity の貢献者

Mathieu Muller (グラフィックス担当リードプロダクトマネージャー)
Vlad Neykov (ディレクター、ソフトウェアエンジニアリング、品質担当)
Orson Favre (テクニカルアーティスト、本書での多くの新しいサンプルの制作者)
Julien Fryer (グラフィックスエンジニア)
Fred Moreau (テクニカルプロダクトマネージャー)

外部の貢献者

Marie Guffroy (テクニカルアーティスト)
Thomas Iché (VFX アーティスト兼スペシャリスト)

リアルタイム VFX を始める

最近のゲーマーは、深い没入型の体験を求めています。ハードウェアの進歩により、モバイルやコンソールのプラットフォームでできることの限界が押し上げられ、かつてはハリウッドの大作映画の制作でしか利用できなかったものが、リアルタイムで実現できるようになりました。



VFX Graph の例



高度なグラフィックスへの関心と投資が増加傾向にある中、ゲームのビジュアルエフェクトは注目され続けています。何と言っても、ゲーム中のエフェクトがプレイヤーをアクションへと導くのです。

キャラクターが魔法を使わないファンタジーロールプレイングゲーム (RPG) や、輝く武器が光の尾を引かない戦闘ゲームは想像できません。レースカーがアスファルトを爆走するときには、煙が巻き上がることを期待します。

ビジュアルエフェクトがないと、風景も違って見えます。フィルムノワールの探偵物語であれば、雨と霧で街を覆うでしょう。キャラクターが荒野を冒険するなら、プレイヤーの動きに反応させて、草や葉を揺らすかもしれません。



Stunlock Studios の『[V Rising](#)』は、HD レンダーパイプラインと VFX Graph を使用して制作されたゲームである。

ビジュアルエフェクトはゲーム体験を独特なものにします。ただし、それらを制作するには、形状、色、タイミングを操る、アーティストの多分野にわたる能力が必要です。

そこで VFX Graph の登場です。この洗練されたツールは、映画の VFX を反映したワークフローとなっており、さらに 30 fps、60 fps、またはそれ以上のフレーム数で動作します。



Sakura Rabbit が Unity の VFX ツールを使用して開発中のプロジェクトの画像

ビジュアルワークフロー

ハイエンドプラットフォームで AAA レベルの複雑なビジュアルエフェクトを実現するために、ノードベースの直感的なインターフェースである VFX Graph を使って、GPU パワーによるパーティクルを生み出しましょう。

VFX Graph の活用方法は以下の通りです。

- 1 つまたは複数のパーティクルシステムを作成する
- 静的メッシュを追加し、シェーダープロパティを制御する
- C# または [Timeline](#) を介してイベントを作成し、エフェクトの一部をオンまたはオフにする
- よく使われるノードの組み合わせのサブグラフを作成することで機能ライブラリを拡張する
- 変更結果をさまざまなレートでプレビューしたり、段階的なシミュレーションを実行したりする

VFX Graph 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) で動作します。また、URP で利用できる [Lit](#) 出力と [2D レンダー](#) のサポートも追加されています。VFX Graph のすべてのレンダーパイプラインの機能の比較については[こちら](#)で確認してください。また、VFX Graph の互換性の詳細については、[こちらのドキュメント](#)を参照してください。

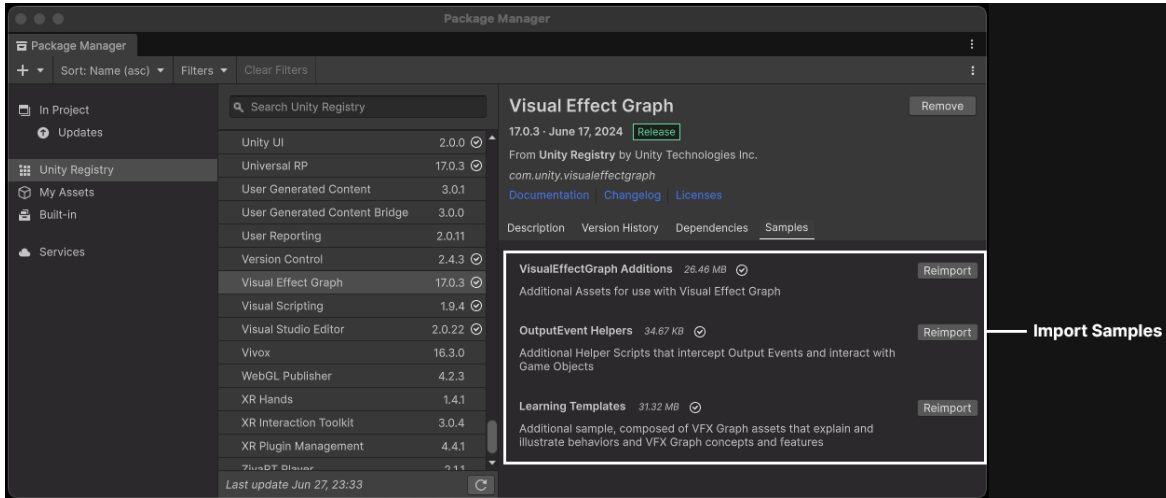
VFX Graph では、デバイスとの互換性を維持するため、コンピュートシェーダーのサポートが必要です。サポートされているデバイス：

- [Metal グラフィックス](#) API を使用した macOS および iOS プラットフォーム
- [Vulkan](#) あるいは [GLES3](#) API 対応の Linux および Windows プラットフォーム
- ハイエンドのコンピューティング機能を持つ一部の Android デバイス (URP のみ)

コアグラフィックスパッケージ

Unity の最新リリースをインストールすると、URP、HDRP、Shader Graph、VFX Graph などの最新パッケージを入手できます。コアグラフィックスパッケージが Unity のメインインストーラーに組み込まれることで、プロジェクトは常に検証済みの最新のグラフィックスコードで実行されるようになります。

詳細については、[VFX の使用開始にあたって](#)を参照してください。



Package Manager からサンプルコンテンツをインストールする。

その他のサンプルコンテンツ

VFX Graph を初めて使用する場合は、**Package Manager** の **Samples** をインストールするのがよいでしょう。基本的なサンプルをいくつか紹介します。

- 新しい **Learning Templates: Learning Templates** (学習用テンプレート) サンプルは、VFX Graph の概念や機能を学べるように用意された VFX アセットのコレクションです。URP と HDRP 両方のレンダーパイプラインに対応しています。



新しい Learning Templates のサンプルでは、VFX Graph のさまざまな機能を紹介している。



- **Visual Effect Graph Additions:** このサンプルには、炎、煙、火花、稲妻のプレハブが含まれています。各サンプルでは、グラフの基本的なロジックと構成を示すために、エフェクトを必要最低限にしています。サンプルのプレハブの 1 つを **Hierarchy** ビューにドラッグアンドドロップするだけで、動作を確認できます。



Visual Effect Graph Additions パッケージのサンプル

ビルトインパーティクルシステム

ターゲットプラットフォームが VFX Graph の **最小システム要件** を満たしていない場合は、**ビルトインパーティクルシステム** も Unity でリアルタイムエフェクトを作成するための選択肢となります。ビルトインパーティクルシステムにアクセスするには、**GameObject > Effects > Particle System** を選択します。

VFX Graph と同様に、ビルトインパーティクルシステムでは、炎、爆発、煙、魔法の呪文など、**さまざまなエフェクト** を作成できます。VFX Graph よりもレンダリングされるパーティクルの数が少ないとはいえ、リアルタイムエフェクトにとっては貴重なツールです。

VFX Graph とビルトインパーティクルシステムの主な違いは、ハードウェアにあります。ビルトインパーティクルシステムが CPU 上でシミュレートされるのに対し、VFX Graph では演算の多くがコンピュータシェーダーに移され、GPU 上で実行されます。

VFX Graph には何百万ものパーティクルをシミュレートできるという利点がありますが、1 つ注意点があります。それは、GPU 上でシミュレートされるということは、データを読み取って CPU に戻したり、CPU 上の他のシステムとやり取りしたりすることが、演算の上では簡単ではないということです。

モバイルプラットフォームを使用している場合、VFX Graph を使用するには、コンピュータシェーダーをサポートしていることを確認する必要があります。そうでない場合は、CPU ベースのエフェクトには、ビルトインパーティクルシステムを使用する必要があるかもしれません。

下図に比較を示します。

	CPU Particle System	GPU VFX Graph
Particle count	Thousands	Millions
Simulation	Simple	Complex
Physics	Underlying physics system	Scene representation •Primitives (spheres, boxes, etc.) •Signed Distance Fields
Gameplay readback	Yes	No*
Other	-	Depth buffer Can read frame buffers

ビルトインパーティクルシステムと VFX Graph の比較

ビルトインパーティクルシステムは、基盤となる物理演算システムを使用し、ゲームプレイとの相互作用もより直接的ですが、パーティクル数が制限され、シミュレーションは比較的単純なものである必要があります。

Unity Asset Store から [Particle Pack](#) をダウンロードして、ビルトインパーティクルシステムを使用した一連のサンプルを入手しましょう。このサンプルアセットでは、ゲーム内のさまざまなエフェクト（炎、爆発、氷、ディゾルブなど）を紹介しています。ビルトインパーティクルシステムで Shader Graph を使用方法の詳細については、こちらの Dev Takeover を参照してください。

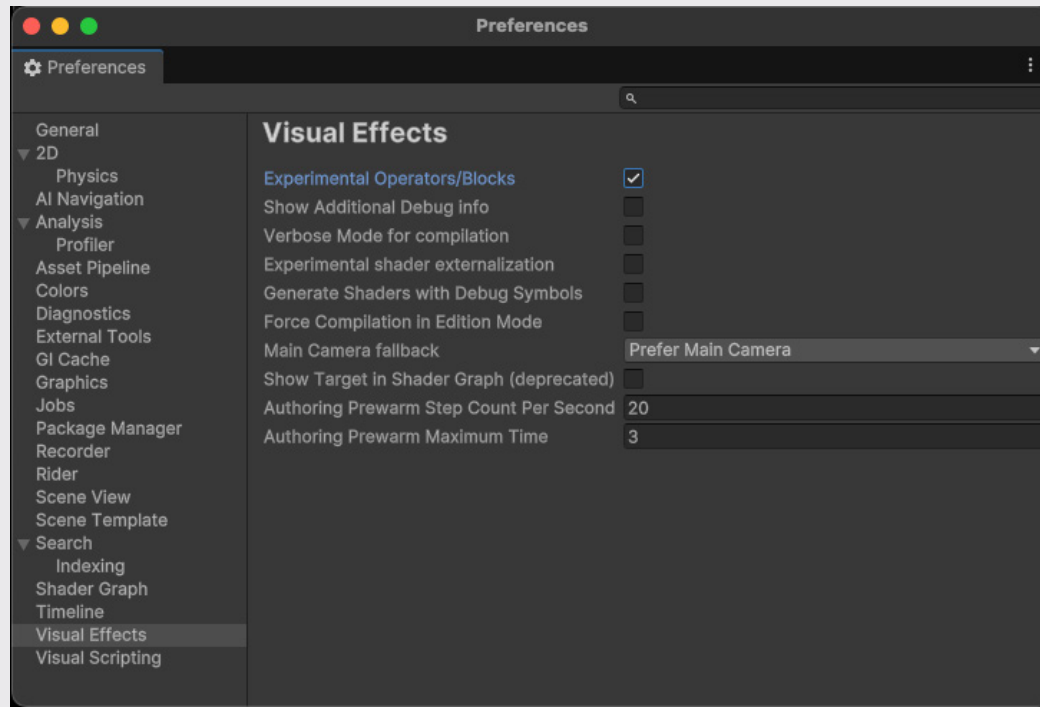
URP はビルトインパーティクルシステム用の標準シェーダー（[Lit](#)、[Unlit](#)、[Simple Lit](#)）を、HDRP は [HDRP パッケージサンプル](#) から Shader Graph ベースのシェーダーを提供しています。レンダーパイプラインのパーティクルシステム機能の比較は[こちら](#)で確認できます。



Particle Pack は Asset Store で入手できる。

注：実験的な機能は完全には検証されていないため、変更される可能性があります。制作作業には、フルリリースバージョンの Unity をお勧めします。Unity 6 のリリースによる命名規則の変更については、[こちらに投稿された記事](#)を参照してください。

このガイドの実験的な機能を有効にするには、下図に示すように、**Preferences > Visual Effects > Experimental Operators/Blocks** を選択します。

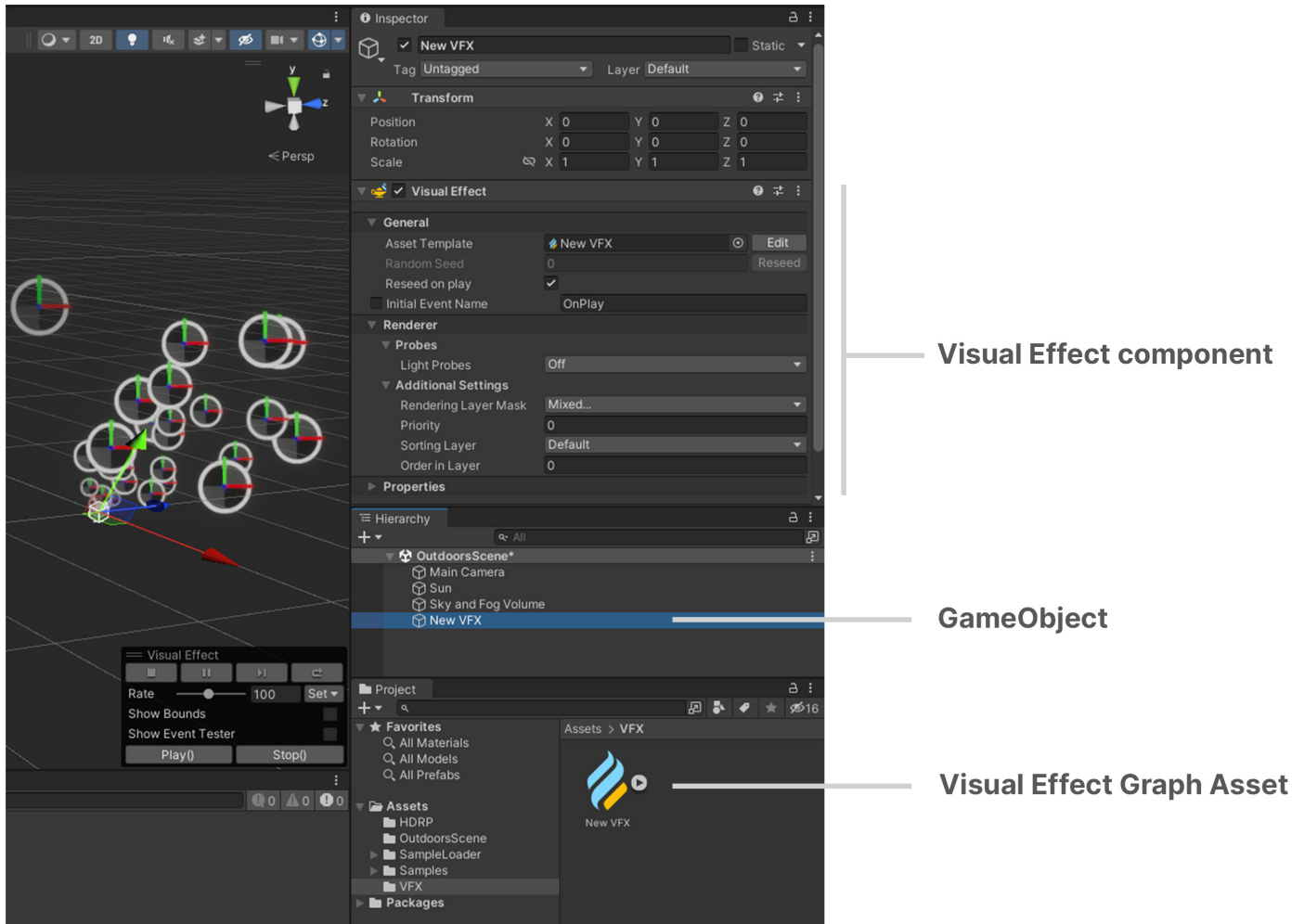


VFX Graph 入門

VFX Graph のビジュアルエフェクトは、次の 2 つの部分で構成されています。

- シーン内のゲームオブジェクトにアタッチされた [Visual Effect \(VFX\) コンポーネント](#)
- プロジェクトレベルに存在する [Visual Effect \(VFX\) Graph アセット](#)

Unity は各 VFX Graph を Assets フォルダーに保存するため、各アセットをシーン内の Visual Effect コンポーネントに接続する必要があります。ランタイム時には、異なるゲームオブジェクトが同じグラフを参照できることに注意してください。



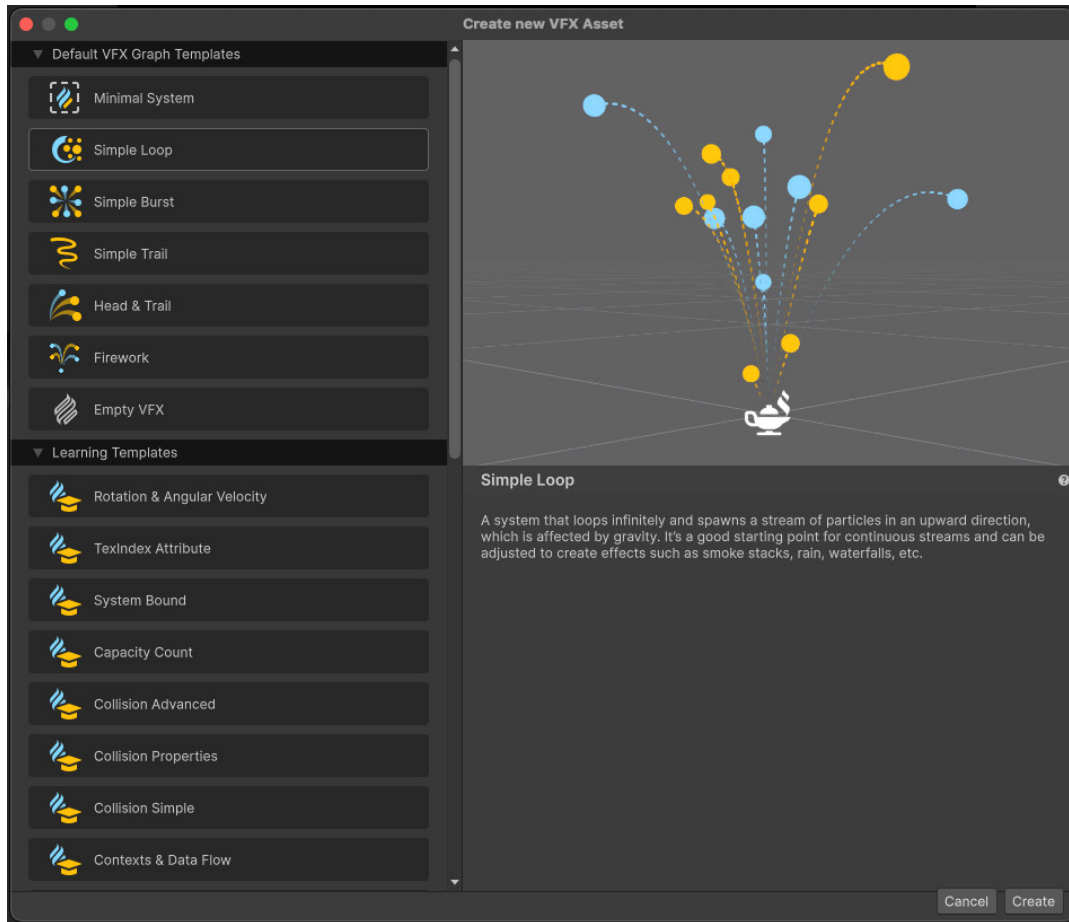
VFX Graph アセットと Visual Effect コンポーネント

VFX Graph アセットとコンポーネント

新しいビジュアルエフェクトを作成するには、**Project** ウィンドウで右クリックし、**Create > Visual Effects > Visual Effects Graph** に移動します。

作成ウィザードが開き、開始テンプレートを選択できます。デフォルトの VFX Graph の 1 つから始めることも、サンプルから Learning Templates の 1 つを選ぶこともできます。

Learning Templates は、Package Manager に追加パッケージがインストールされていないとウィンドウに表示されません。



テンプレートを選択して新しい VFX Graph を作成する。

Create を選択して、プロジェクトに **VFX Graph アセット** を作成します。

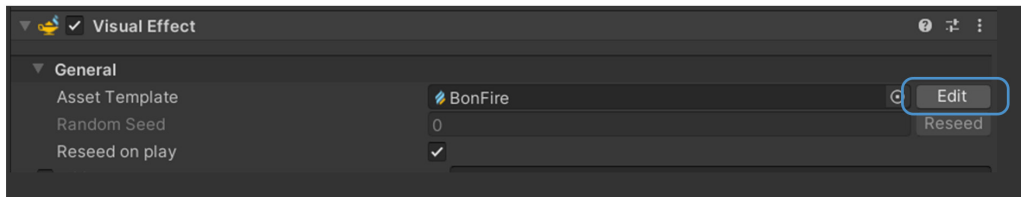
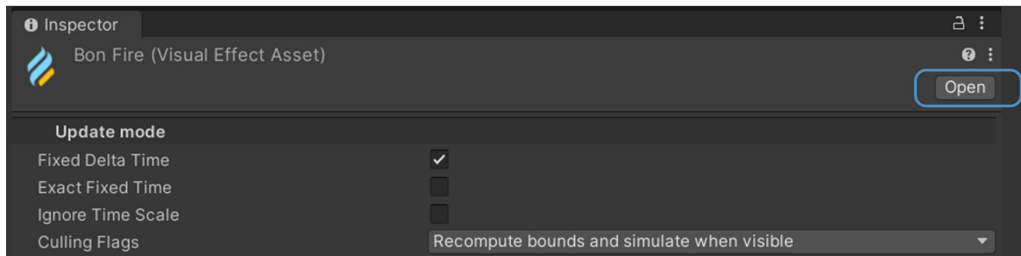
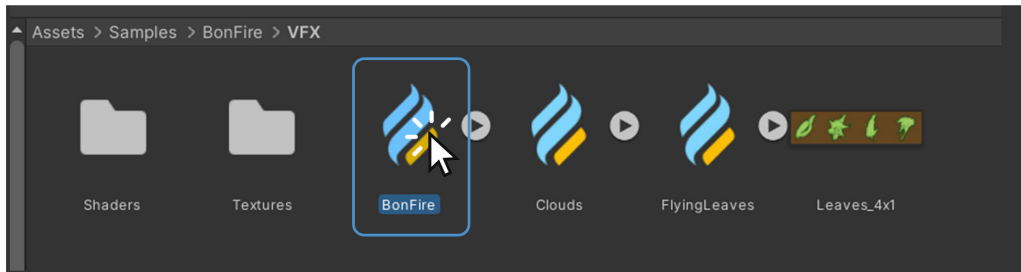
エフェクトをシーンに追加するには、**Visual Effect コンポーネント** をゲームオブジェクトにアタッチしてから、VFX Graph アセットを接続します。これを行う方法はいくつかあります。

- 作成したアセットを Scene ビューまたは Hierarchy にドラッグします。新しいデフォルトのゲームオブジェクトが Hierarchy ウィンドウに表示されます。
- Inspector で既存の **Visual Effect コンポーネント** にアセットを割り当てます。Hierarchy で右クリックして空のゲームオブジェクトを作成する (**Visual Effects > Visual Effect**) ことも、ゲームオブジェクトを作成してから、Visual Effect コンポーネントを手動で追加することもできます。
- ゲームオブジェクトを選択した状態で、アセットを Inspector ウィンドウにドラッグアンドドロップします。これにより、Visual Effect コンポーネントが作成され、アセットがワンアクションで割り当てられます。

VFX Graph アセットにはすべてのロジックが含まれています。次のいずれかの方法を選択して動作を編集します。

- Project ウィンドウで VFX Graph のアセットをダブルクリックします。
- Project ウィンドウで VFX Graph のアセットを選択し、ヘッダーの **Open** ボタンをクリックします。

- Visual Effect コンポーネントの **Asset Template** プロパティの横にある **Edit** ボタンをクリックします。



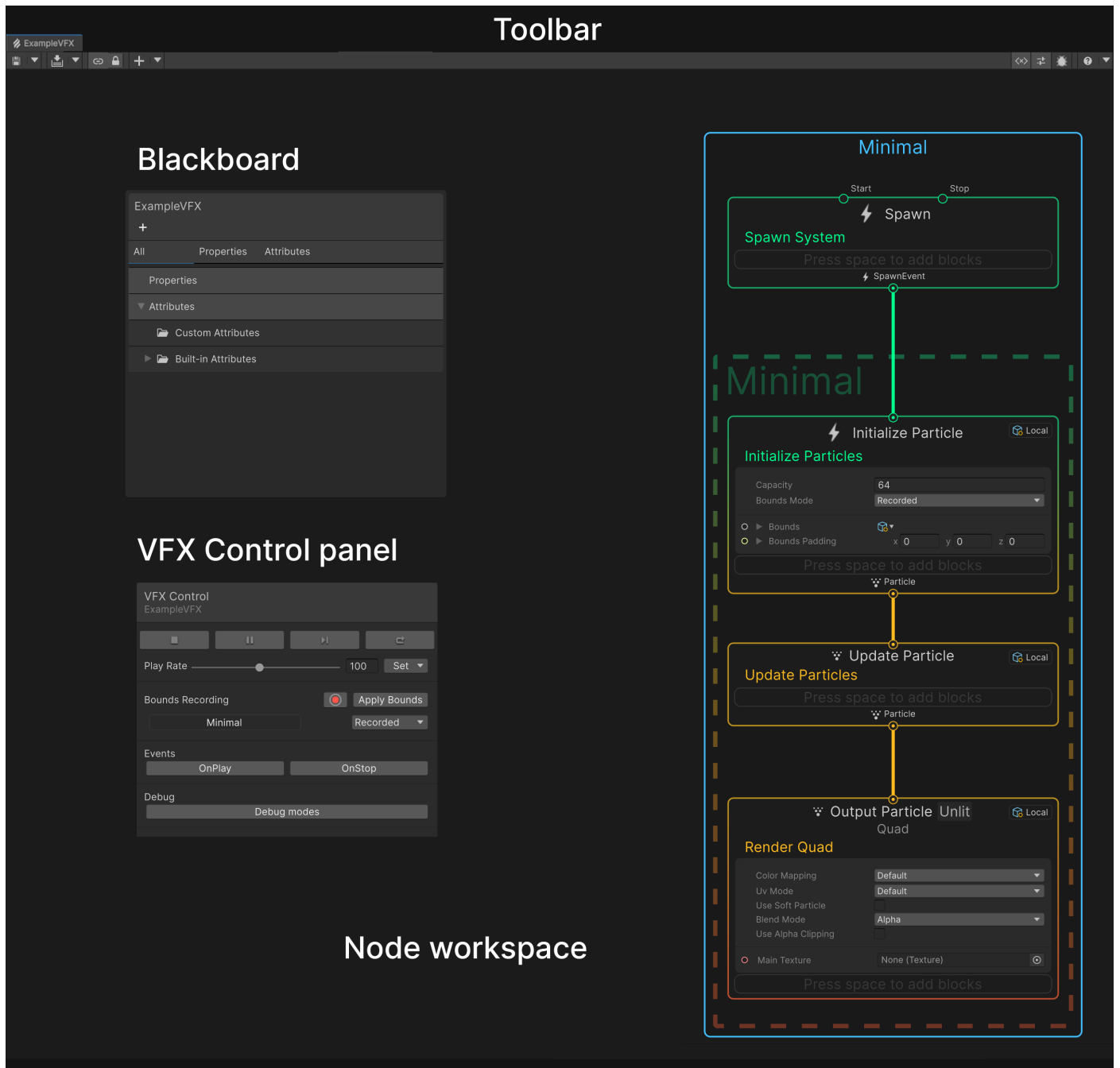
VFX Graph を開く 3 つの方法

アセットが **Window > Visual Effects > VFX Graph** にある **VFX Graph ウィンドウ** で開きます。

VFX Graph ウィンドウ

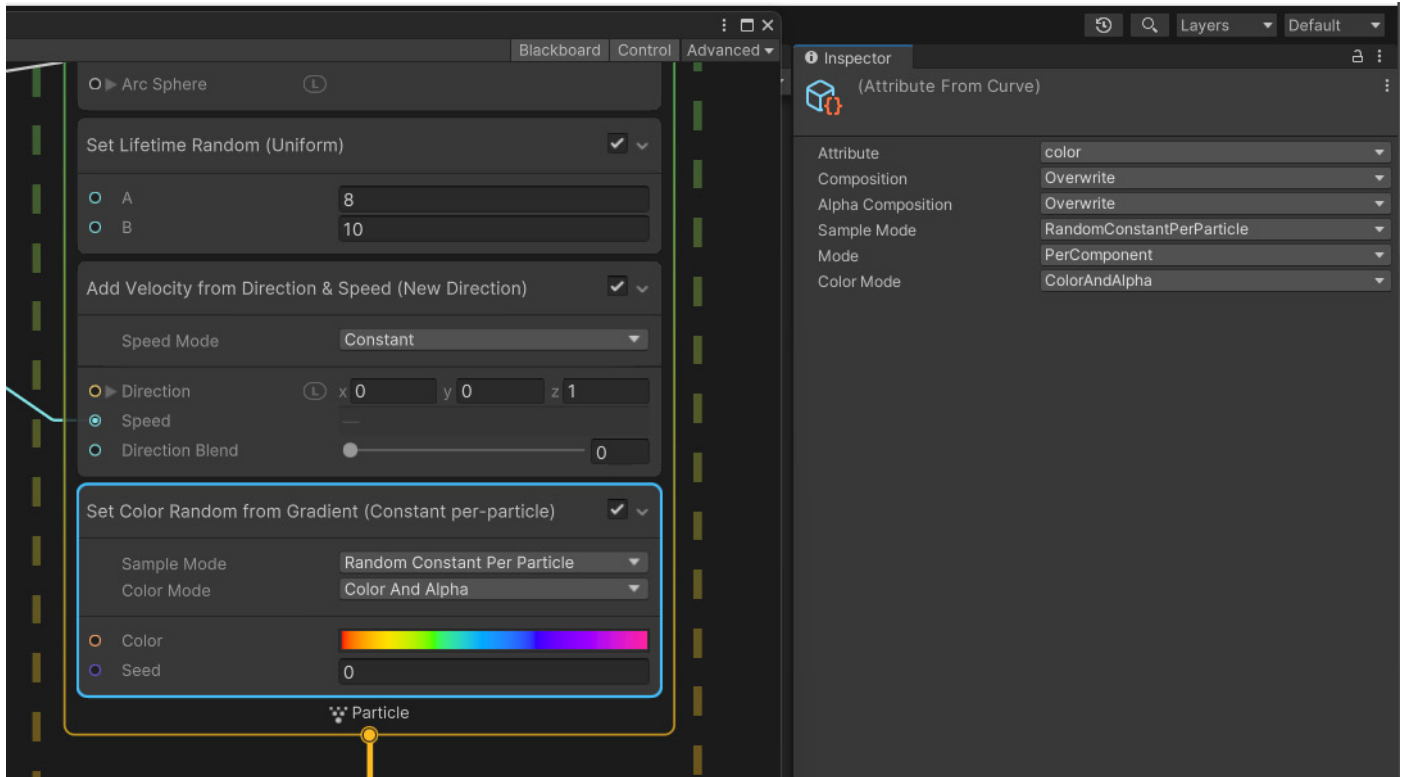
この**ウィンドウレイアウト**に慣れてください。下記のものがあります。

- **Toolbar**: グローバル設定へのアクセスとパネルの切り替えを行う
- **ノードワークスペース**: VFX Graph の作成と編集を行う
- **Blackboard**: グラフ全体で再利用可能な属性とプロパティを管理する
- **VFX Control パネル**: アタッチされているゲームオブジェクトの再生を制御する



VFX Graph ウィンドウ

エディターのレイアウトに Inspector 用のスペースを空けておいてください。グラフの一部を選択すると、パーティションオプションやレンダリング状態などの特定のパラメーターが表示されることがあります。

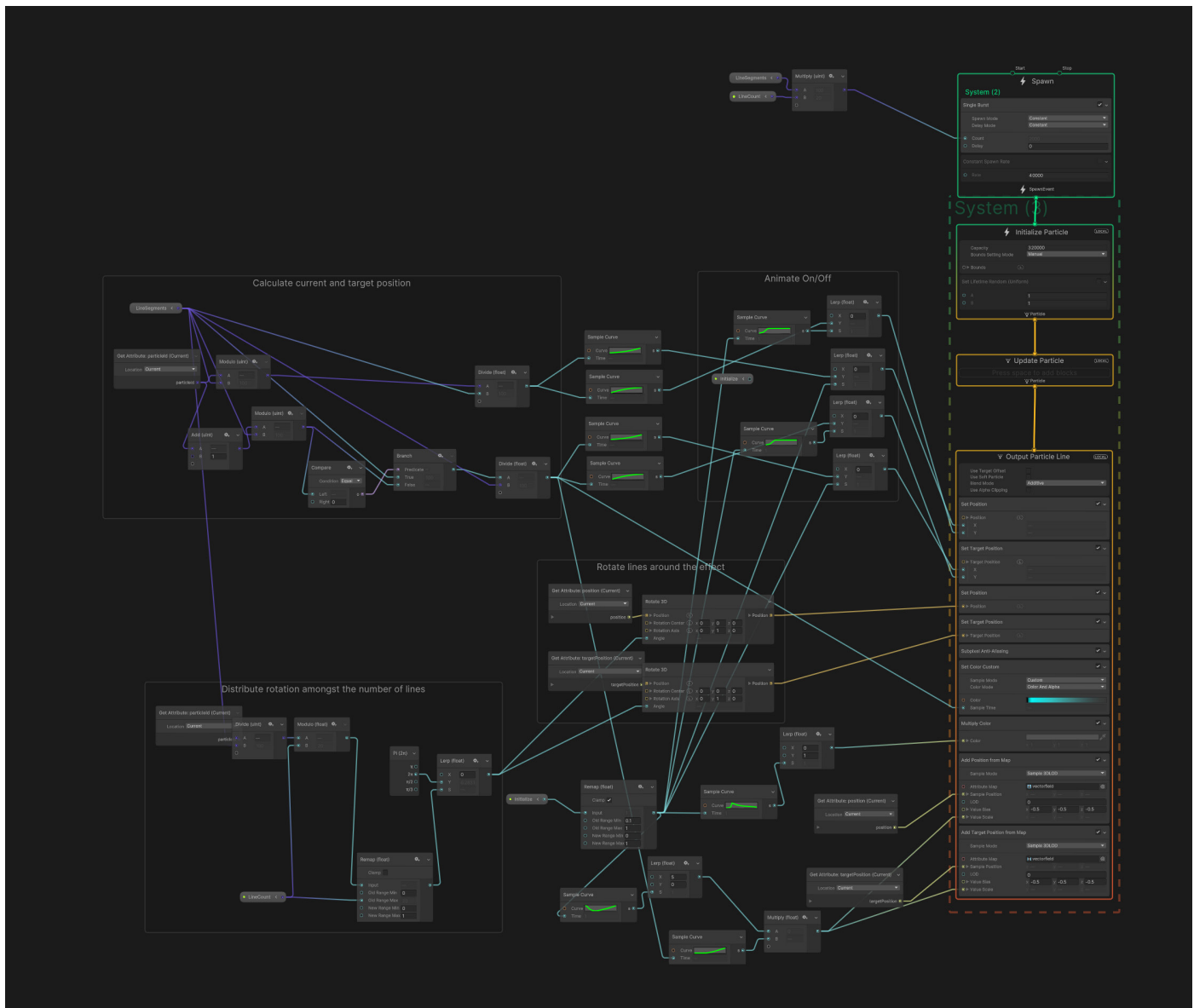


Inspector を使用して特定のパラメーターを変更する。

グラフロジック

ビジュアルエフェクトは、ウィンドウのワークスペース内のノードのネットワークから構築する必要があります。VFX Graph は、**Shader Graph** などの他のノードベースのツールに似たインターフェースになっています。

スペースキーを押すか、右クリックして新しいグラフ要素を作成します。空のワークスペースにマウスを置き、**Create Node** を選択して、グラフの**コンテキスト**、**オペレーター**、**プロパティ**を作成します。既存のコンテキストの上にマウスを持っていけば、**Create Block** を使用できます。



VFX Graph は複雑なネットワークを構成可能

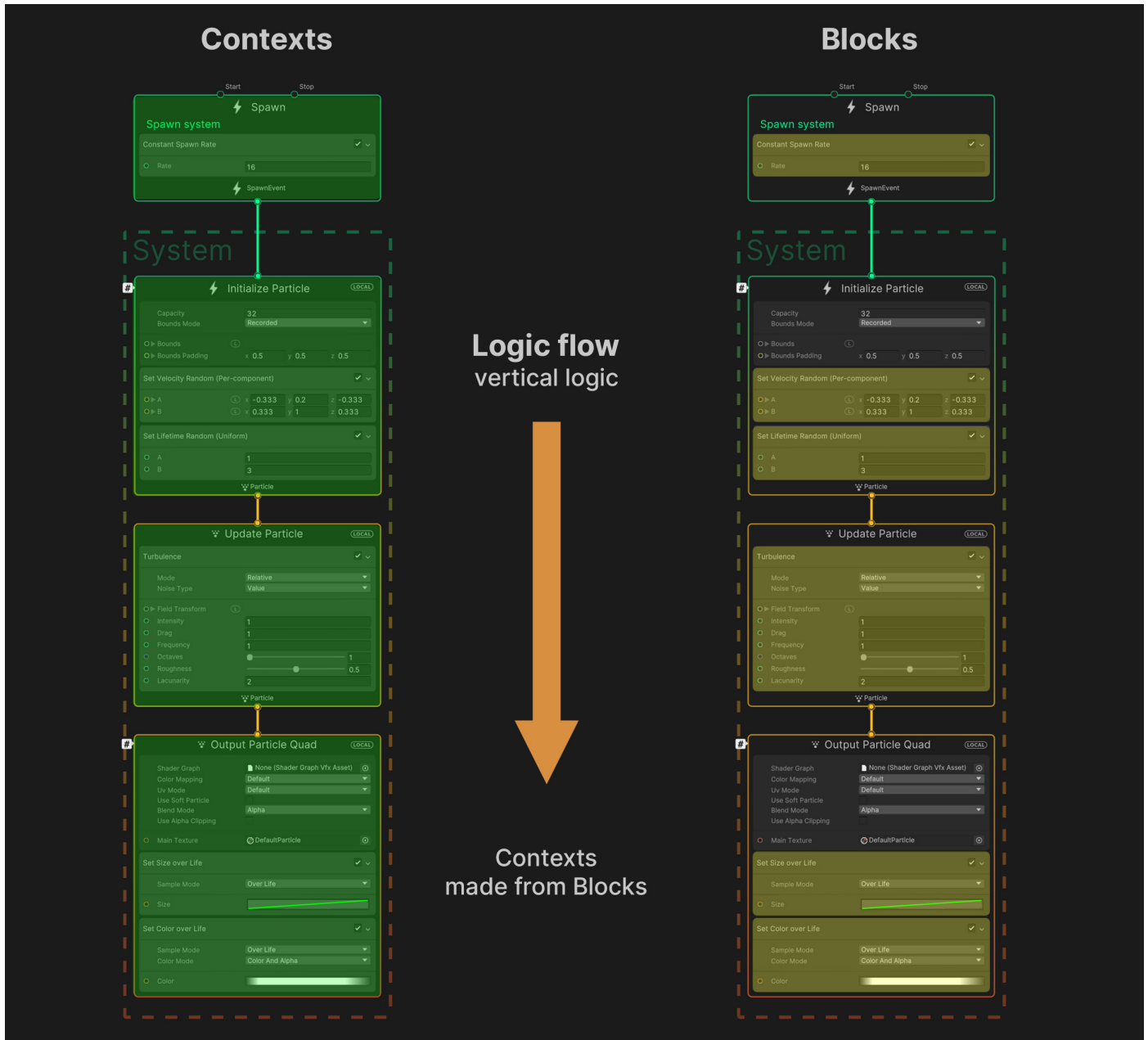
複雑な VFX Graph を始めるのは、はじめは大変かもしれませんが、ですが、幸いなことに、プロダクションレベルのグラフには数百のノードを含めることができますが、どのグラフもサイズに関係なく同じルールセットに従います。

VFX Graph の各パーツがどのように連携するのか、詳しく学びましょう。

システム、コンテキスト、ブロック

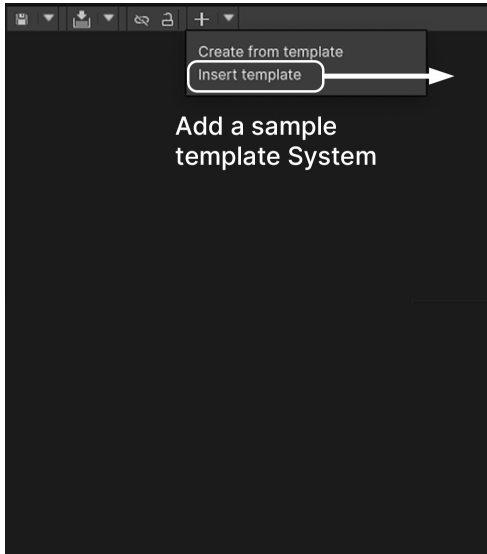
VFX Graph には、**システム**と呼ばれる 1 つ以上の垂直スタックがあります。システムは、グラフのスタンドアロン部分を定義し、いくつかの**コンテキスト**を格納します。システムは、システムを構成するコンテキストを囲む点線によって表されます。

各コンテキストは個々のブロックからなり、パーティクルやメッシュの属性（サイズ、色、速度など）を設定できます。複数のシステムを1つのグラフ内で連動させ、最終的なビジュアルエフェクトを作成できます。

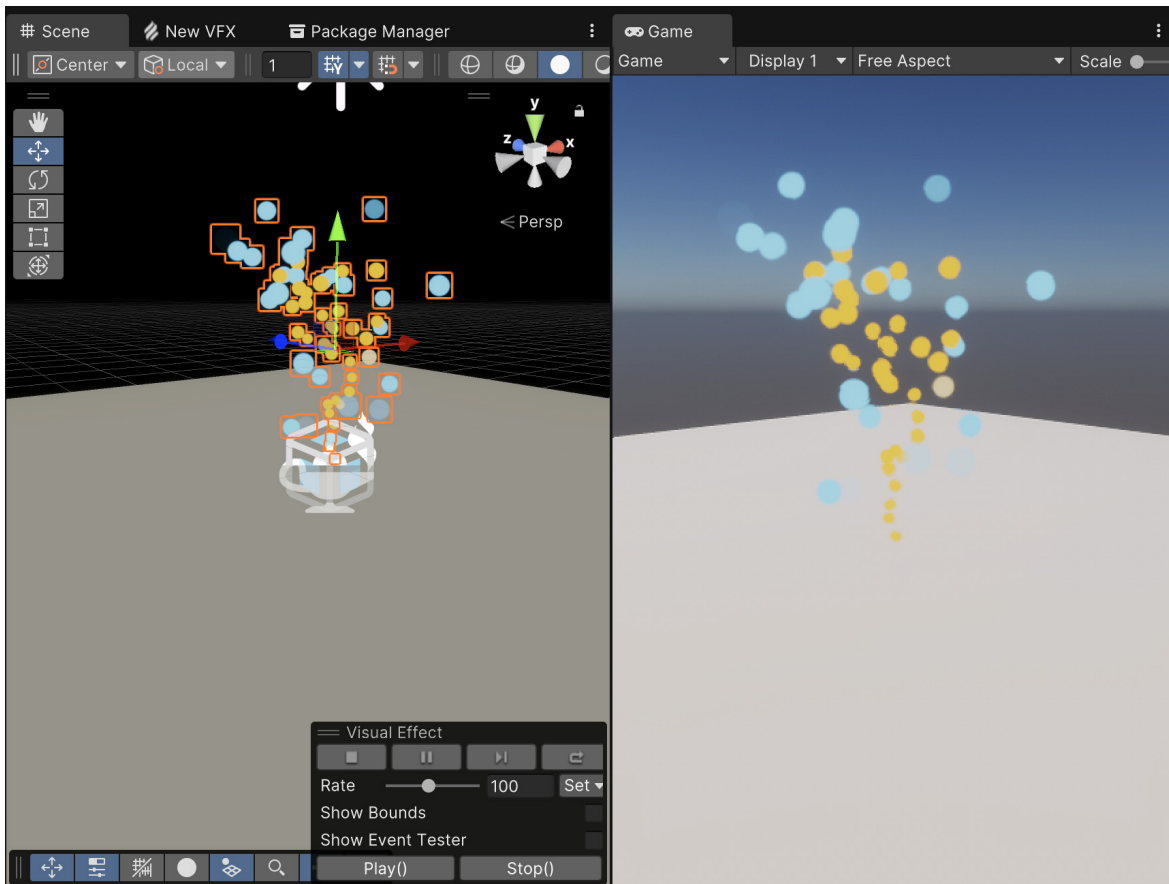
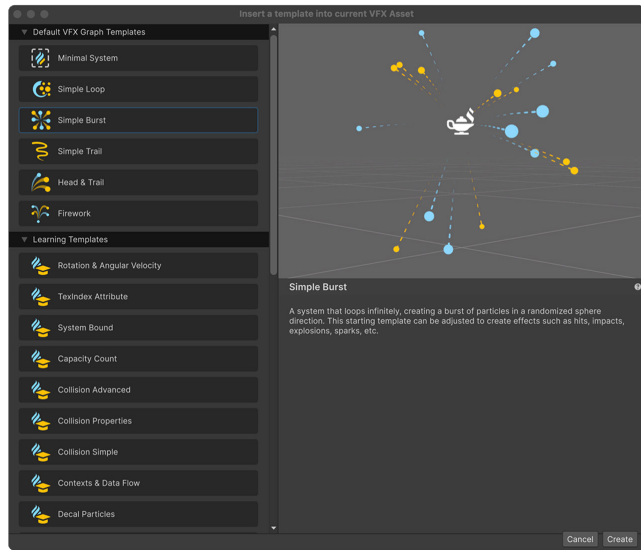


グラフの垂直ロジックは下方向に流れます。

メニューのドロップダウンから **Insert template** を選択し、既存テンプレートのサンプルシステムを現在の VFX Graph に追加します。これは、あらかじめ用意されているグラフロジックを使用して作業を開始するのに役立ちます。シンプルなシステムであれば Default VFX Graph Templates のいずれかを選択し、目的のエフェクトに似たものがある場合は Learning Templates から選択します ([VFX サンプルコンテンツを詳しく見る](#)を参照)。

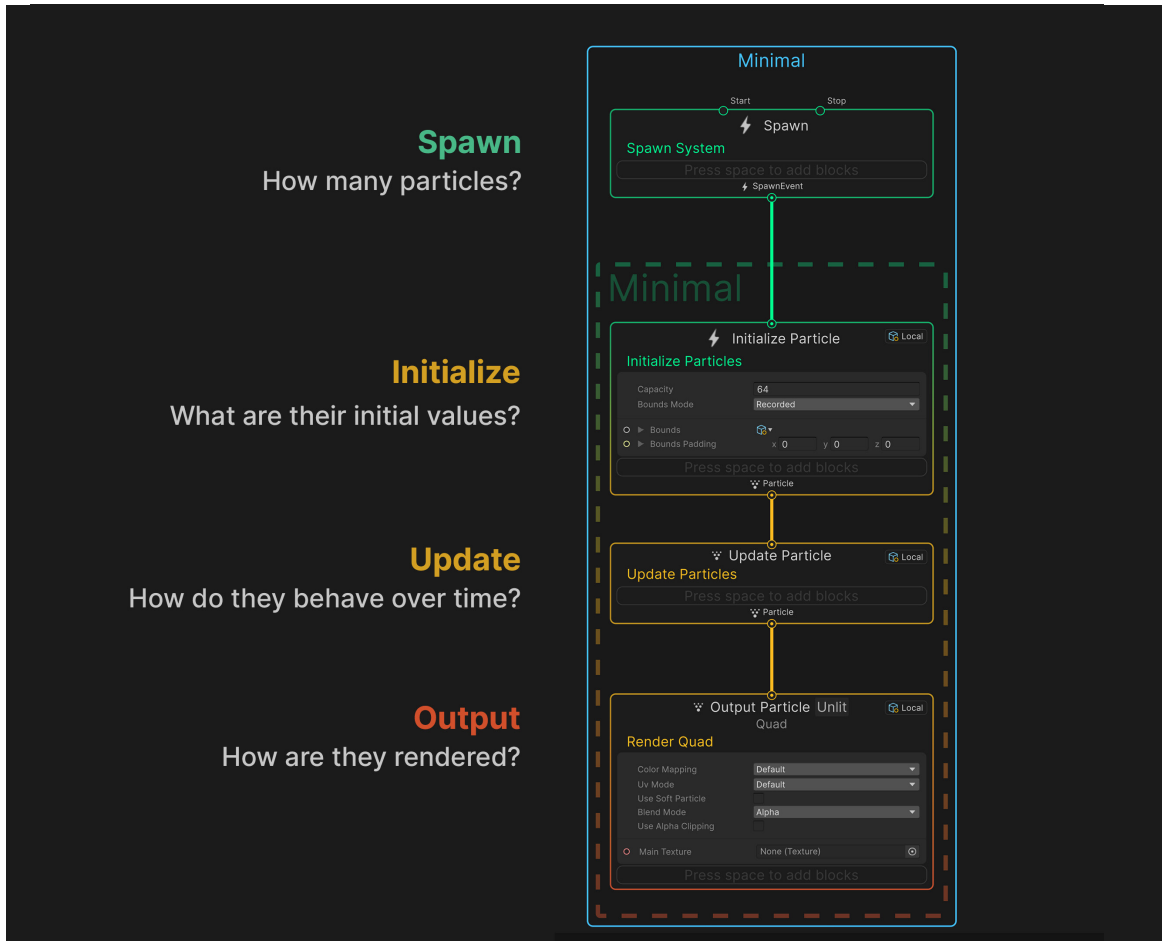


メニューからテンプレートを挿入する。



Scene ビューと Game ビューのビジュアルエフェクト

Default VFX Graph Templates から **Minimal System** テンプレートを選択すると、次のような 4 つの部分が含まれる基本的なシステムになります。



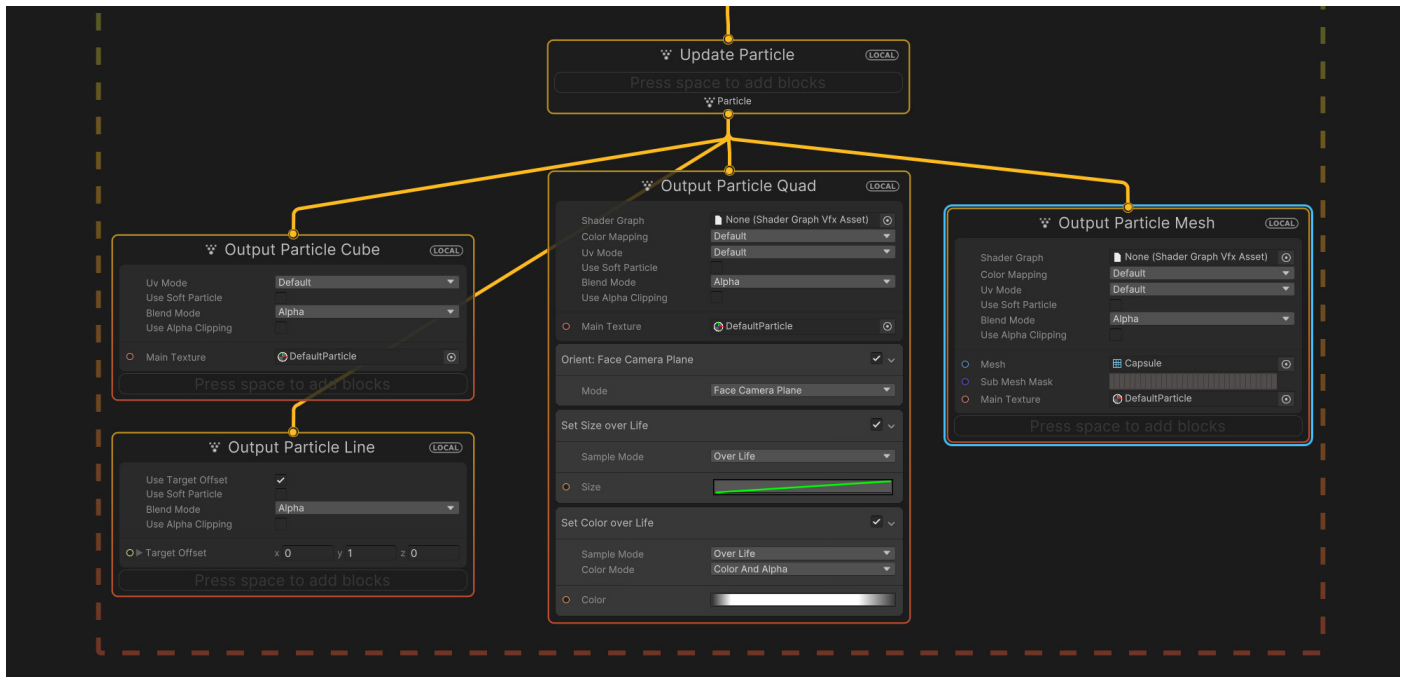
デフォルトテンプレートの Minimal System

コンテキスト間のフローによって、パーティクルのスポーンとシミュレーションの方法が決まります。各コンテキストは、1つの演算ステージを定義します。

- **Spawn** (スポーン): 作成するパーティクルの数とそれらをスポーンするタイミングを決定 (1回のバースト、ループ、遅延など)。
- **Initialize** (初期化): パーティクルの開始属性、キャパシティ (パーティクルの最大数)、バウンド (エフェクトがレンダリングされる境界) を決定
- **Update** (更新): 各フレームのパーティクルのプロパティの変更。力の適用、アニメーションの追加、衝突の作成、符号付き距離フィールド (SDF) などのインタラクションを設定可能
- **Output** (出力): パーティクルをレンダリングし、最終的な見た目 (色、テクスチャ、向き) を決定。各システムは柔軟性を最大化するために複数出力可能

システムとコンテキストは、グラフの“垂直論理”、つまり**処理ワークフロー**のバックボーンを形成します。システム内のデータは上から下に流れ、その過程で遭遇する各コンテキストがシミュレーション結果に応じてデータに変更を加えます。

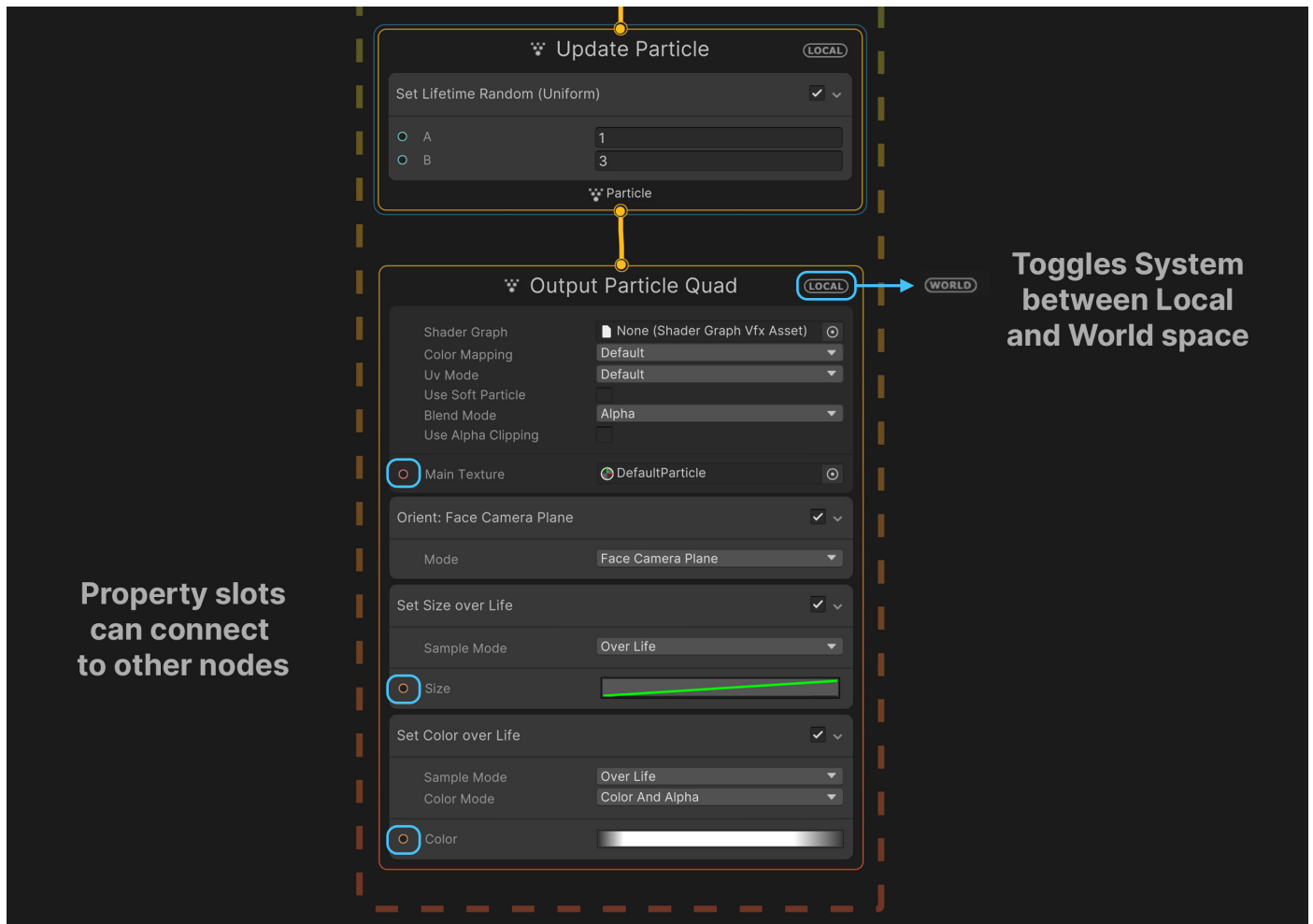
システムは柔軟なので、必要に応じてコンテキストを省略したり、複数の出力をリンクしたりできます。この例では、同一システム内で複数の **Output コンテキスト** をレンダリングしています。



同一システム内の複数の Output コンテキスト

コンテキスト自体の動作は、コンテキストを構成するブロックによって異なります。ブロックも同様に上から下の順にデータを計算します。ブロックをさらに追加して操作し、データを処理させることができます。

コンテキストの右上にあるボタンをクリックして、システムのシミュレーションスペースを **Local** と **World** の間で切り替えます。



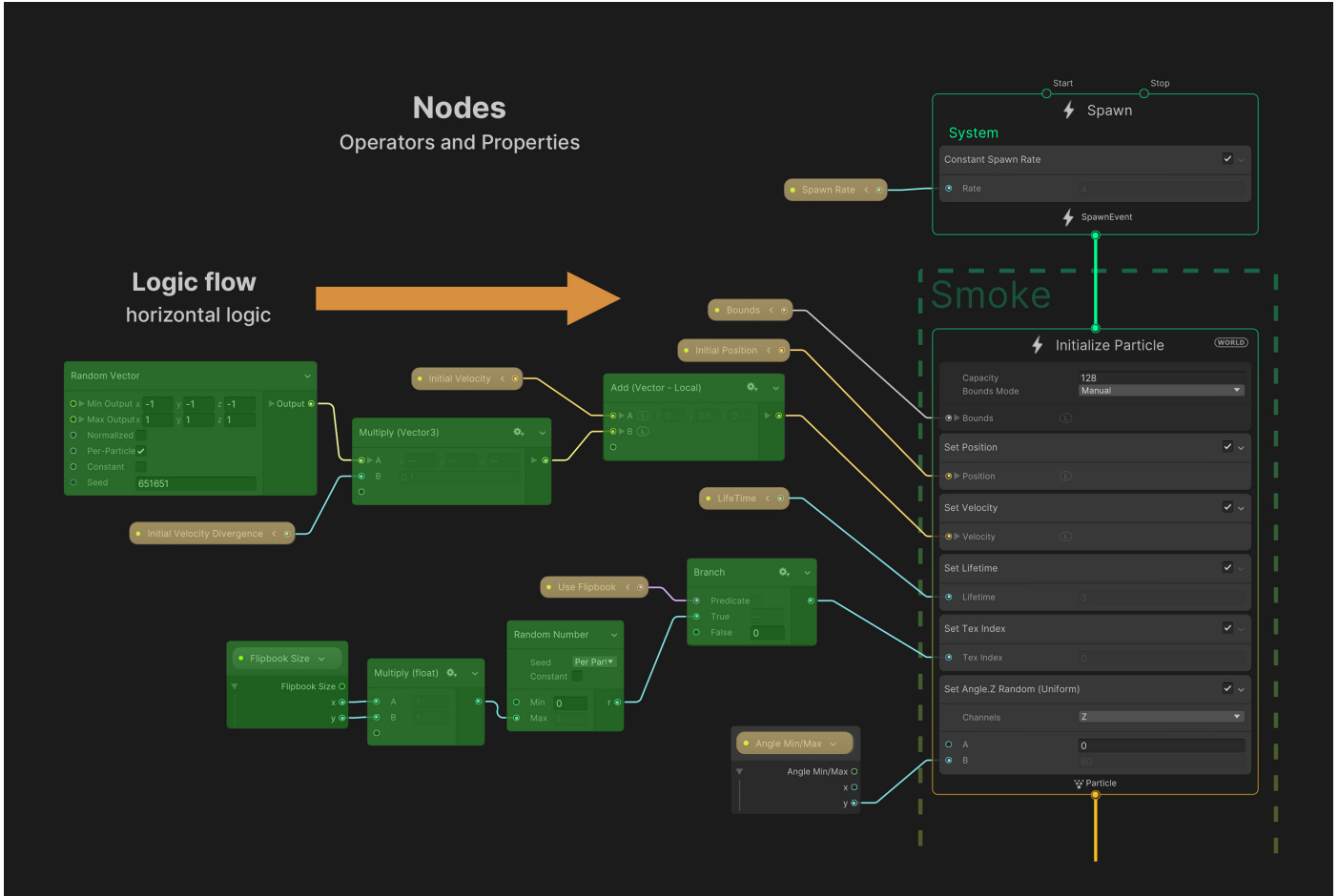
さまざまなブロックの例

ブロックは、単純な値を格納する **Color** (色) から、**Noise** (ノイズ)、**Force** (力)、**Collision** (衝突) などの複雑な演算まで、ほぼすべてのことを実行できます。多くの場合、左側にスロットがあり、オペレーターやプロパティから入力を受けることができます。

コンテキストとブロックの完全なリストについては、[ノードライブラリ](#)を参照してください。

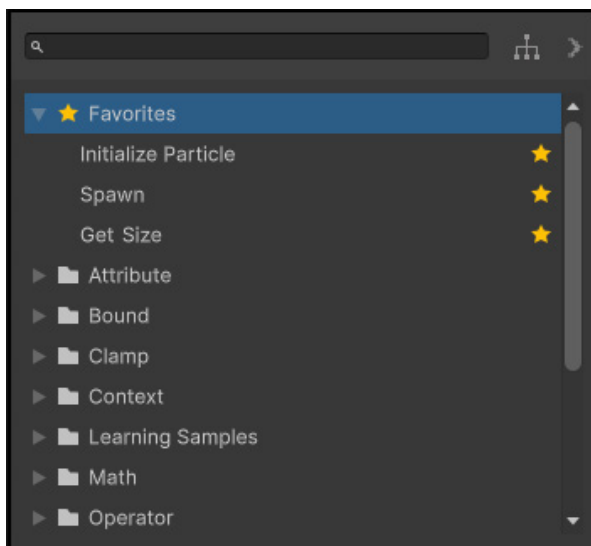
プロパティとオペレーター

システムがグラフの垂直論理の大部分を構成するように、オペレーターはその[プロパティワークフロー](#)の“水平論理”を構成します。これは、カスタムの式や値をブロックに渡すのに役立ちます。



水平論理

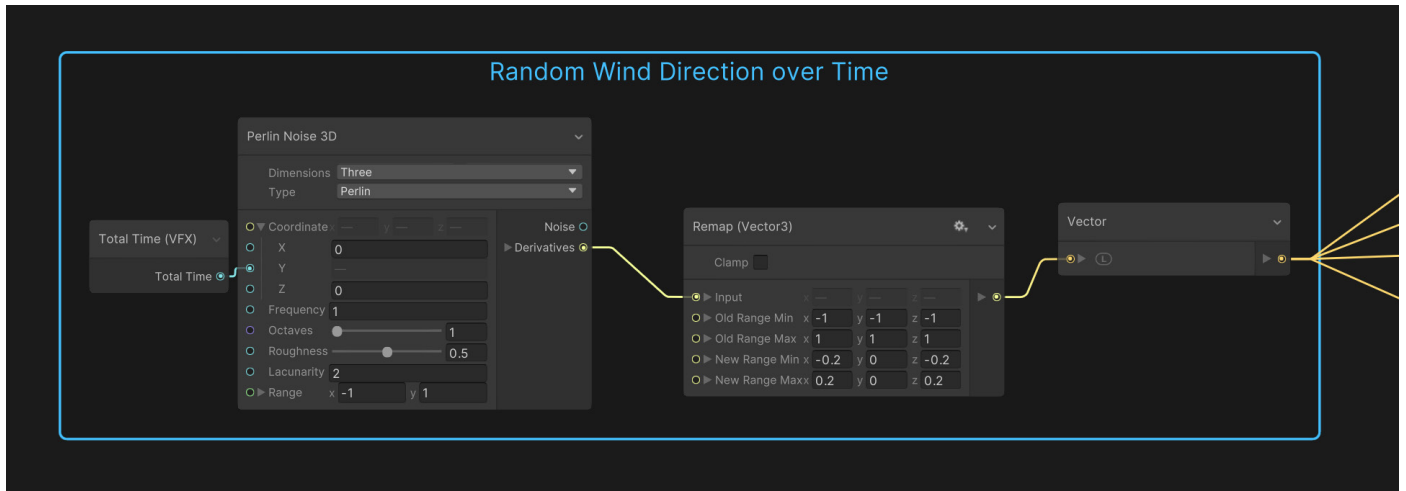
Shader Graph のノードと同様に、オペレーターは左から右に流れます。値の処理やさまざまな演算の実行に使用できます。



メニューからオペレーターノードを作成する。

Create Node メニュー（右クリックするかスペースバーを押す）を使用して、**オペレーターノード**を作成します。また、プロパティスロットから Edge Connection をドラッグして何もないスペースで放すと、同じメニューが開き、対応するオペレーターのみが表示されます。

例えば、Bonfire サンプルのオペレーターは、ランダムな風向きを計算します。

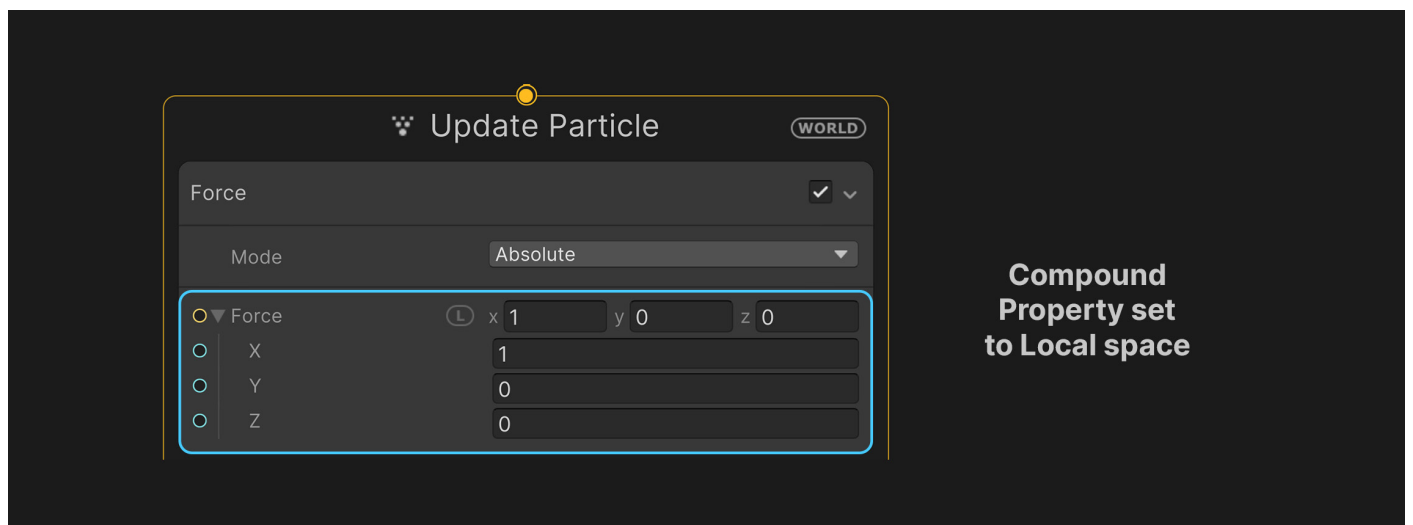


Bonfire サンプルでの風向きの決定方法

プロパティは、[プロパティワークフロー](#)を使用してグラフ要素に接続する、編集可能なフィールドです。プロパティには次のものがあります。

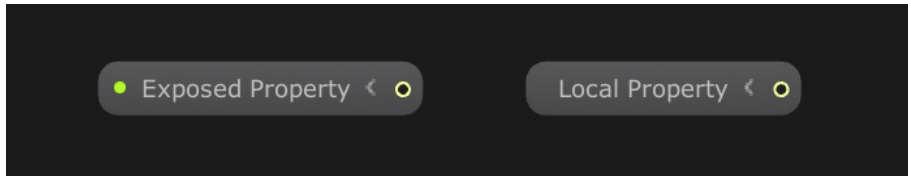
- 整数、Float、ブーリアンを含む任意の型
- ベクトルや色などの複合コンポーネントから作成
- キャストおよび変換（例：整数から Float へ）
- Local または World 空間。L か W をクリックして切り替え

プロパティの値は、グラフ内の実際の値に応じて変化します。入力ポート（プロパティの左側）を他のグラフノードに接続できます。



ブロック内の Force プロパティ

プロパティノードは、グラフのさまざまなポイントで同じ値を再利用できるオペレーターです。これには、Blackboard に表示される対応するグローバルなプロパティがあります。



プロパティノード

Blackboard

Blackboard ユーティリティパネルは、**プロパティ**と**属性**を管理します。これを開くには、ウィンドウの Toolbar にある **Blackboard** ボタンをクリックするか、デフォルトの Shift + 1 ショートカットを使用します。

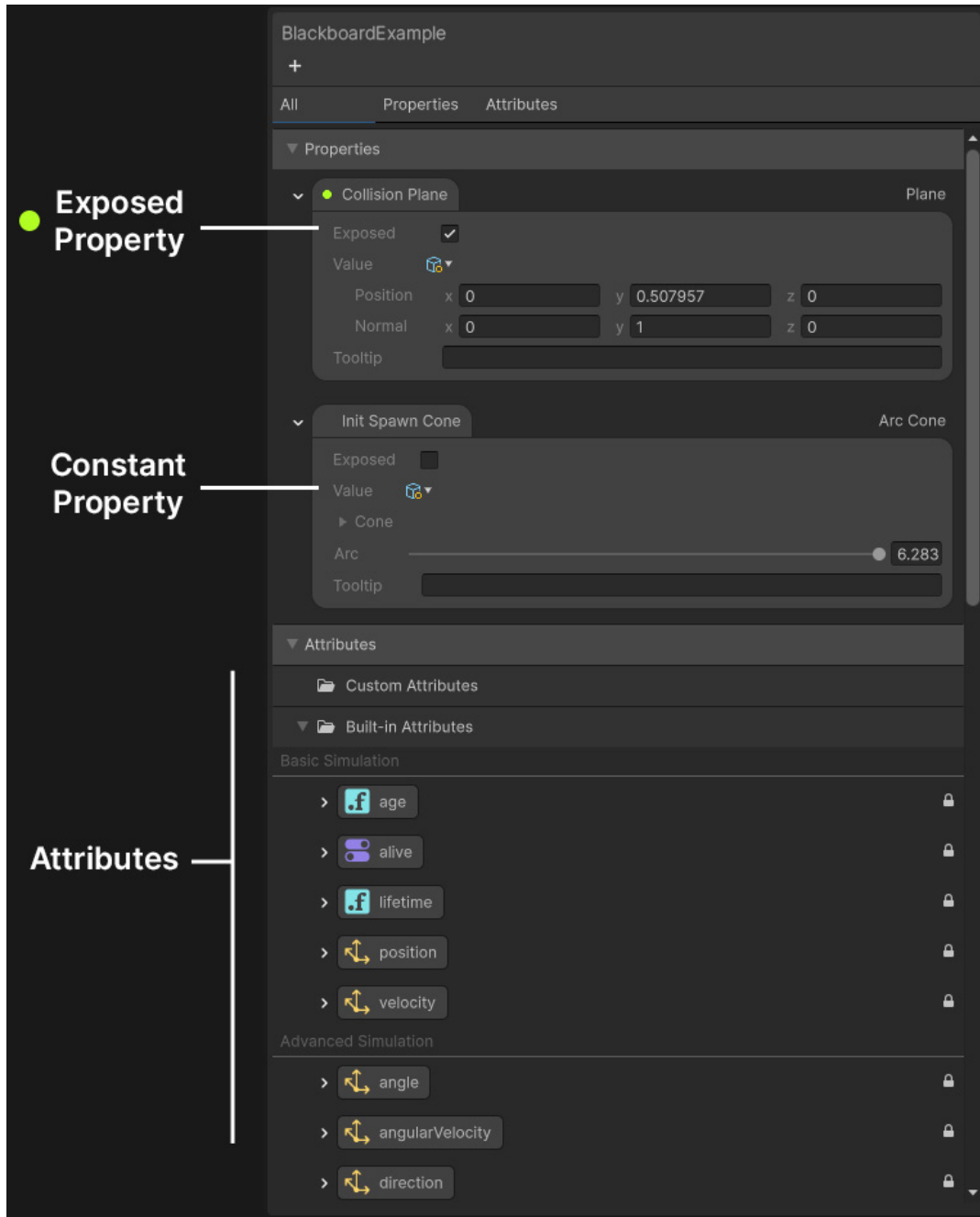
プロパティと属性をまとめて表示するには、Blackboard の上部にある **All** タブを選択します。タイプ別にフィルターをかけるには、該当する Properties (プロパティ) タブまたは Attributes (属性) タブを選択します。

Blackboard で定義したプロパティは、グラフ全体で**プロパティノード**として再利用できるグローバル変数として機能します。例えば、バウンディングボックスのプロパティを一度定義すれば、同じグラフ内の複数のパーティクルシステムに適用できます。

Blackboard のプロパティは次のとおりです。

- **Exposed** (公開): Exposed Property の左側にある緑の丸は、グラフの外部で表示および編集できることを示します。Exposed Property クラスを使用したスクリプトにより、Inspector の Exposed Property にアクセスします。
- **Constant** (定数): 緑の丸のない Blackboard プロパティは Constant です。グラフ内で再利用できますが、Inspector には表示されません。

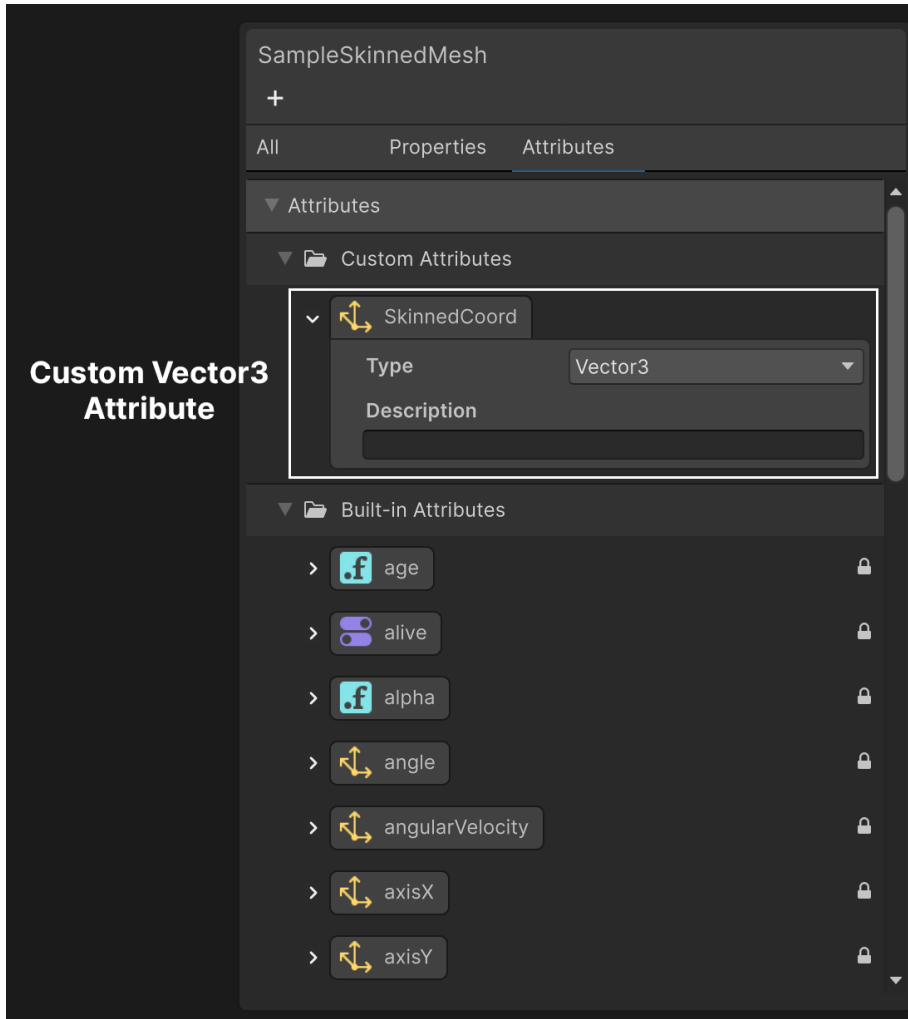
新しいプロパティはデフォルトで Exposed に設定されており、Inspector に表示されます。グラフの外部でプロパティを非表示にする場合は、**Exposed** オプションのチェックをはずし、**Categories** を作成してプロパティを整理する必要があります。



Blackboard と利用可能なプロパティ

また、Blackboard はビルトインとカスタムの両方の属性を管理し、グラフにドラッグアンドドロップするか、インターフェースから直接作成できます。各属性には簡単な説明があります。属性にカーソルを合わせると、その属性がグラフ内で強調表示されます。

詳細については、後述の [Blackboard の属性](#) を参照してください。



Sample Skinned Mesh テンプレートには、カスタム属性が含まれている。

グループノードと Sticky Note

グラフのロジックが大きくなったら、グループノードと [Sticky Notes](#)（付箋）を使用してわかりやすくしましょう。グループノードを使用すると、一連のノードにラベルを付けて 1 つのノードとして移動させることができます。また、Sticky Note はコードのコメントのように機能します。

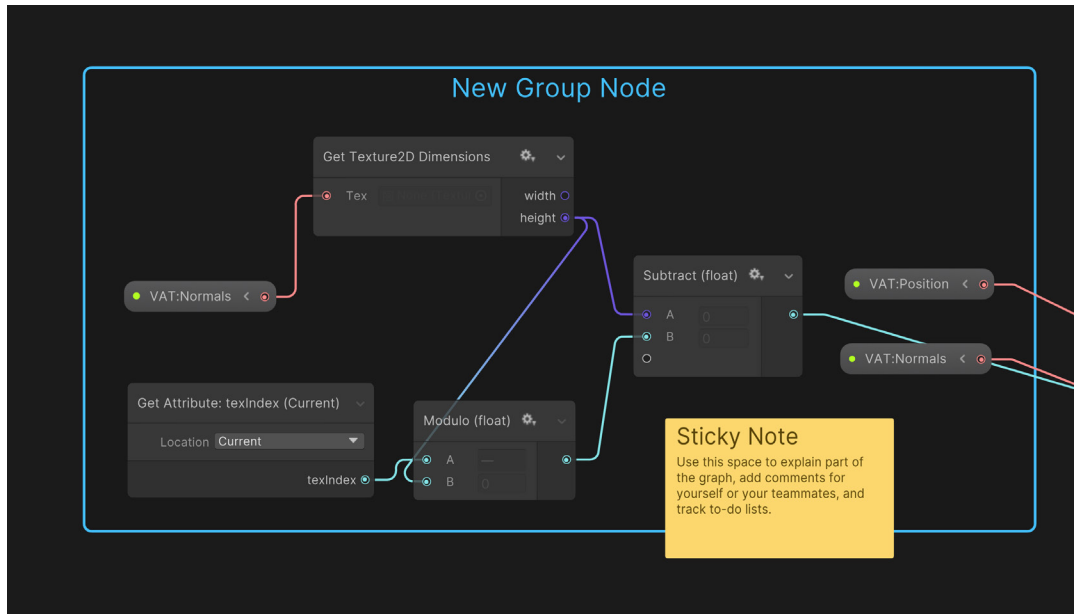
グループノードを作成するには、ノードのグループを選択して右クリックし、**Group Selection** を選択します。新しいデフォルトのショートカット、Shift + G を使用することもできます。

既存のグループノードにノードをドラッグアンドドロップすることもできます。Group の上にノードを持っていき、Group がハイライトされたら放します。Group からノードを削除するには、Shift キーを押しながらドラッグして外に出します。

Delete キーまたは右クリックメニューからグループノードを削除しても、その中にあるノードは削除されません。

一方、Sticky Note を使用すると、グラフのセクションの動作を説明したり、自分自身やチームメイトにコメントを残したりできます。必要なだけ Sticky Note を追加し、自由に移動したり、サイズを変更したりできます。

各 Sticky Notes にはタイトルと本文があります。グラフビューで右クリックして Sticky Note を作成します。テキスト領域をダブルクリックして内容を編集します。右クリックメニューからテーマの色（ダーク / ライト）とテキストサイズを設定し、Sticky Note を整理します。



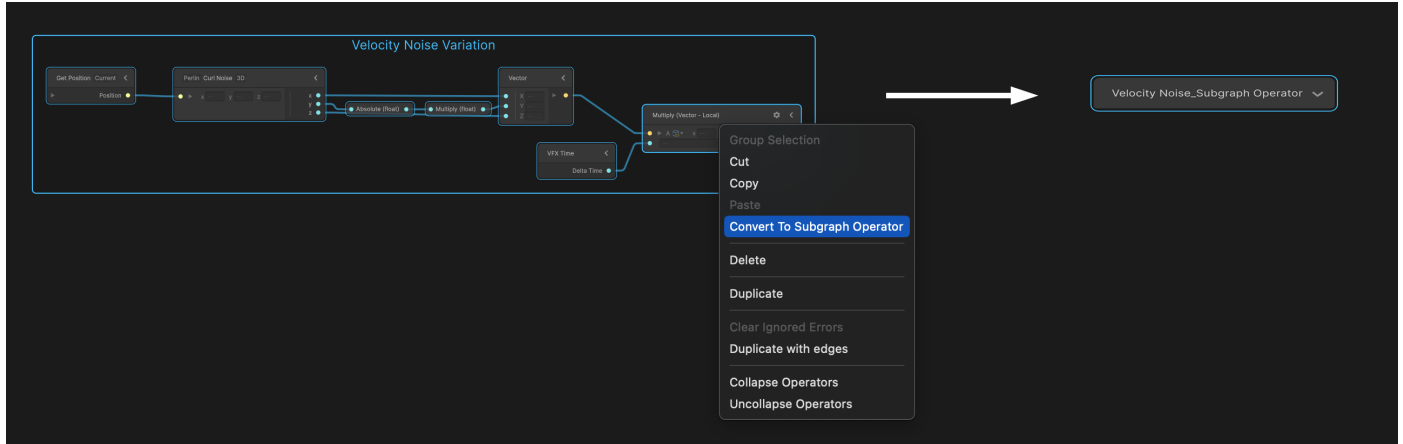
グループノードで作業し、Sticky Note を追加する。

サブグラフ

サブグラフは 1 つのノードとして表示されるため、グラフのロジックを整理するのに役立ちます。これを使用すると、VFX Graph の一部を別のアセットとして保存し、別の VFX Graph にドロップして再編成し、再利用できます。システム、ブロック、オペレーターを、さまざまなタイプのサブグラフにパッケージできます。

サブグラフは Project ウィンドウから直接作成できます。新規のサブグラフをゼロから作成するには、**Create > Visual Effect > Subgraph Operator** または **Subgraph Block** を選択します。この方法では、既存の VFX Graph 内にサブグラフを構築せずにサブグラフを設計できます。

あるいは、VFX Graph の編集時に一連のノードを選択し、右クリックメニューから適切なサブグラフタイプ（ブロックまたはオペレーター）を選ぶことでも、サブグラフを作成できます。例えば、一連のオペレーターを変換する場合は、**Convert To Subgraph Operator** を選択します。アセットをディスクに保存すると、選択したノードが単一のサブグラフノードに置き換えられます。



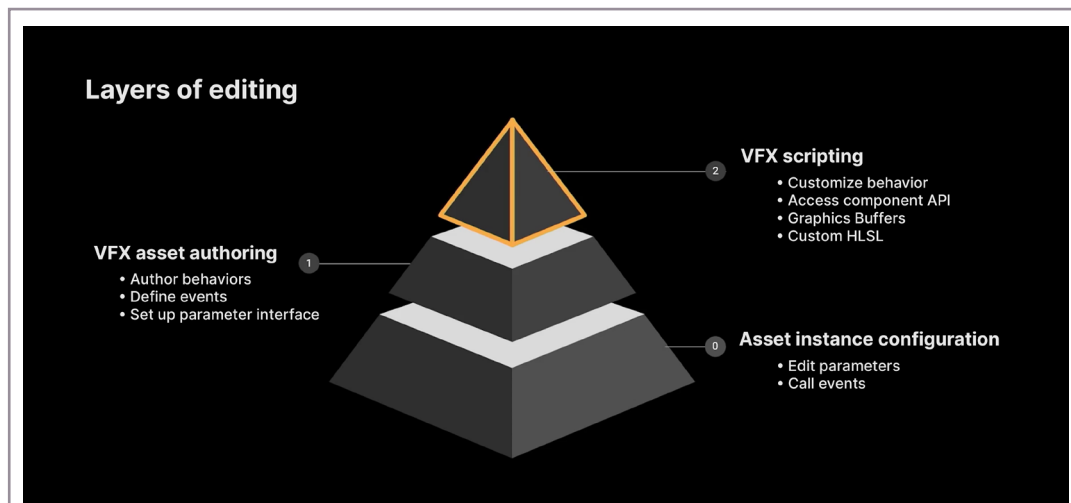
サブグラフの作成方法

サブグラフの Input プロパティを作成するには、Blackboard に新しいプロパティを追加し、Exposed フラグを有効にします。

サブグラフの Output プロパティを作成するには、新しいプロパティを追加し、Blackboard の Output Category に移動させます。

Blackboard では、サブグラフブロックが表示されるメニューのカテゴリを定義することもできます。これを、サブグラフをソートしたり検索したりするために使用します。

サブグラフの作成は、コードのリファクタリングに似ています。ロジックを再利用可能なメソッドや関数にまとめるように、サブグラフは VFX Graph の要素をモジュール化します。



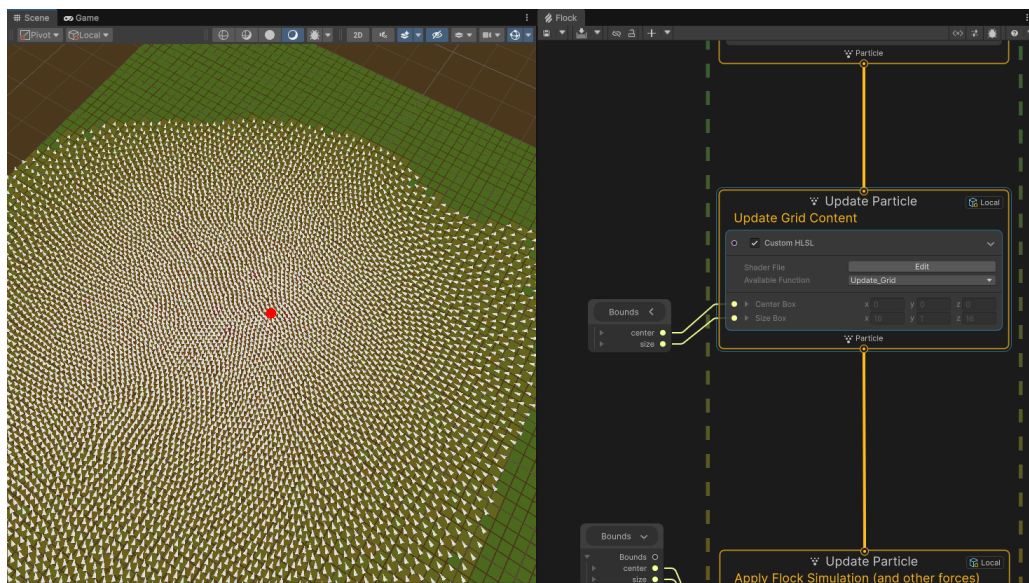
VFX Graph での編集のレイヤー

VFX Graph は、3 つの異なるレベルの編集をサポートしています。

- **アセットインスタンス構成** : 既存の VFX Graph を変更します。デザイナーもプログラマーも、Inspector で公開パラメーターを調整して、エフェクトの外観、タイミング、設定を調整できます。アーティストは、外部のスクリプティングやイベントを使用して、作成済みコンテンツを変更することもできます。このレベルでは、各グラフをブラックボックスとして扱います。

- **VFX アセットのオーサリング**: 創造力が真価を発揮するところです。オペレーターノードのネットワークを構築して独自の VFX Graph 作成を開始し、カスタムの動作とパラメーターを設定してカスタムシミュレーションを作成します。既存のサンプルをベースにする場合でも、ゼロから始める場合でも、特定のエフェクトを自分のものにすることができます。
- **VFX スクリプティング**: 経験豊富なテクニカルアーティストやグラフィックスプログラマーが **コンポーネント API** を使用して VFX Graph の動作をカスタマイズできます。VFX スクリプティングにより、チームは特定のエフェクトを管理する、より効率的なパイプラインを活用し、グラフィックスバッファなどの高度な機能にアクセスできます。

Unity 6 の **カスタム HLSL** を使用すると、標準の VFX Graph ノードでは簡単には実現できない、複雑あるいはユニークなパーティクル動作を実装できます。例えば、カスタム物理演算シミュレーション、パーティクルインタラクション、フロッキング (群れ行動) を作成できます。



カスタム HLSL は、カスタムの動作やインタラクションを作成できる。

経験レベルに関係なく、VFX Graph でエフェクトの制作を開始できます。既製のエフェクトから始めてワークフローに慣れ親しみ、徐々に独自のグラフを組み立てていきましょう。

属性

属性 (Attribute) は、システム内で使用する可能性のあるデータの一部で、パーティクルの色、位置、サイズ、スポーンする数といったものです。シミュレーション中に属性を読み取ったり、変更したりすることで、動的なエフェクトを作成できます。

属性の型は、float (単精度浮動小数点)、Vector2、Vector3、Vector4 (2D、3D、4D ベクトル)、bool (真 / 偽)、int / uint (整数 / 符号なし整数) です。

VFX Graph で頻繁に使用される属性をいくつか紹介します。

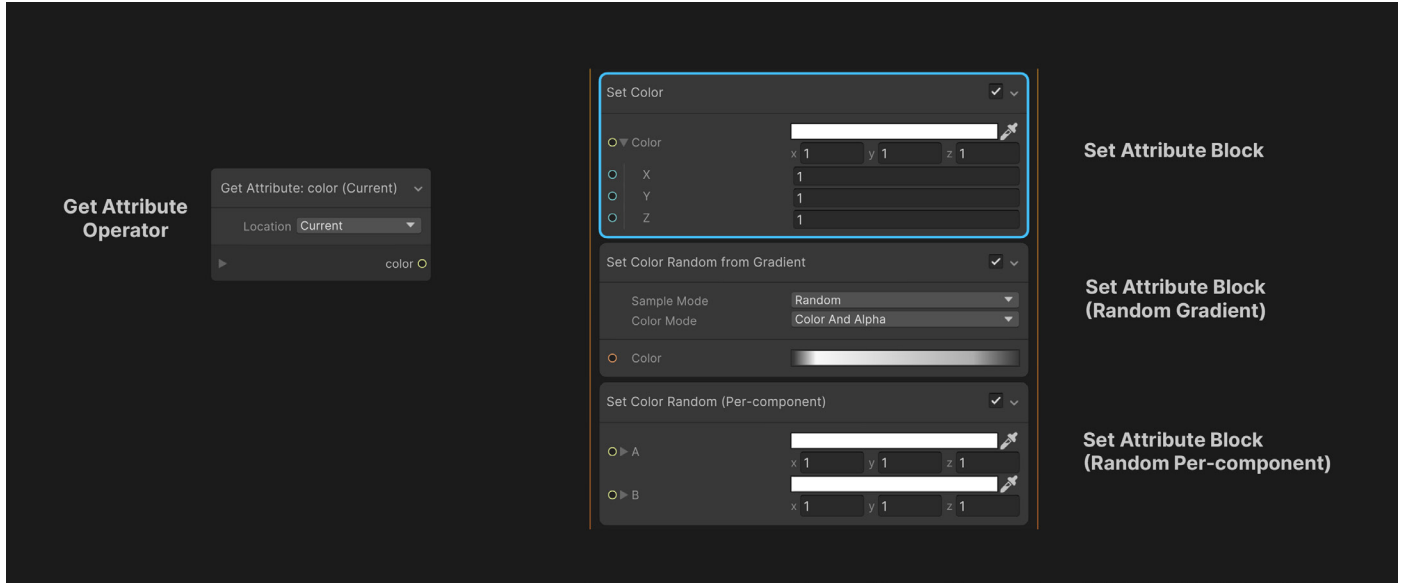
属性	Type	説明
position	Vector3	システム空間 内でのパーティクルの位置を定義
Velocity	Vector3	パーティクルの移動速度と方向を決定
age	浮動小数点	シミュレートされた要素のスポンからの経過時間（秒単位）
lifetime	浮動小数点	シミュレートされた要素の生存期間（秒単位）
color	Vector3	パーティクルの RGB カラーの設定
alpha	浮動小数点	パーティクルの透明度の調整
size	浮動小数点	パーティクルの均一なサイズを制御
scale	Vector3	パーティクルに不均一なスケーリングを適用
mass	浮動小数点	重力などの力に対する反応に影響するパーティクルの質量
direction	Vector3	任意の方向を格納、またはパーティクルの動きのガイドに使用可能
angularVelocity	Vector3	パーティクルの回転速度（度 / 秒）
targetPosition	Vector3	パーティクルが移動する目標位置。Line Renderer（ラインレンダラー）などの特定のエフェクトで便利
texIndex	浮動小数点	レンダリングされた要素の Flipbook UV のサンプリングに使用されるアニメーションフレーム

属性は、VFX Graph パーティクルの基本的な性質を管理するうえで非常に重要です。標準属性の完全なリストについては、[ドキュメントページ](#)を参照してください。

Blackboard の Attributes タブ（[後述](#)）を使用して、多くのビルトイン属性を確認したり、カスタム属性を定義したりすることもできます。

属性の読み出しや書き込みにはノードを使用します。具体的には、以下を使用します。

- **Particle** または **ParticleStrip System** 内の属性から値を読み取るための **Get Attribute オペレーター**
- **Spawn システム**の属性から値を読み取るための実験的な [SpawnerCallback](#)
- 属性に値を書き込む **Set Attribute ブロック**。属性に値を直接設定するか、ランダムモードを使用（**Random Gradient** または **Random Per-Component ブロック**で**色属性**を設定するなど）



オペレーターで属性を取得し、ブロックで属性を設定する。

ほとんどの属性はパーティクルごとに保存されるため、パーティクルや属性の数が増えるにつれてメモリフットプリントが増加する可能性があります。例えば、パーティクルが 10,000 個ある場合、各パーティクルに位置、速度、色、サイズなどの複数の属性を保存すると、このデータ維持に必要なメモリが膨大になります。

コンテキストを選択して、Inspector に表示される System Attribute Summary や Current Attribute Layout を使用して、属性を監視し、最適化します。

- **System Attribute Summary:** このセクションは、現在のシステムで使用されているシステムレベルのすべての属性の概要を示します。
- **Current Attribute Layout:** 選択したコンテキストで使用されている属性が表示されます。
- **Source Attribute Layout:** これは、ソースコンテキスト（初期データまたは入力を提供するコンテキスト）で使用される属性を示します。例えば、Initialize コンテキストで初期化され、その後 Update コンテキストで使用される属性です。

例えば、**Trigger Event on Collide** テンプレートで Dart_Spawn System 内のコンテキストを選択すると、Inspector には次のように表示されます。

Name	Value		
Capacity	32		
System Attribute Summary			
alive	Boolean	4 bytes	ReadWrite
color	Float3	12 bytes	ReadWrite,
angleZ	Float	4 bytes	ReadWrite,
direction	Float3	12 bytes	ReadWrite
lifetime	Float	4 bytes	ReadWrite
seed	UInt32	4 bytes	ReadWrite
position	Float3	12 bytes	ReadWrite
velocity	Float3	12 bytes	ReadSource
collisionEventPosition	Float3	12 bytes	ReadSource
particleId	UInt32	4 bytes	Read
pivotX	Float	4 bytes	Read
pivotY	Float	4 bytes	Read
pivotZ	Float	4 bytes	Read
size	Float	4 bytes	ReadWrite
scaleX	Float	4 bytes	Read
scaleY	Float	4 bytes	Read
scaleZ	Float	4 bytes	Read
axisX	Float3	12 bytes	ReadWrite
axisY	Float3	12 bytes	ReadWrite
axisZ	Float3	12 bytes	ReadWrite
angleX	Float	4 bytes	ReadWrite
angleY	Float	4 bytes	ReadWrite
age	Float	4 bytes	ReadWrite
Current Attribute Layout			
0	alive	age	
1	color	color	color
2	direction	direction	direction
3	lifetime	angleZ	position
		position	position
		position	particle
Source Attribute Layout			
0	position	position	position
1	angleZ	alive	velocity
2	lifetime	velocity	velocity
3	color	color	color
4	collisior	age	
5	collisior	collisior	collisior

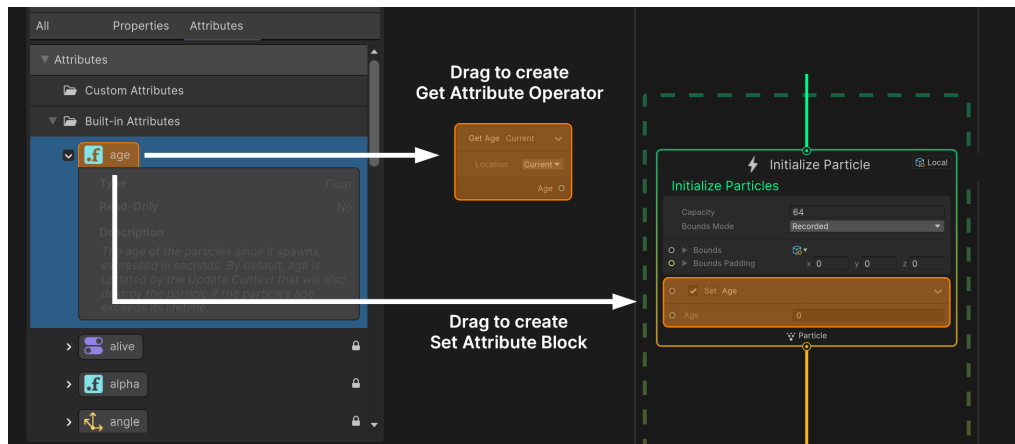
Inspector を使用して、コンテキストでの属性の使用状況を確認する。

メモリの使用を最適化するために、システムは実際に必要とされる属性のみを格納します。属性のシミュレーションデータが格納されていない場合、VFX Graph は不要なデータを格納する代わりに、デフォルトの定数値を使用します。VFX Graph のプロファイリングツールの詳細については、[最適化](#)セクションを参照してください。

Blackboard の属性

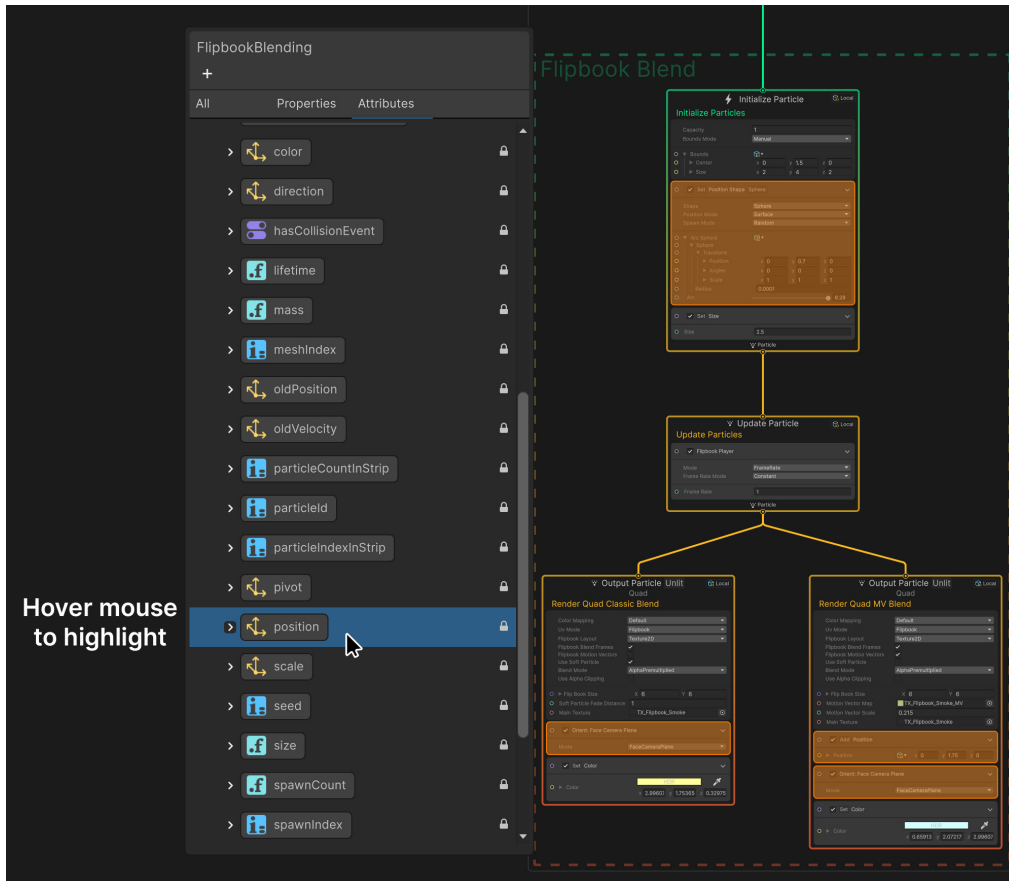
Blackboard パネルに属性専用のセクションが追加され、関連するオペレーターやブロックを、コンテキストに応じてドラッグアンドドロップで簡単に作成できるようになりました。これらの新機能は Unity 6 で利用できます。

- Blackboard からノードワークスペースにドラッグアンドドロップして、Get Attribute オペレーターを作成する。
- Blackboard からシステムのコンテキストにドラッグアンドドロップして、[Set Attribute ブロック](#)を作成する。



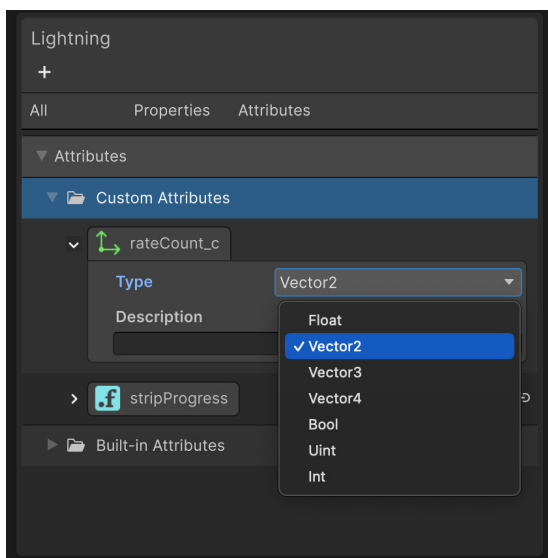
Blackboard から属性をドラッグする。

- グラフ要素にマウスカーソルを合わせると、属性の使用状況がハイライトされる。



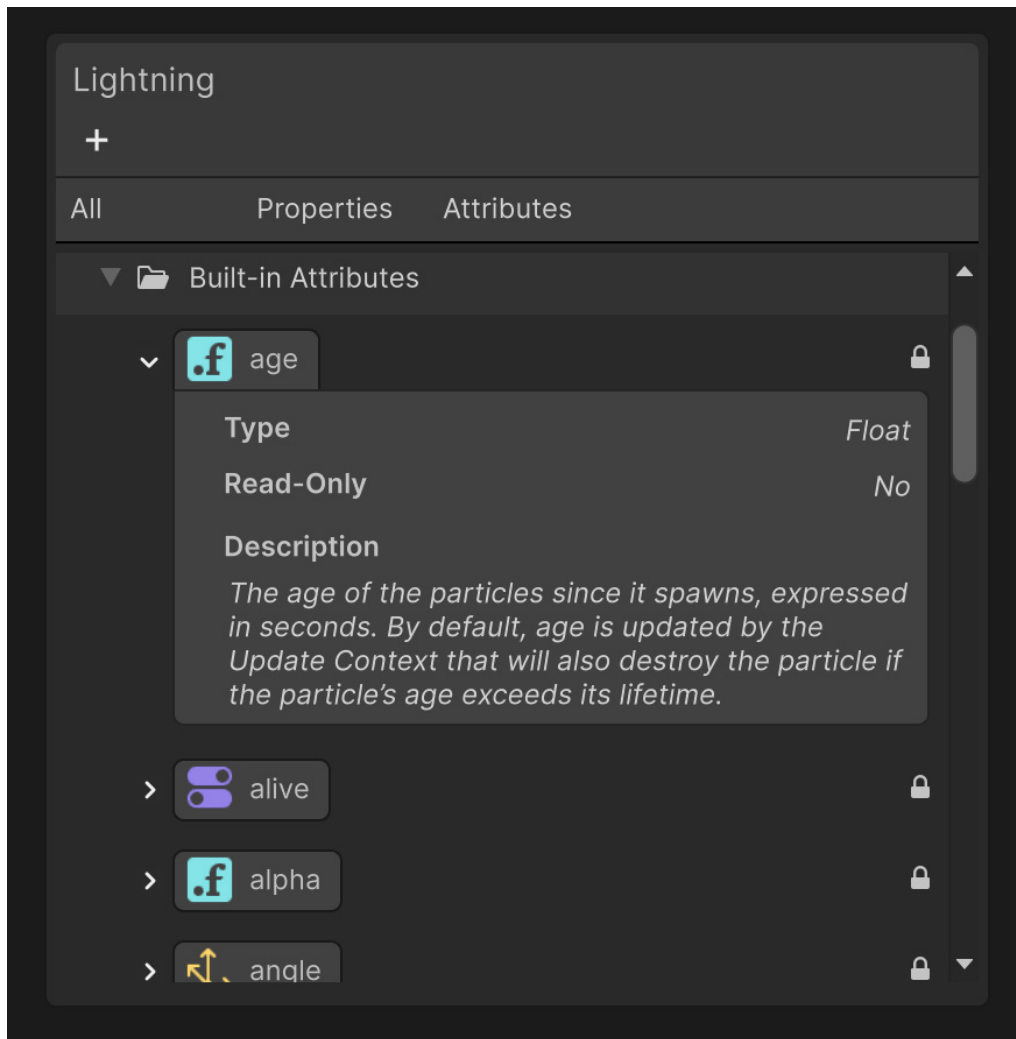
Blackboard にマウスカーソルを合わせると、属性の使用状況がハイライトされる。

- Blackboard からカスタムの Attribute オペレーターを作成し、型を変更する。



カスタム属性の型を選択する。

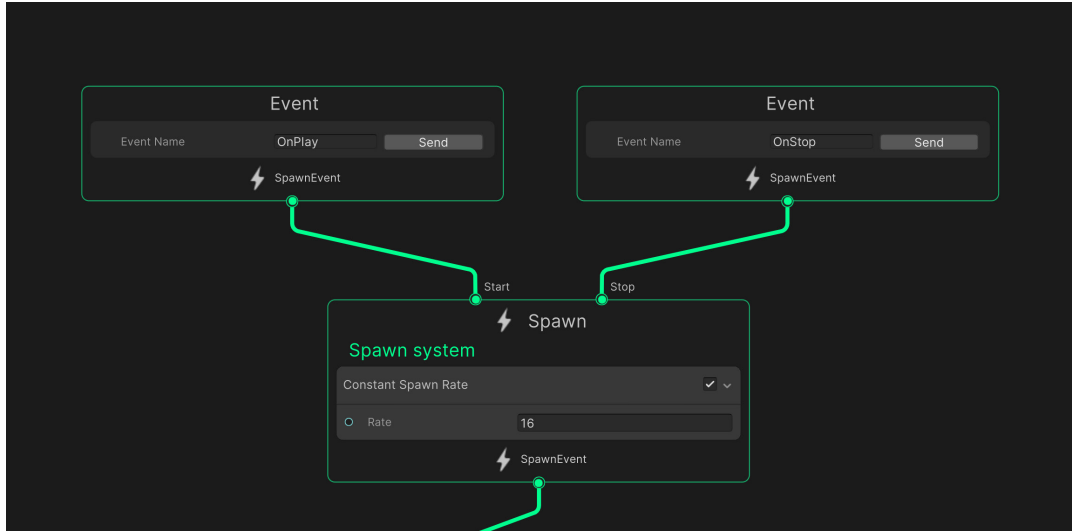
— ビルトイン属性の説明と型にアクセスする。



ビルトイン属性を調べる。

イベント

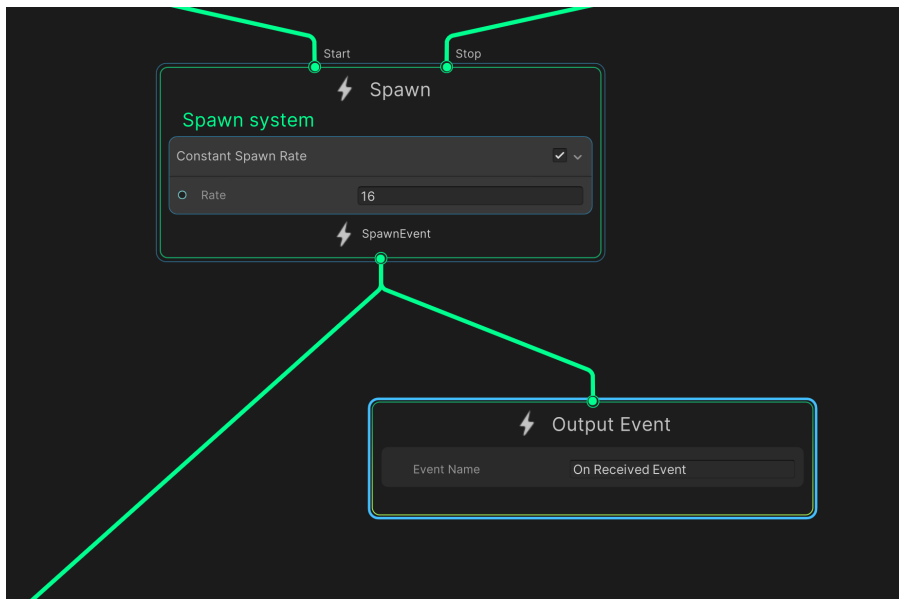
VFX Graph のさまざまなパーツは、**イベント**を通じて互いに（およびシーンの残りの部分と）通信します。例えば、各 **Spawn コンテキスト**にはフローの **Start** ポートと **Stop** ポートがあり、パーティクルのスポーンを制御するイベントを受け取ります。



イベントはパーティクルのスポーンを制御する。

何かを実行する必要があるときは、外部のゲームオブジェクトが **C# API** の `SendEvent` メソッドを使用してグラフの一部を通知できます。その後、Visual Effect コンポーネントはイベントを文字列名またはプロパティ ID として渡します。

イベントコンテキストは、**グラフ内の Event 文字列名または ID** でイベントを識別します。上記の例では、シーン内の外部オブジェクトが **OnPlay イベント**を発生させて Spawn システムを起動したり、**OnStop イベント**で停止させたりできます。

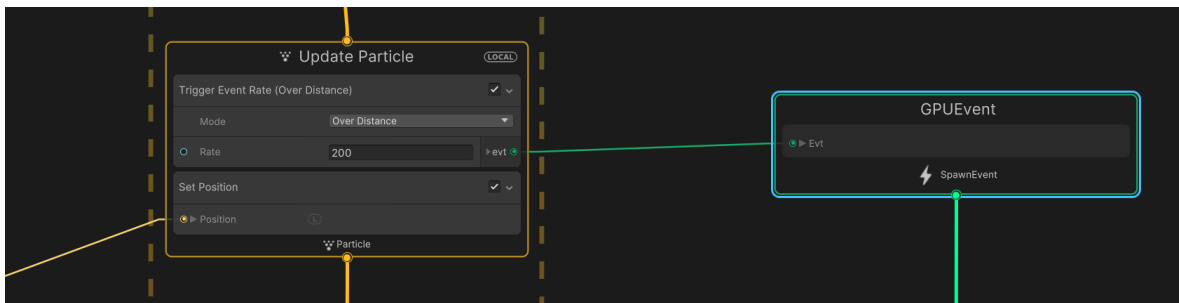


Output イベントは、シーンにメッセージを送信できる。

Output イベントは、**Output イベントハンドラー**と組み合わせることができます。Output イベントは、パーティクルの最初のスポーンでシーン内の他の要素を動かす必要がある場合に便利です。これは、ライティングやゲームプレイをビジュアルエフェクトと同期させるためによく使用されます。

上記の例では、グラフ外のゲームオブジェクトコンポーネントに **OnReceivedEvent** を送信しています。その後、C# スクリプトが反応し、光や炎を強めたり、火花を飛ばしたりします。Output イベントの詳細については、[インタラクティブ性のセクション](#)を参照してください。

同時に、GPU イベントを使用して、他のパーティクルの動作に基づいてパーティクルをスポーンできます。このことにより、あるシステムでパーティクルが消滅したときに別のシステムに通知し、エフェクトの便利な連鎖反応（死亡時に塵のエフェクトをスポーンする投射パーティクルなど）を生み出すことができます。



GPU イベントのコンテキストは、Trigger Event Rate ブロックからイベントを受け取る。

これらの **Update ブロック**は、以下の方法で **GPU イベント**データを送信できます。

- **Trigger Event On Die:** パーティクルが消滅したときに別のシステムでパーティクルをスポーン
- **Trigger Event Rate:** 毎秒スポーンするパーティクル（またはその速度に基づく）
- **Trigger Event Always:** パーティクルをフレームごとにスポーン

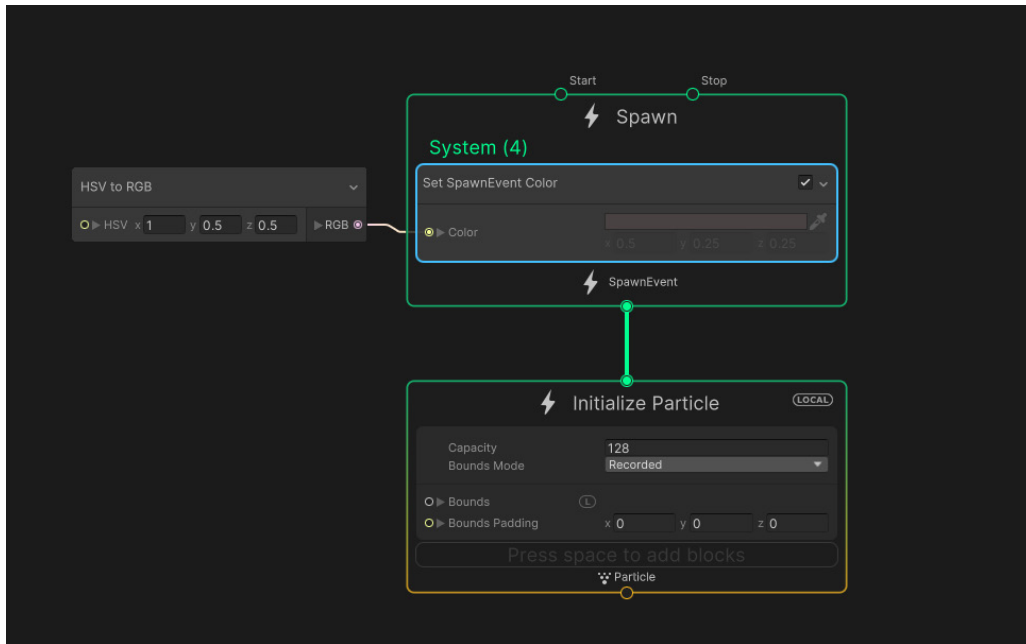
ブロックの出力は **GPU イベントコンテキスト**に接続され、その後、依存するシステムの **Initialize コンテキスト**に通知できます。異なるシステムをこの方法でつなぐことで、緻密で複雑なパーティクルエフェクトを作成できます。

GPU イベントシステムの Initialize コンテキストは、Trigger イベントに先立つ親システムで利用可能な属性も継承できます。そのため、例えばその位置を継承することで、新しいパーティクルはスポーンした元のパーティクルと同じ場所に現れます。

イベント属性

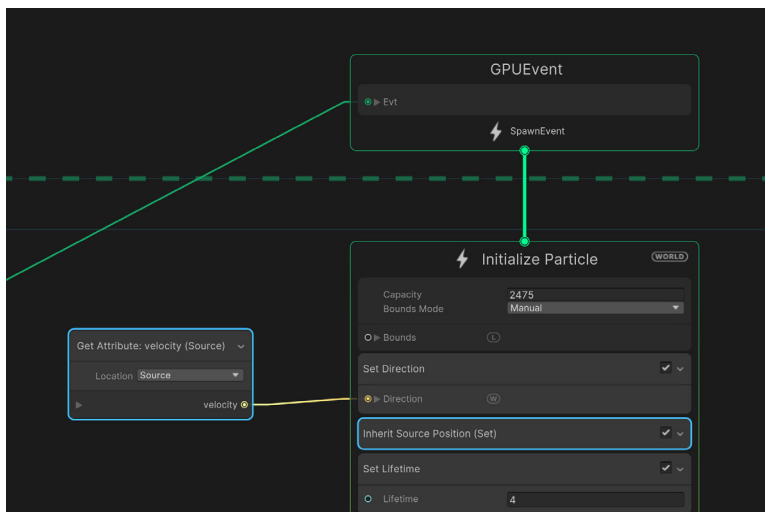
イベント属性ペイロードを使用して、3D の位置や色などのデータをイベントと一緒に渡します。これらのペイロードは、グラフ内を暗黙に伝わる属性を持ち、オペレーターまたはブロックでデータを“キャッチ”できます。

Spawn イベントや **Timeline イベント**で渡される属性を読み取ることもできます。**Set SpawnEvent Attribute ブロック**は、Spawn コンテキストのイベント属性を変更します。



Spawn コンテキストでのイベント属性の読み取り

Initialize コンテキストでペイロードをキャッチするには、**Get Source Attribute オペレーター** または **Inherit Attribute ブロック**を使用します。

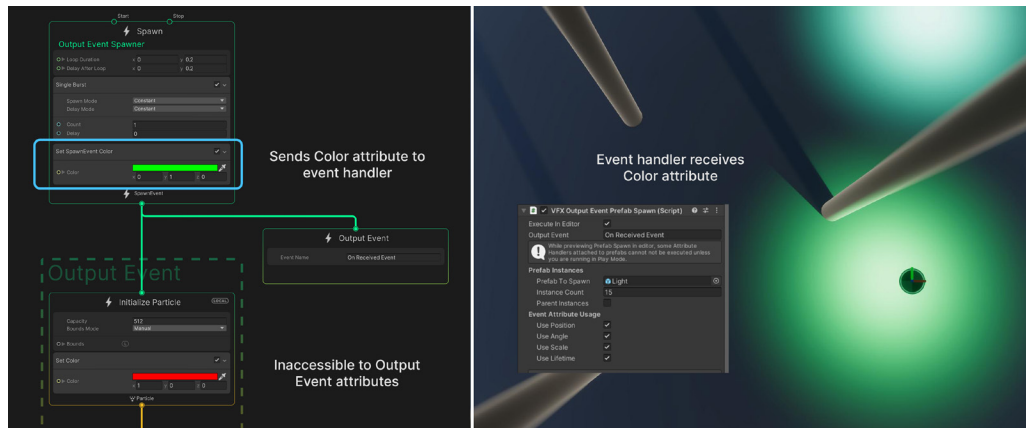


Initialize でのイベントペイロードのキャッチ

ただし、イベント属性を使用する際は、次の点に注意することが重要です。

- **Regular Event Attributes** は、Initialize コンテキストでのみ読み取ることができます。Update や Output では継承できません。属性を後のコンテキストで使用するには、それを Initialize で継承して設定する必要があります。
- **Output Event Attributes** は、Spawn コンテキストで設定された初期値のみを保持します。その後にグラフで発生する変更はキャッチされません。

詳細については、Visual Effect コンポーネント API の [Sending Events](#) を参照してください。



Output イベント属性は Spawn コンテキストからの値を保持する

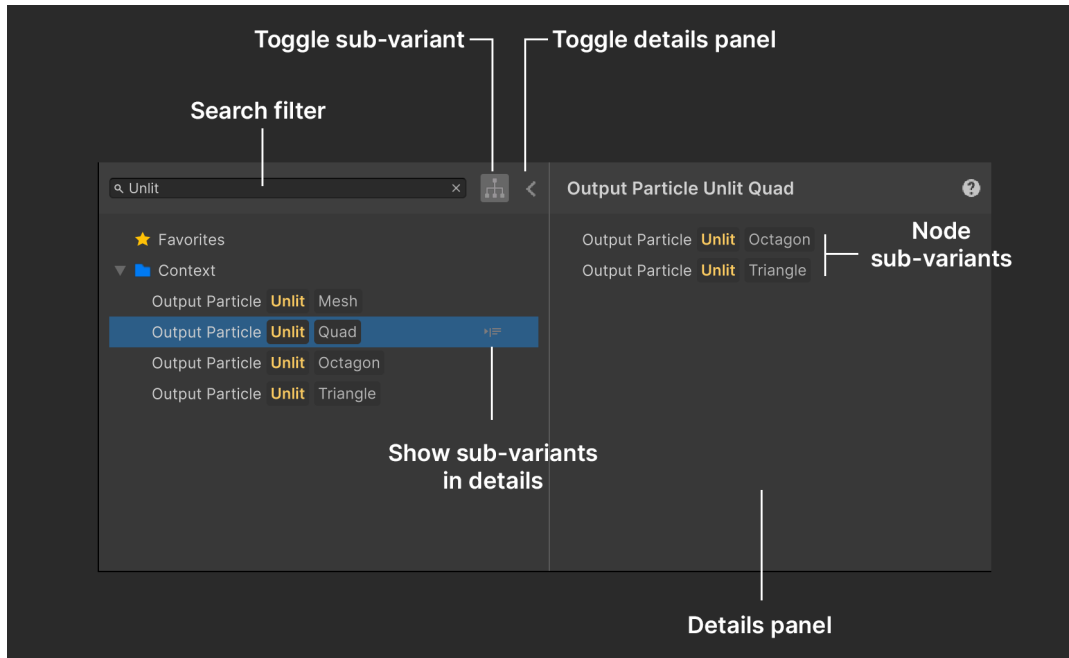
Unity 6 での UI 改善

Unity 6 では、いくつかのクオリティオブライフ (QOL) の改善や、VFX Graph UI のアップデートが行われています。

ノード検索

ノードまたはブロックの作成に階層ツリービューが使用され、ノードライブラリの参照が簡単になりました。改良点として、より効率的でパーソナライズされた検索体験を実現するカスタムカラーとお気に入りフォルダーがあります。高度な検索フィルタリングを使用して、利用可能なノードから選択します。

新しいサイド詳細パネルには、ノードのサブバリエーションも表示されます (例えば、Output Particle Unlit Octagon や Output Particle Unlit Triangle は Output Particle Unlit Quad のサブバリエーションです)。サブバリエーションの表示を切り替えるボタンがあります。無効にすると、最も一般的なノードのみが表示されます。有効にすると、利用可能なすべてのバリエーションにアクセスできます。



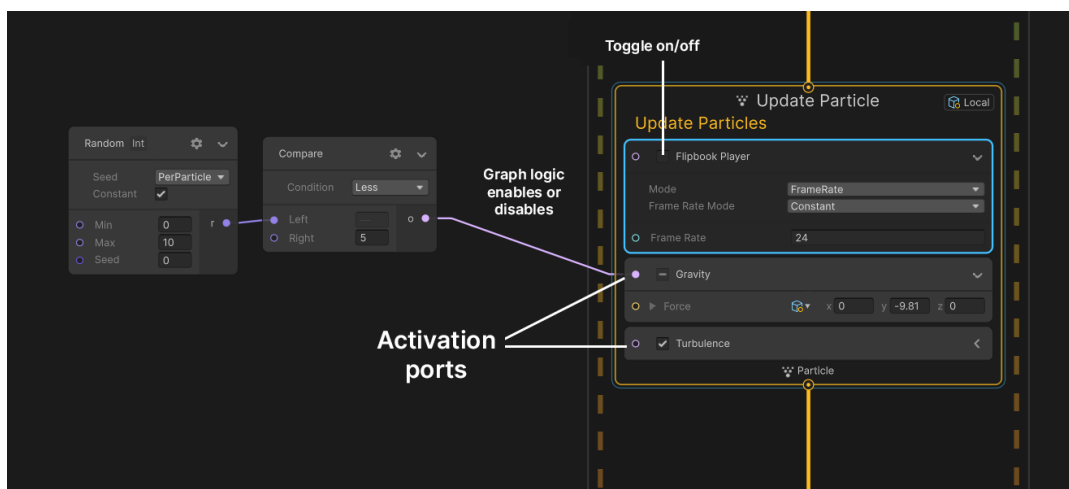
ノードまたはブロックを作成すると、サイドパネルと検索フィルタリングが追加される。

アクティベーションポート

ブロックの左上、名前の横に特別な**アクティベーションポート**があり、ブーリアンのプロパティにリンクされています。このポートでは、ブロックをアクティブにするかどうかを制御できます。

ブロックのオンとオフを手動で切り替えることも、グラフロジックをポートに接続してブロックをアクティブにするタイミングを制御することもできます。これにより、同一システム内でパーティクルごとに異なる動作やスタートを実装できます。

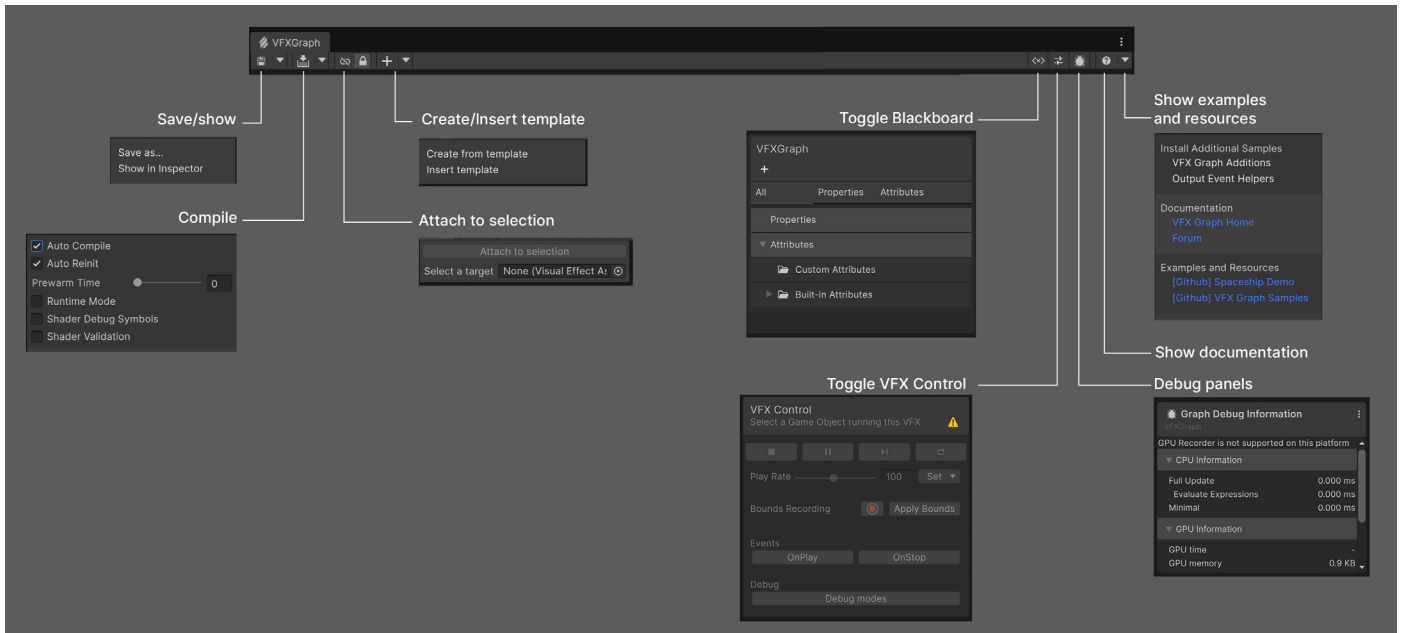
静的に非アクティブなブロックはコンパイル中にグレイアウトされて自動的に削除されるため、ランタイムコストはゼロです。



ブロックごとにアクティベーションポートがある。

新しい VFX Toolbar

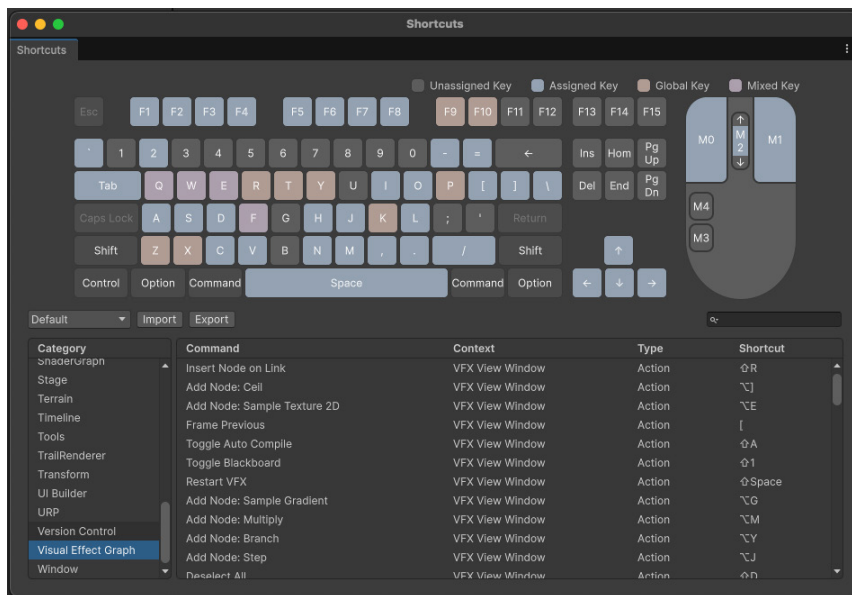
VFX Toolbar が簡素化され、ドキュメントやサンプルにすばやくアクセスするための新しいオプションが追加されました。



VFX Toolbar が簡略化された。

キーボードショートカット

ショートカットマネージャーに VFX Graph カテゴリが追加され、VFX Graph ウィンドウで利用可能なショートカットコマンドを編集できるようになりました。VFX アーティストのワークフローをスピードアップするために、新しいショートカットコマンドが追加されました。

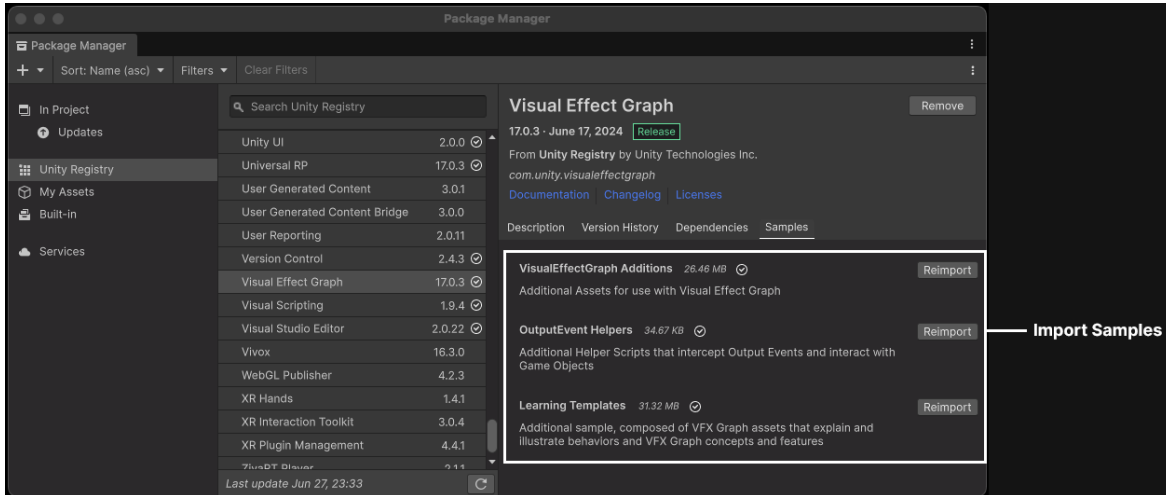


ショートカットマネージャーに VFX Graph カテゴリが追加された。

VFX サンプルコンテンツを詳しく見る

VFX Graph は、単にパーツを足し合わせただけのものではありません。ノードとオペレーターをそれぞれ適用する方法と、それらを連携させる方法を確実に理解する必要があります。

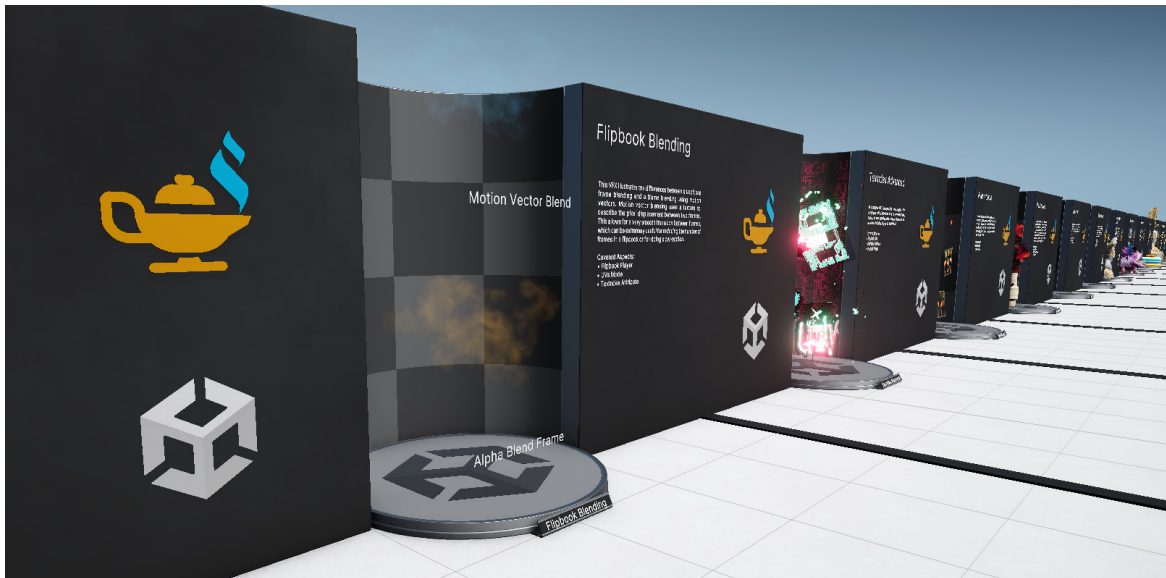
Package Manager で入手できる 2 つのサンプルが、これらの機能をコンテキスト内で示すのに役立ちます。
[VFX Graph Learning Templates](#) と [VFX Graph Additions](#) です。



Package Manager からサンプルコンテンツをインストールする。

[VFX Graph Learning Templates](#) には、さまざまなテクニックが用意されています。この教育用サンプルのコレクションは、VFX Graph の特定の側面や機能セットを詳しく知るのに役立ちます。

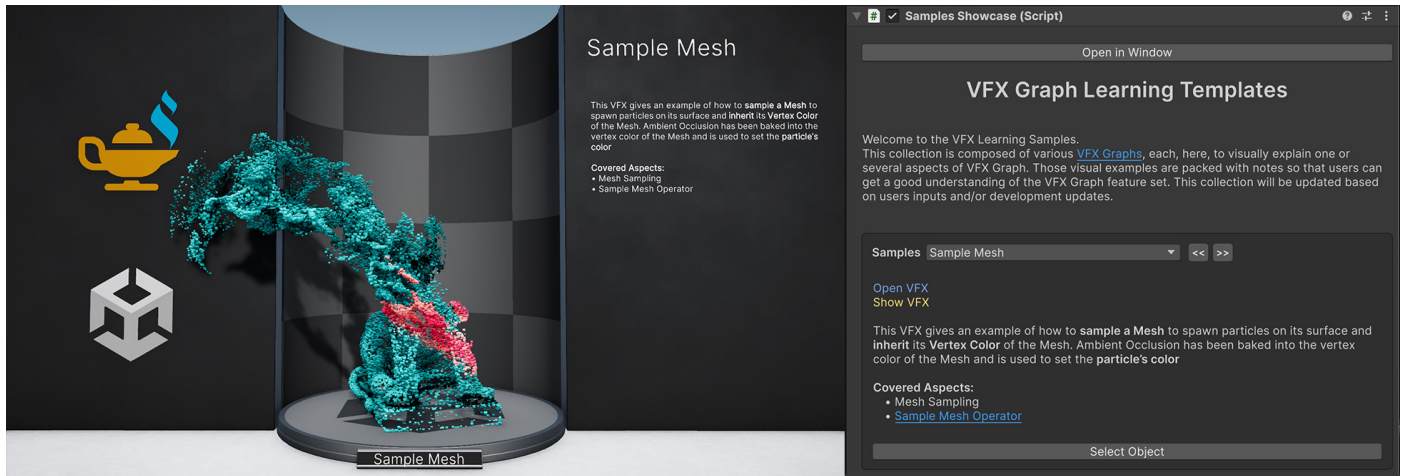
サンプルコンテンツは、VFX Graph バージョン 17.0 (Unity 6) 以降の URP プロジェクトと HDRP プロジェクトの両方に対応しています。



VFX Graph の Learning Templates の使用を開始する。

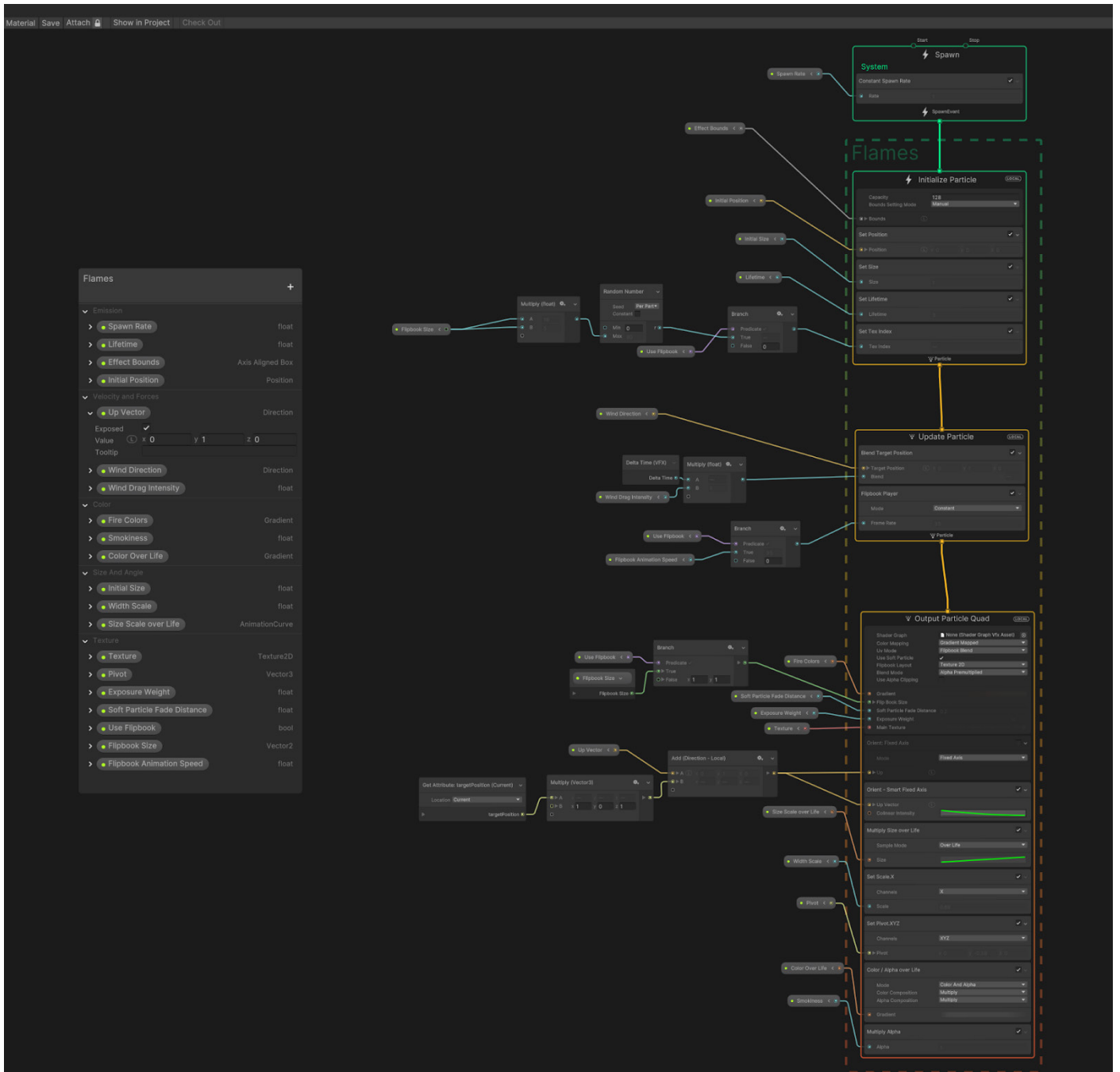
Scene ビューで自由に動き回ったり、Game ビューで各エフェクトにフォーカスを当てたりできます。Inspector の **Sample Showcase ウィンドウ**には、対応する情報が表示されます。ドキュメントへのクイックアクセスリンクや、エフェクト間を移動するリンクも表示されます。各 VFX アセットには、ガイドとなるメモと説明が埋め込まれています。

これらのサンプルは、[ビジュアルエフェクトの例](#)で詳しく説明します。

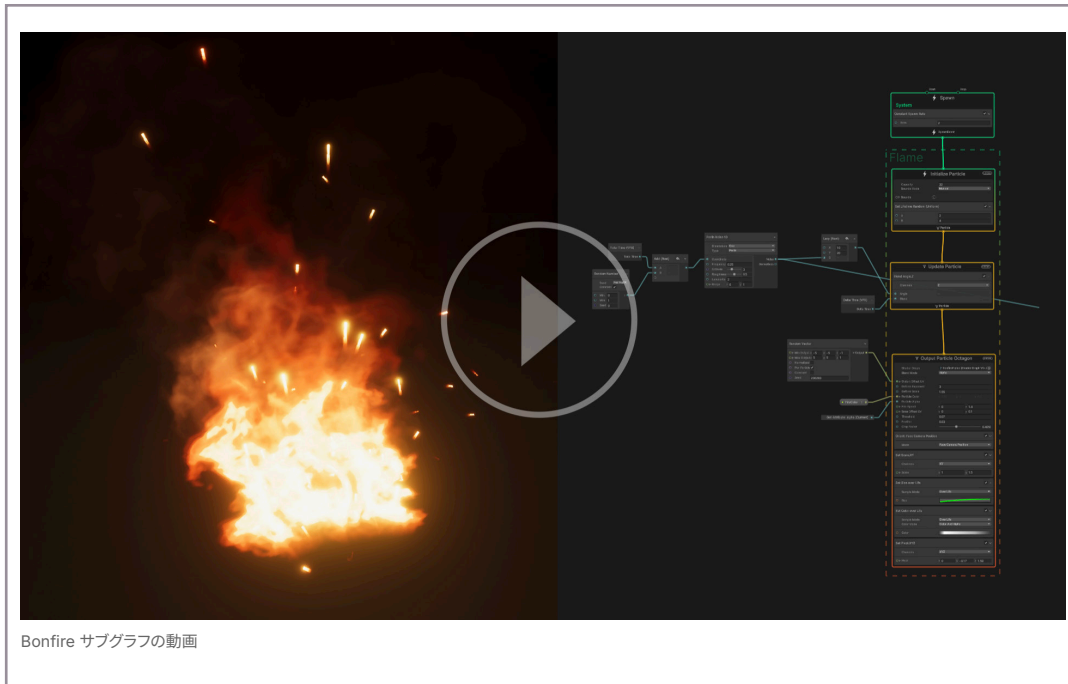


Sample Showcase ウィンドウを使用して各 VFX Graph をナビゲート。

一方、Package Manager の [VFX Graph Additions](#) にはいくつかのシンプルなグラフが用意されており、パーティクルの管理方法を学ぶ開始点になります。以下の例では、煙、炎、火花がどのように立ち上って Bonfire エフェクトを形成するかを確認できます。

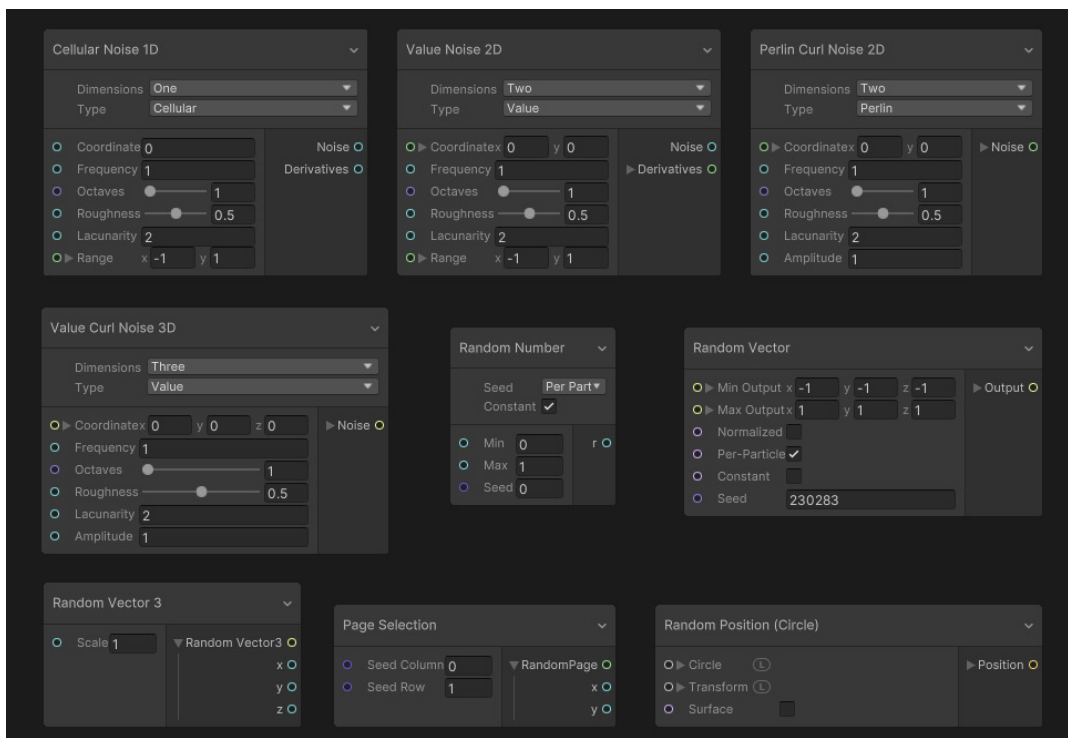


Flames エフェクトは基本的なグラフを示す。

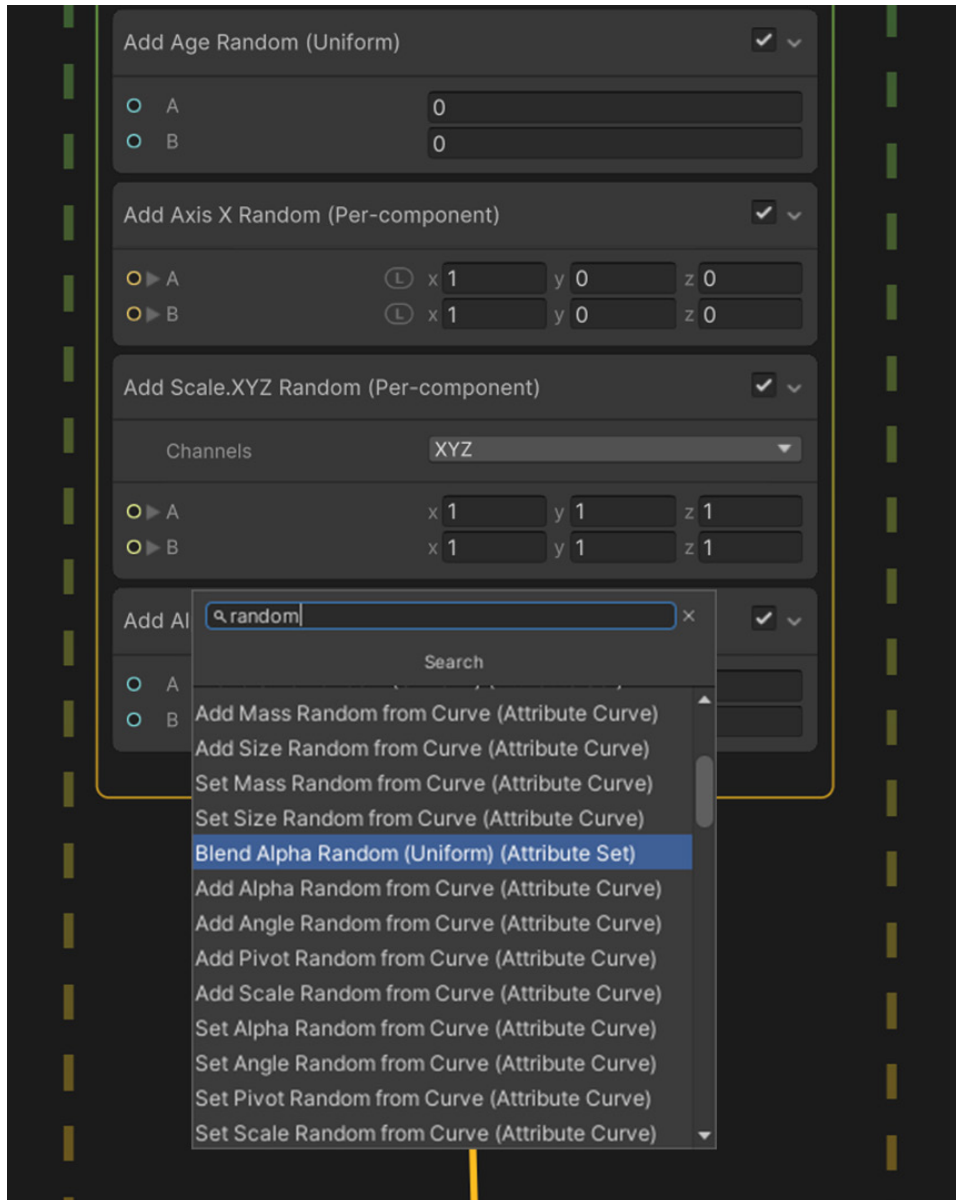


提供されているサンプルを探索すると、一般的なブロックやオペレーターがいくつか見つかります。

- **Noise オペレーターと Random オペレーター** : プロシージャルノイズは、レンダリングされた画像の“機械っぽさ”を減らすのに役立ちます。VFX Graph には、1 次元、2 次元、3 次元の **Noise** と **Randomness** に使用できるオペレーターがいくつか用意されています。



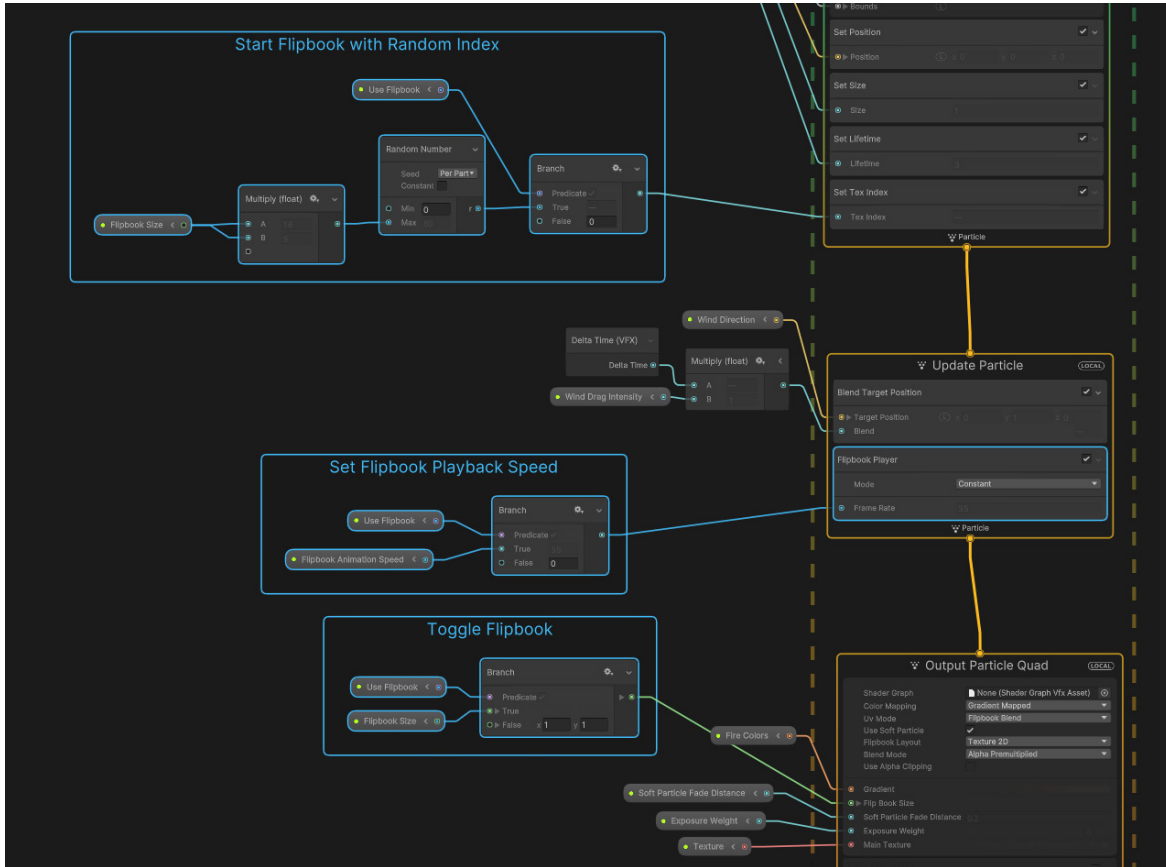
- **Attribute ブロック**: これらにも同様に、さまざまなモードで Randomness を適用するオプションがあります。属性ごとに若干異なる可能性があるため、いろいろと試して動作に慣れておきましょう。



Randomness ブロック

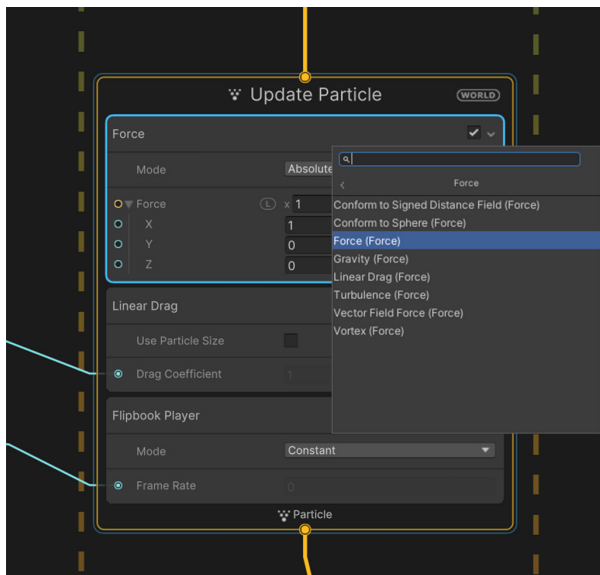
- **Flipbook (フリップブック)**: 煙や炎のサンプルを見ればわかるように、アニメーション化されたテクスチャはエフェクトを本物らしく仕上げるのに驚くほど効果的です。こういったものを、外部の**デジタルコンテンツ制作 (DCC)** ツールまたは Unity で生成します。オペレーターを使用して **Flipbook ブロック**を管理します。

Unity で独自のフリップブックを作成する方法の詳細については、[VFXToolbox セクションの Image Sequencer](#) を参照してください。



Flipbook ノード

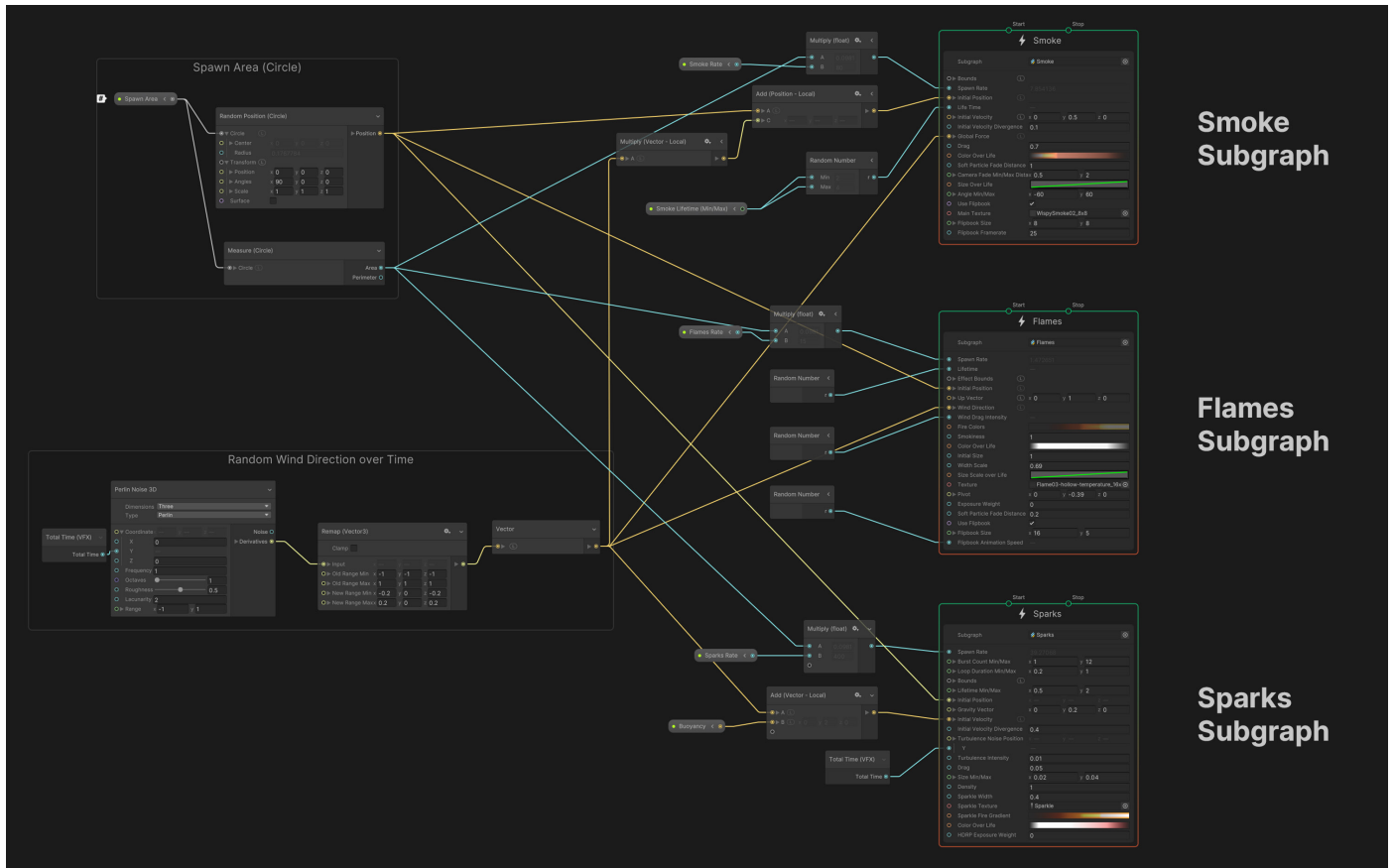
- **Physics** (物理演算): **Force** (力)、**Collision** (衝突)、**Drag** (抗力) は、パーティクルが自然現象をシミュレートするのに不可欠です。現実の限界を押し広げることを恐れることはありません。アーティストとして、何が適切に見えるかを、あなたが決めるのです。



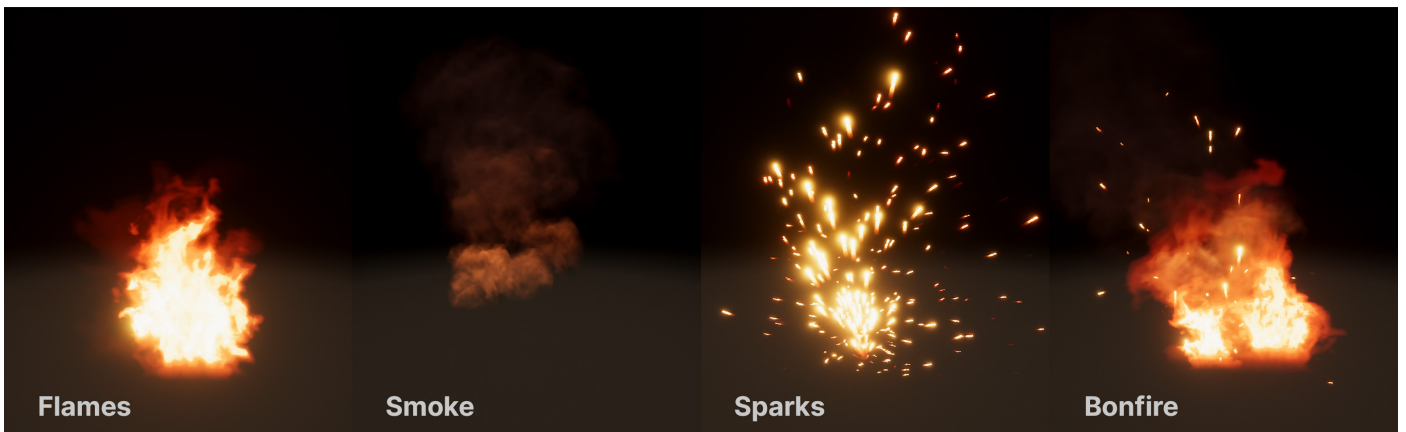
Physics ブロック

- **サブグラフ**: 煙、炎、火花は**サブグラフ**です。これらは、後で再利用するためにアセットとして保存できる VFX Graph の一部です。

メイン要素を小さな部分に分割すると、Bonfire グラフが読みやすくなります。例えばアプリケーションの別の場所で新しい爆発エフェクトを作成する必要がある場合は、それを別のグラフにドラッグアンドドロップして配置できます。これは、サブグラフがアセットであるからです。



Bonfire グラフで 3 つのサブグラフを使用する。



個々のサブグラフ要素が組み合わさって Bonfire を形成する。



Bonfire グラフの構築方法、特にエフェクトの詳細については、サンプルや Unity の Spaceship Demo の作成に携わったシニア VFX およびテクニカルアーティスト Thomas Iché による[これらのコミュニティの動画](#)をご覧ください。

その他のリソース

VFX Graph の基本的な作業に慣れたら、いくつかのエフェクトをゼロから作成してみましょう。[雪が降る](#) シンプルなシステムから始めて、[炎](#)、[煙](#)、[そして霧](#)などを体験しましょう。

以下の動画ではいくつかのエフェクトを紹介していますが、古いバージョンの VFX Graph を使用していることに注意してください。

- [Making snow with the VFX Graph](#) (VFX Graph で雪を作る)
- [Creating fire, smoke, and mist effects with the VFX Graph in Unity](#) (Unity VFX Graph で 炎、煙、霧のエフェクトを作る)
- [Rendering particles with the VFX Graph in Unity](#) (Unity VFX Graph でパーティクルをレンダリングする)
- [Multilayered effects with the VFX Graph in Unity](#) (Unity VFX Graph によるマルチレイヤーエフェクト)
- [Hardspace: Shipbreaker Tech Talk: Explosions with the VFX Graph](#) (VFX Graph による爆発)
- [Real-time VFX workflows in The Heretic](#) (The Heretic におけるリアルタイム VFX ワークフロー)

その他の参考資料

VFX Graph に慣れてくると、さらに掘り下げて、そのニュアンスを発見できます。特定のノードまたはオペレーター機能を参照する必要がある場合は、これらの Web ページをすぐに参照できるようにしておいてください。

- [ノードライブラリ](#)は、VFX Graph 内のすべてのコンテキスト、ブロック、オペレーターを説明しています。
- [標準属性リファレンス](#)には、一般的なすべての属性の包括的なリストが用意されています。
- [VFX の型のリファレンス](#)には、VFX Graph で使用されるデータ型の一覧があります。

ビジュアルエフェクトの例

VFX Graph の基礎を理解したら、もう少し複雑なグラフの作成に挑戦しましょう。制作中に遭遇する可能性のある問題への備えとして、サンプルプロジェクトが多数用意されています。

これらのサンプルは、ビジュアルで可能なことのすべてを網羅しています。アンビエントスモークや炎から、完全にスクリプト化された AAA 映画級のゲームプレイまで、じっくり探索しましょう。

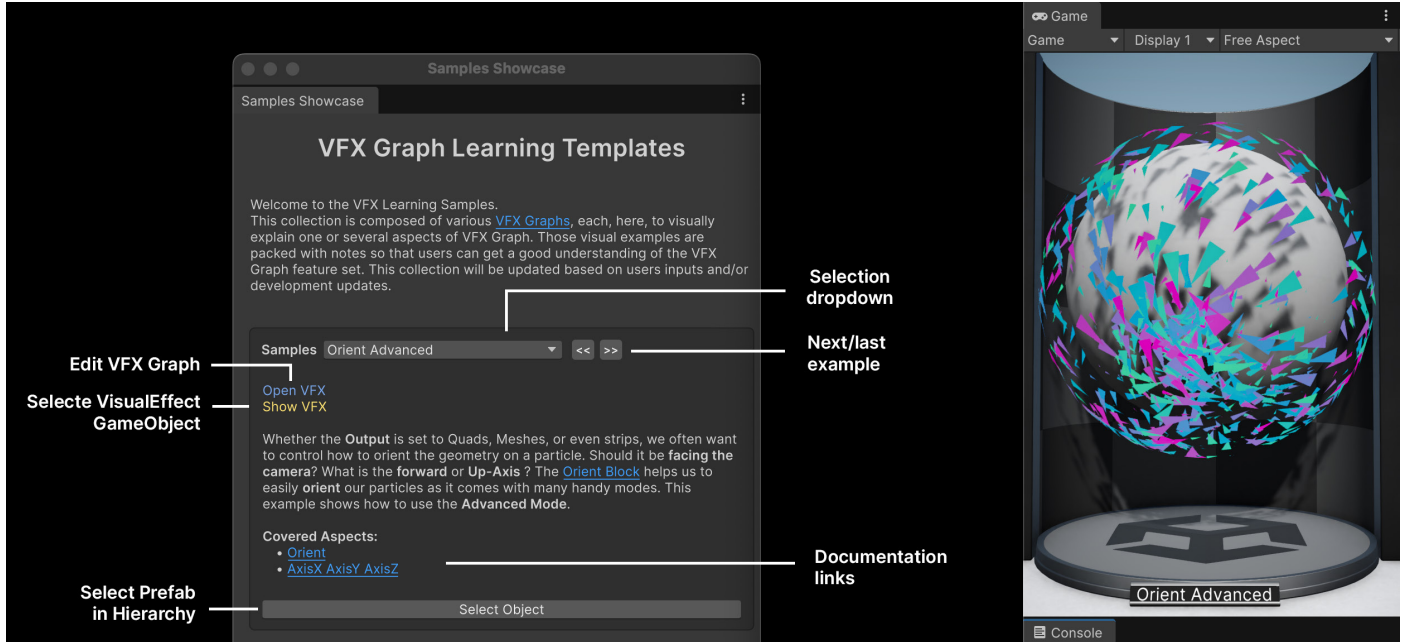
VFX Graph の Learning Templates

VFX Graph を初めて使用する場合は、こちらの VFX Graph のコレクションから始めるのが最適です。Learning Templates にはさまざまなリアルタイムエフェクトが用意されており、VFX Graph のさまざまな側面を学習できます。

焦点を絞った小さなグラフですので、学習用サンプルとして最適です。各テンプレートを掘り下げて新しい手法をマスターしたり、独自のエフェクトの出発点として利用したりしましょう。各グラフには、構造を理解するのに役立つ詳しい説明があります。

Learning Templates は、新しい VFX Graph の作成時にウィザードから利用できます。あるいは、Package Manager から URP または HDRP 用の完全なサンプルシーンとしてインポートすることもできます。

VFX Samples Showcase ウィンドウでサンプルを見て回ります。Learning Templates にさっそく行ってみましょう。

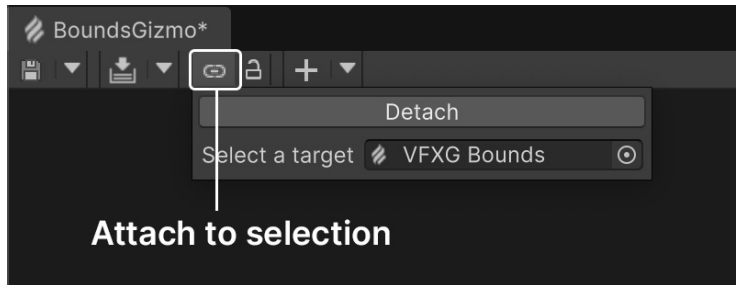


Samples Showcase ウィンドウ

ギズモの視覚化

多くのブロックには、Scene ビューのビジュアルギズモに対応する調整可能なパラメーターがあります。ブロック内で直接値を変更したり、Scene ビューでギズモを操作したりできます。

ギズモを操作するには、編集中の VFX Graph を、シーン内の対応する VFX インスタンスに“アタッチ”する必要があります。

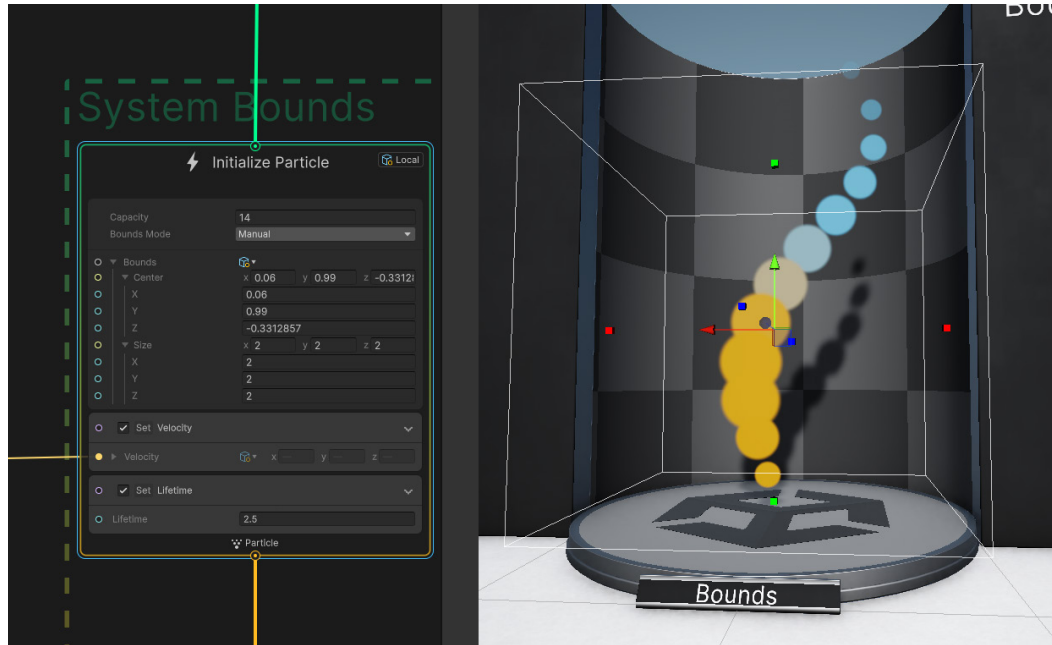


VFX Graph を、VisualEffect コンポーネントを持つゲームオブジェクトにリンクする。

ウィンドウの上部にある小さな“リンク”アイコンを使用して、VFX Graph を、Visual Effect コンポーネントを持つ選択したゲームオブジェクトにアタッチします。

アタッチすると、VFX Graph エディターで対応するブロックを選択することにより、同期して表示されます。ギズモのハンドルを使用して設定をインタラクティブに変更できます。

例えばこれは、Initialize Particle コンテキストのバウンド設定と、それに対応する Scene ビューのギズモを示しています。



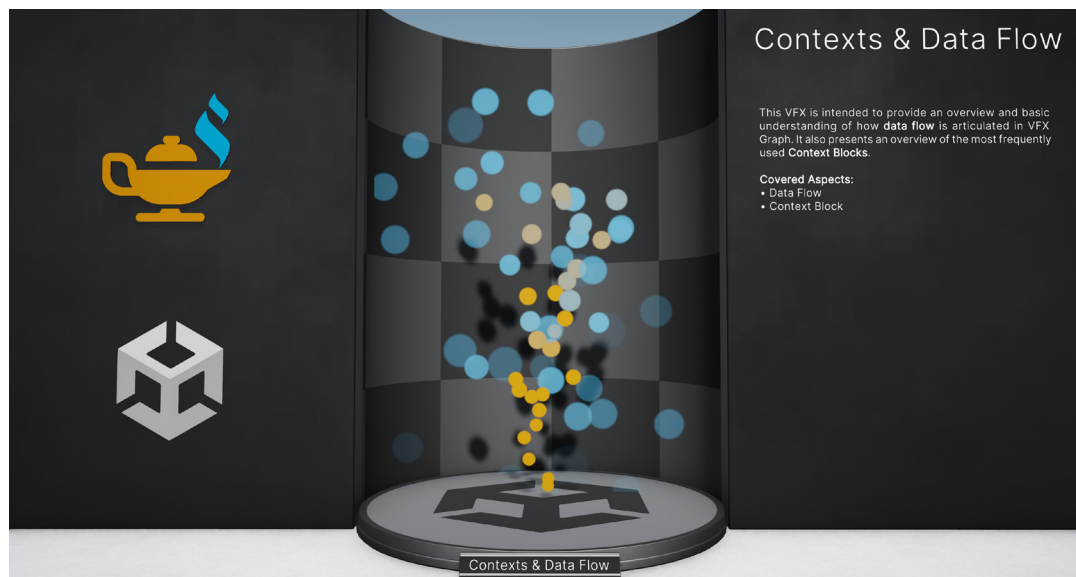
バウンドギズモは、Initialize Particle コンテキストのバウンド設定を表す。

グラフの基礎

これらのサンプルは、グラフロジックの基本と、VFX Graph を構築して最適化する方法を示しています。

Contexts & Data Flow

このサンプルでは、VFX Graph におけるデータフローの処理方法についての概要と基本的な理解が得られます。また、最も頻繁に使用されるコンテキストとブロックの概要も示します。



Contexts & Data Flow サンプル

Spawn Context

この VFX Graph は、Spawn コンテキストのオプションを使用してパーティクルを最初に表示する方法を示しています。Spawn State を利用することで、ループインデックス、スポーン数、ループの状態、ループ継続時間などの重要な情報にアクセスできます。

このエフェクト例では、テクスチャインデックスを操作することで数字のスタックがカウントされているように見えます。これは、各バースト間にわずかな遅延があるパーティクルの単一のバーストを生成することで実現しています。



Spawn コンテキストを使う。

さらに、Spawn コンテキストで生存期間、色、アルファなどの属性を設定し、後で Initialize Particle コンテキストで継承できるようにする方法を示します。

Capacity Count

この VFX Graph では、ボリューム内にランダムなサイズのパーティクルを生成することで、キャパシティ属性の使用方法を説明しています。

Capacity Count は、システムのパーティクルメモリ割り当てに使用されます。この値を大きくすると、割り当てられるメモリが増加します。この数が、現在アクティブなパーティクルの最大数となります。

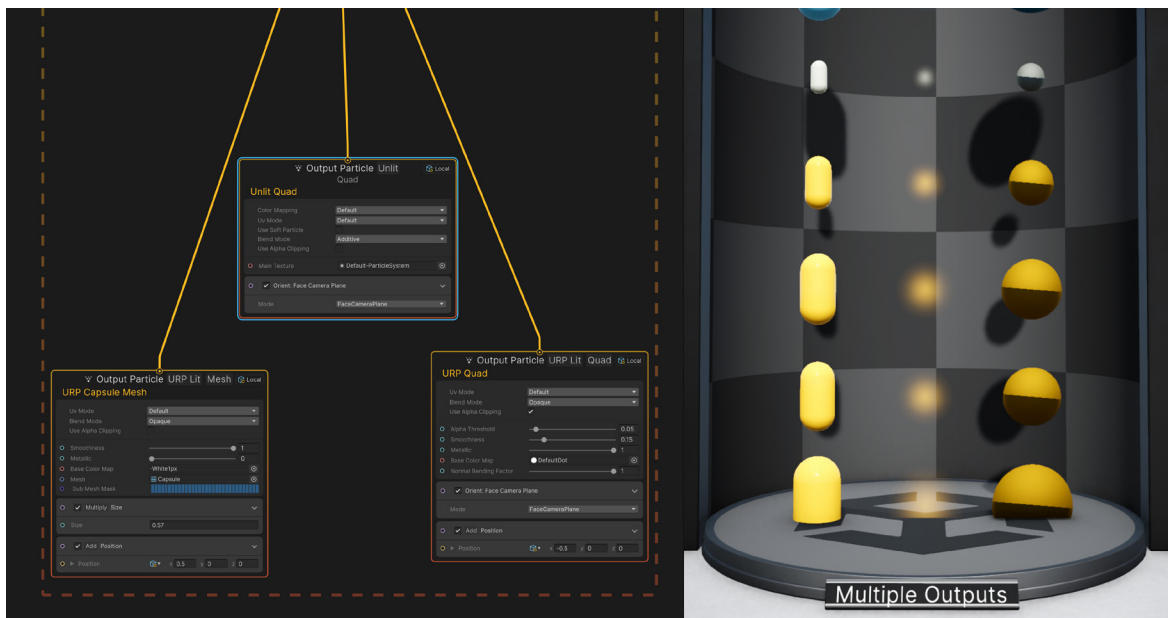


Capacity Count を大きくすると、割り当てられているメモリが増加する。

Multiple Outputs

この VFX Graph は、Initialize または Update コンテキストを 1 つまたは複数の出力コンテキストに結び付ける方法を示しています。これにより、1 つのパーティクルシミュレーションを作成し、各パーティクルに数種類のレンダラーを生成できます。

このデモでは、各パーティクルを unlit パーティクル、メッシュ、クアッドとしてレンダリングします。例えば、メッシュの Output を使用してミサイルのコアを作成し、2 つ目のクアッドの Output を使用してミサイルの先に放射状の光を追加することを想像してみてください。



さまざまなエフェクトを得るために複数の Output を使う。

一部の Output は特定の SRP にのみ対応していることに注意してください (例 : HDRPVolumetric Fog 出力は HDRP にのみ対応)。

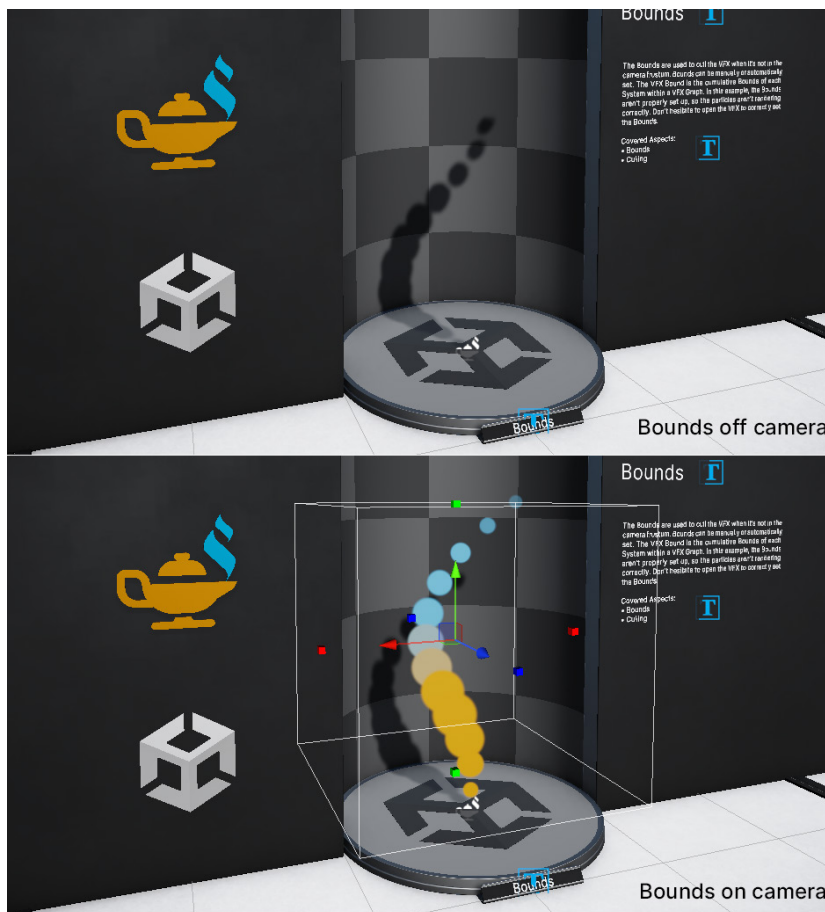
Bounds

このサンプルは、バウンド設定でエフェクトをカリングしてパフォーマンスを改善する方法を示しています。この例では、バウンドがデフォルトでカメラの錐台の外側に意図的に設定されています。その結果、パーティクルはカリングされます (この例では、Output コンテキストで有効にしても、影が差すことがあります)。

バウンド Gizmo がカメラの錐台内に入り、パーティクルが再表示されるようにバウンド設定を調整します。

バウンドモードの選択 :

- **Manual (手動):** Bound Size と Center を手動で設定します。
- **Recorded (記録):** ユーザーは VFX Control を使用してバウンドを記録し、その設定をシステムに適用できます。
- **Automatic (自動):** バウンドはフレームごとに計算されます。これは、動的 VFX や VFX のイテレーション時に必要になりますが、リソース消費が激しくなります。可能な場合は Manual モードもしくは Recorded モードを使用してください。



バウンド設定を調整して、パーティクルを再表示させる。

パーティクルの方向と回転

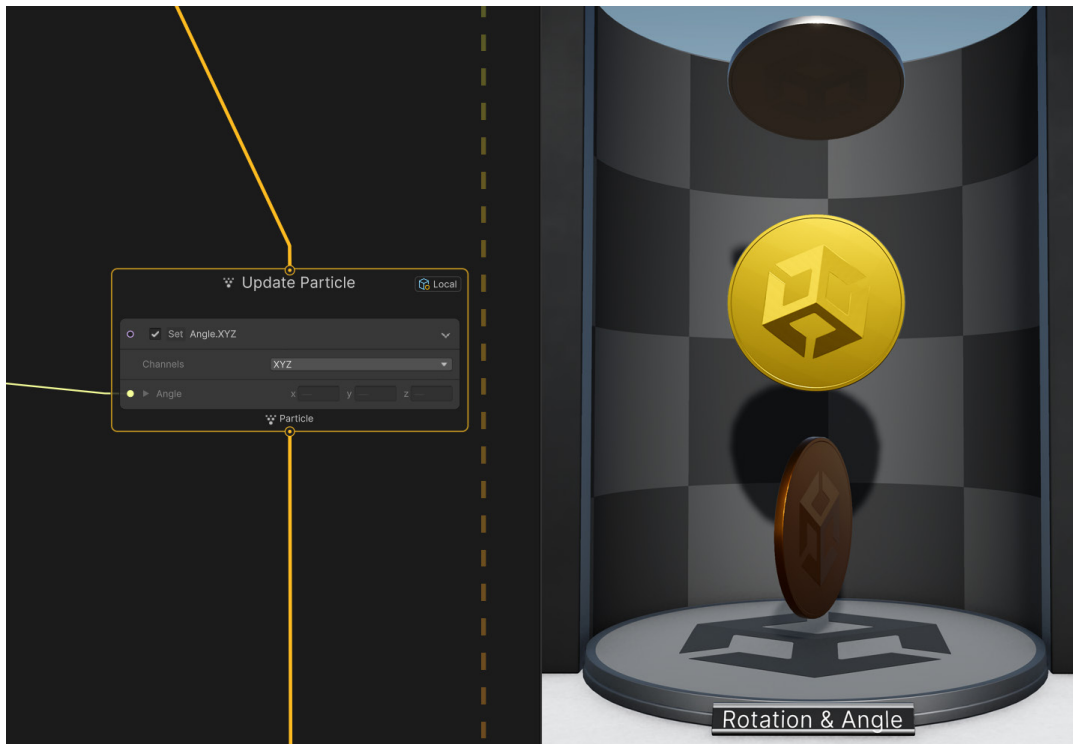
出力がクアッド、メッシュ、ストリップのいずれに設定されていても、パーティクルのジオメトリの方向を制御したい場合があるでしょう。カメラに向けるべきでしょうか。前方向や上方向はどうでしょうか。Orient ブロックは、さまざまなモードでパーティクルの方向を簡単に決めるのに役立ちます。

Orient Face Camera、**Orient Fixed Axis**、**Orient Advanced** VFX Graph は、パーティクルの方向を使うさまざまな方法を示しています。



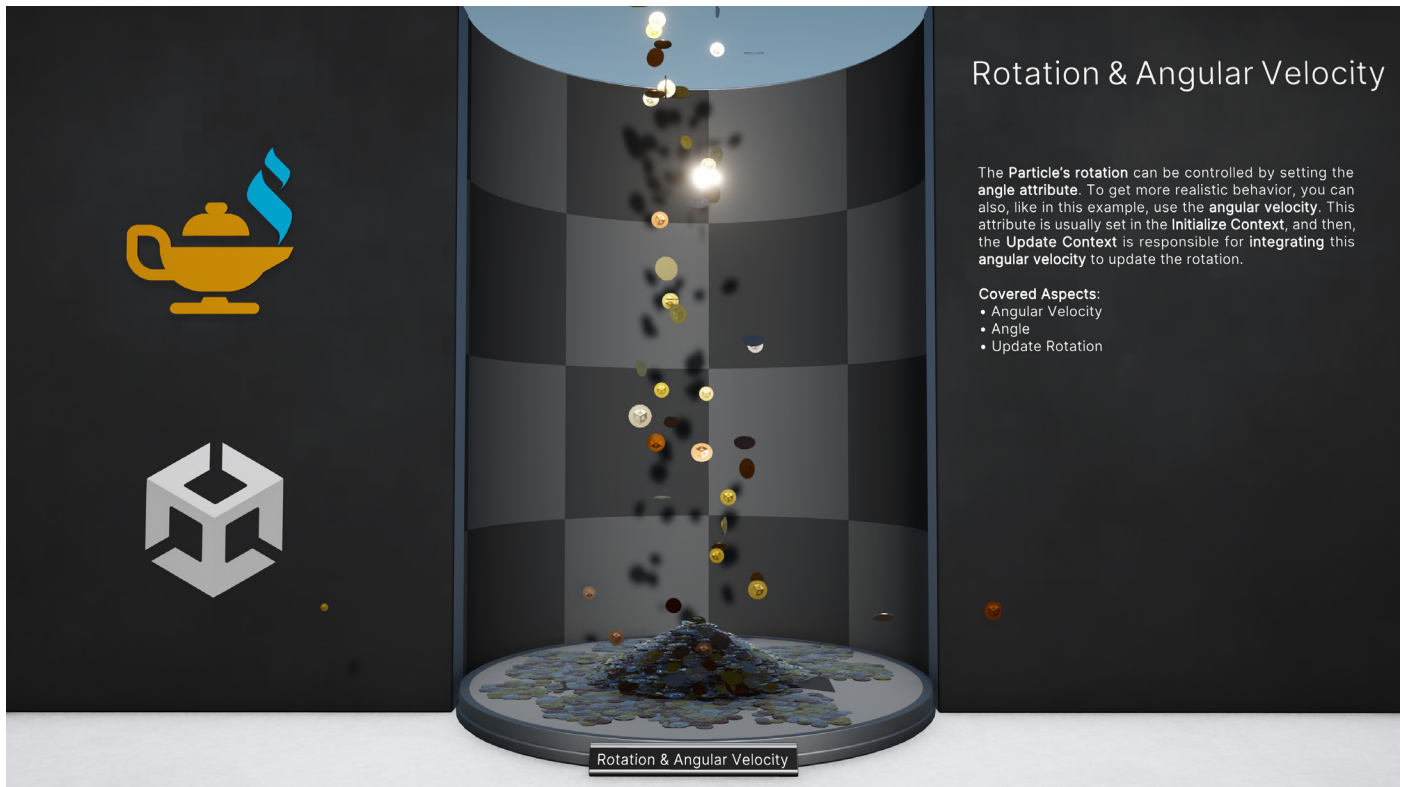
これらのサンプルは、パーティクルの方向を変えるさまざまな方法を示している。

Rotation & Angle VFX Graph が示すように、Angle 属性を使用してパーティクルを回転させることもできます。



Angle 属性を使用して、パーティクルを回転させる。

よりリアルな結果を得るには、角速度を使用します。**Rotation & Angular Velocity** サンプルは、これを Initialize コンテキストに設定し、Update コンテキストを使用して回転を更新する方法を示しています。



よりリアルな動きのために角速度を使う

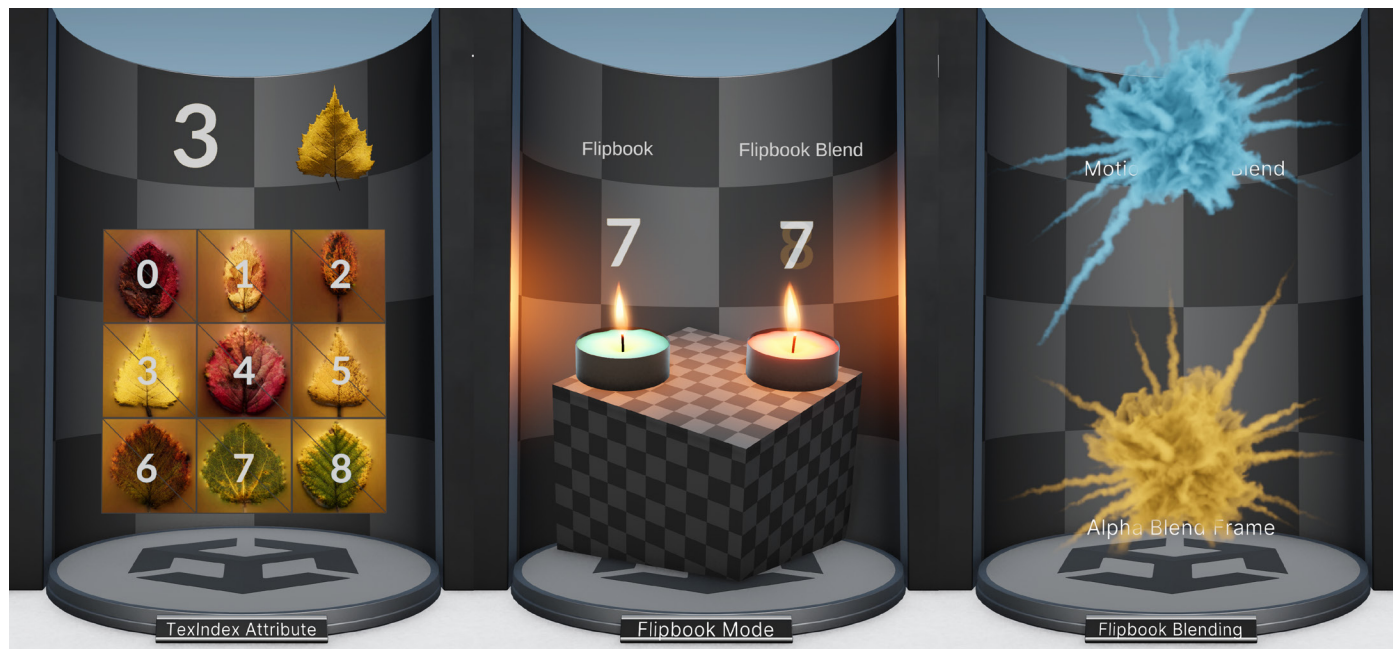
テクスチャリングとフリップブック

リアルタイム VFX はスプライトシートを使用することがよくあります。**TexIndex Attribute** サンプルは、スプライトシートを使用するように Output の UV モードをフリップブックに設定する方法を示しています。

Flipbook ブロックはスプライトシートのアニメーション化に役立ちますが、スプライトシートのどの部分を表示するかを決定する Float 型の **texIndex** 属性を操作することで、手動で制御できます。

Flipbook Mode VFX Graph では、**Flipbook Blend Frames** を有効にすることで、アニメーション化された基本的なフリップブックテクスチャを超えた、より滑らかな補間アニメーションを確認できます。

Flipbook Blending VFX Graph は、従来のフレームブレンディングとモーションベクトルを使用したフレームブレンディングの違いを示しています。モーションベクトルブレンディングは、テクスチャを使用してフレーム間のピクセルディスプレイメントを表します。これは、フリップブックのフレーム数を減らしたり、エフェクトをスローモーションで表示したりするのに役立ちます。



これらのサンプルは、フリップブックテクスチャの使い方を示している。

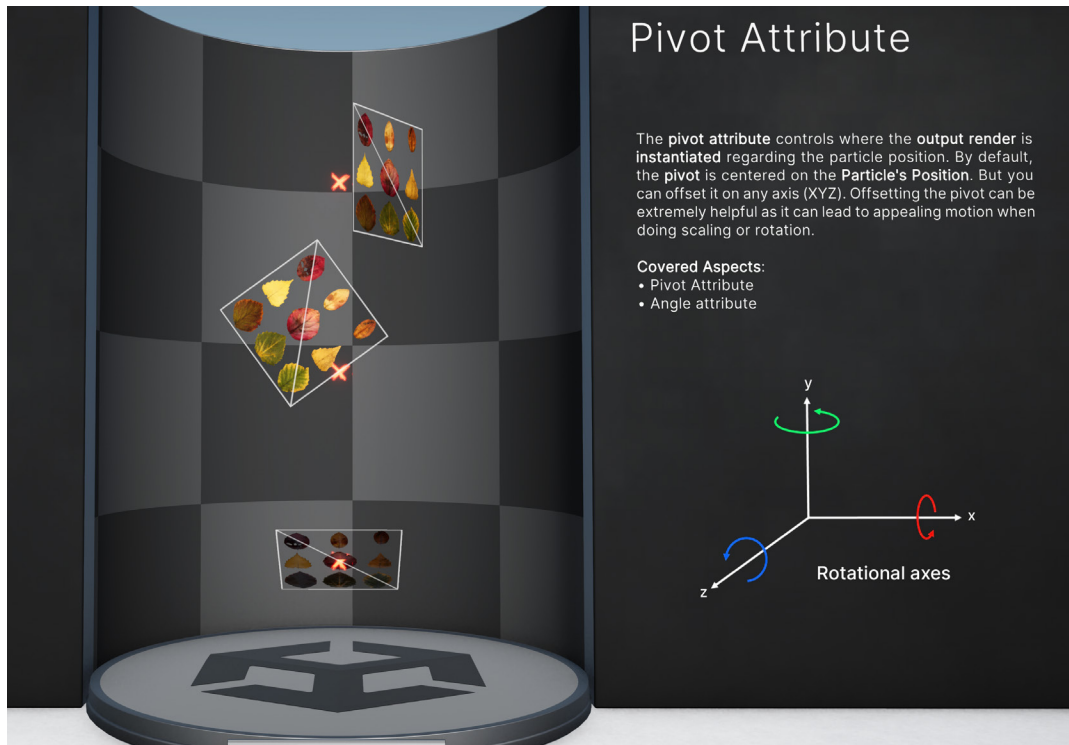
TexIndex Advanced VFX Graph は、**texIndex** 属性をクリエイティブに活用している複数のシステムで構成されています。属性値を活用するために、時間、ノイズ、さらにはパーティクルの位置が使用されます。複数のパーティクルをまとめてアニメーション化することで、マルチレイヤーのモーショングラフィックエフェクトを生成します。



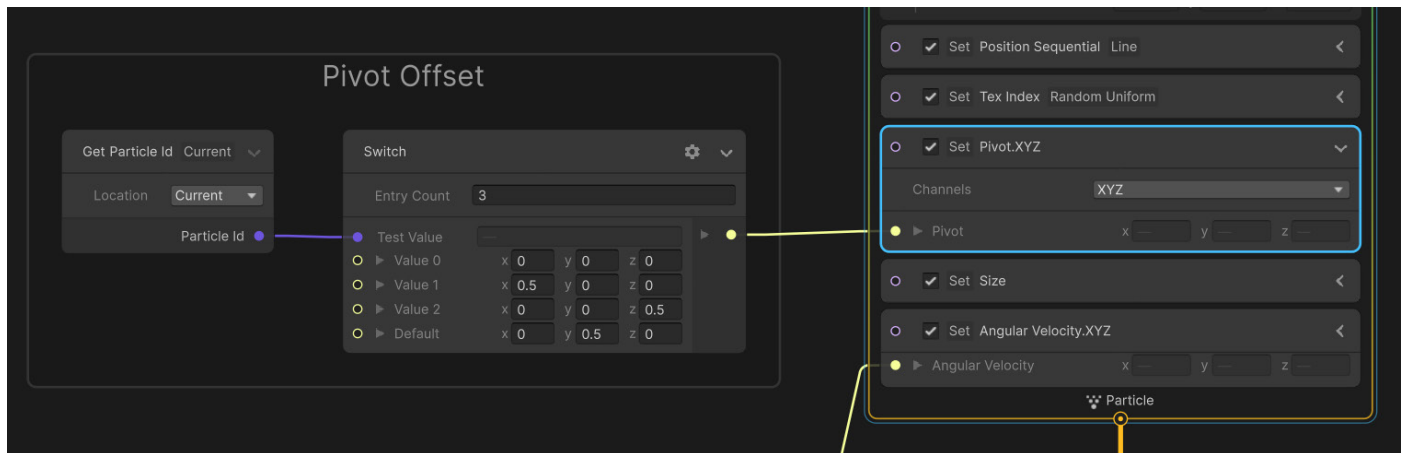
複数のシステムをモーショングラフィックエフェクトに統合。

パーティクルのピボット

デフォルトでは、ピボットはパーティクル位置の中心に置かれますが、任意の軸 (XYZ) でオフセットできます。**Pivot Attribute** VFX Graph では、パーティクルごとにピボットオフセットと角速度が異なります。



Pivot Attribute VFX Graph では、ピボットが異なる 3 つのパーティクルが表示される。



パーティクルのピボットを調整する。

パーティクルの **Pivot** 属性を制御することで、興味深い動きをさせることができます。**Pivot Advanced** VFX Graph が、ピボット操作の例を示します。花卉、葉、花穂はすべてパーティクルです。

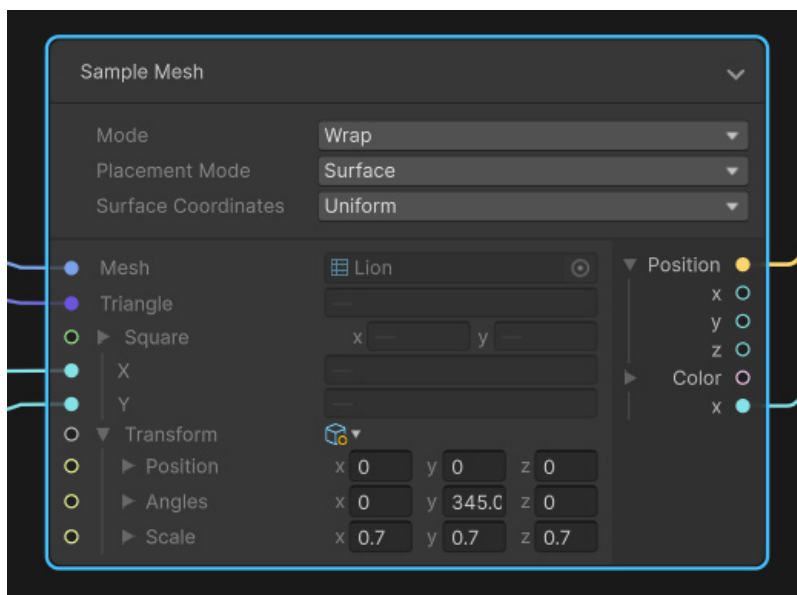
花卉のピボットは、最初は付け根にセットされているため、花卉がくっついている間は適切になります。パーティクルの生存期間が進むにつれて、ピボットが中央にずれるようにアニメーション化され、花卉が離れて自然に宙に浮くようになります。



ピボットをアニメーション化すると、興味深い動きが生まれる。

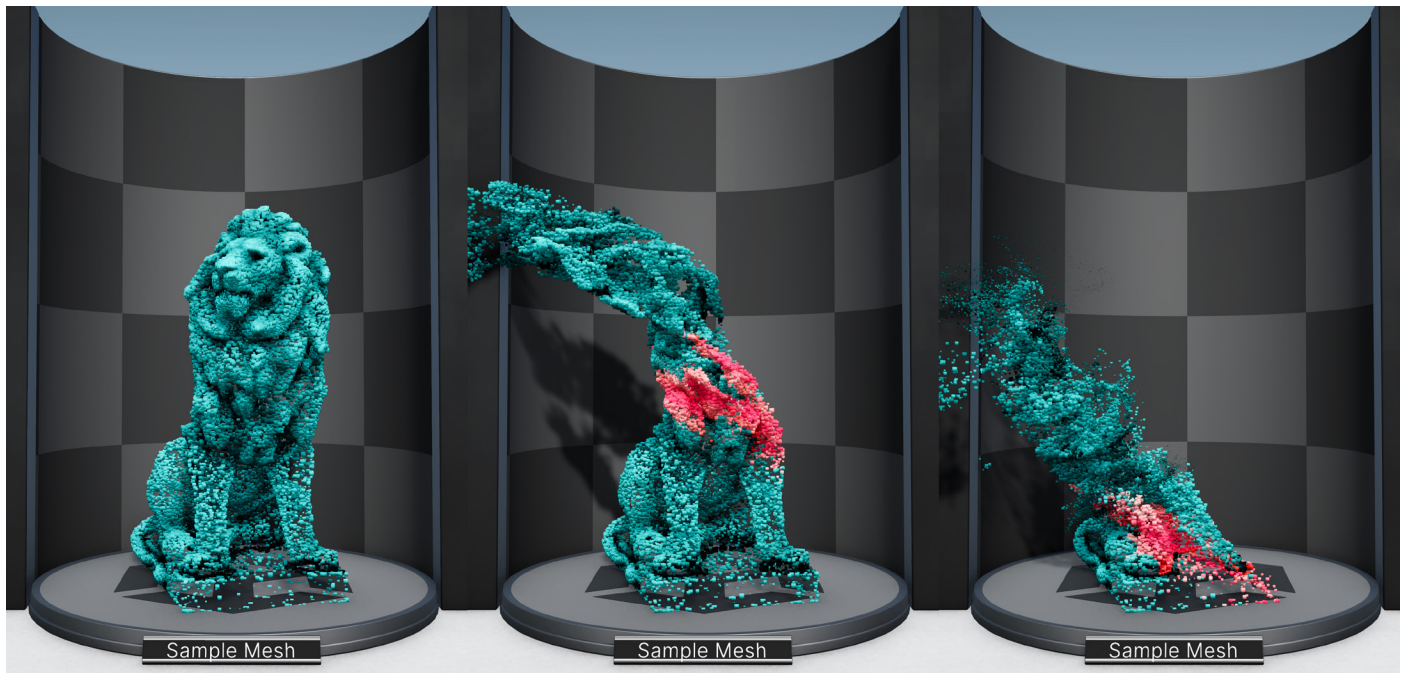
メッシュとテクスチャのサンプリング

Mesh Sample VFX Graph は、メッシュをサンプリングしてパーティクルをスポーンする様子を示します。このグラフでは、**Sample Mesh** オペレーターが使用され、**Mesh** 入力は目的のメッシュを参照し、**Placement Mode** は **Surface** に設定されています。



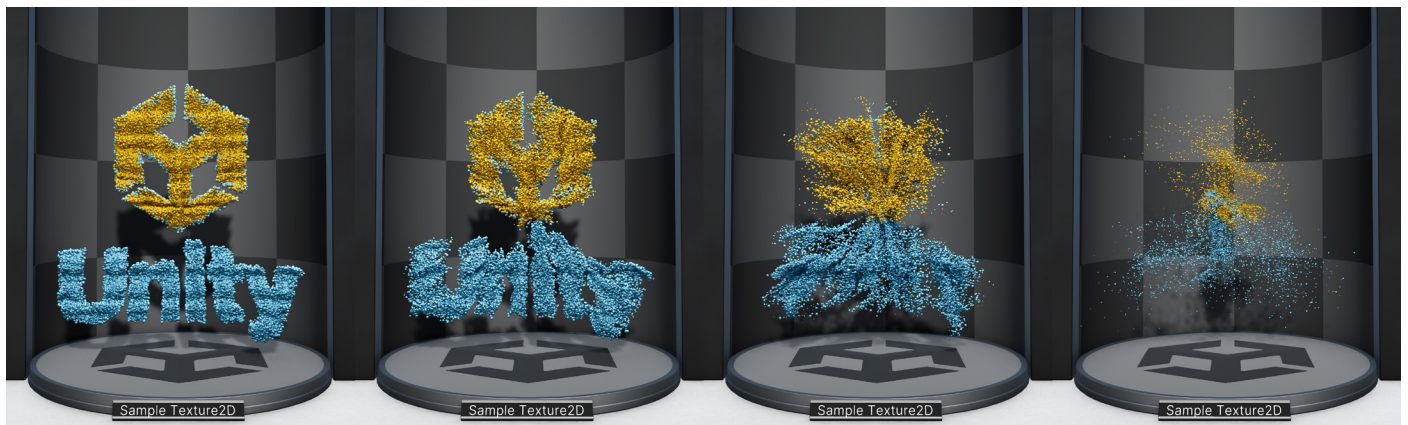
Sample Mesh ノードを使用してパーティクルを表面に配置する。

メッシュの表面にパーティクルがランダムにスポーンされます。**Position** 出力と **Color** 出力は、Initialize Particle コンテキストに渡されます。このコンテキストには、後のアニメーションのためにパーティクルのプロパティが格納されます。この例では、外からの風と乱流が、ライオン像を塵の雲として散らしています。



メッシュの表面にパーティクルをスポーンする。

2D 画像を使用しても同様の効果が得られます。**Sample Texture 2D** VFX Graph は、Texture2D Sample オペレーターを使用してパーティクルの色を決定し、リジェクションサンプリングを実行する方法を示しています。このグラフは 2D グリッド内にパーティクルをスポーンし、テクスチャをサンプリングするためにその XY 座標を再マップします。その後、サンプリングされたテクスチャ値のしきい値に基づいてパーティクルは消滅します。



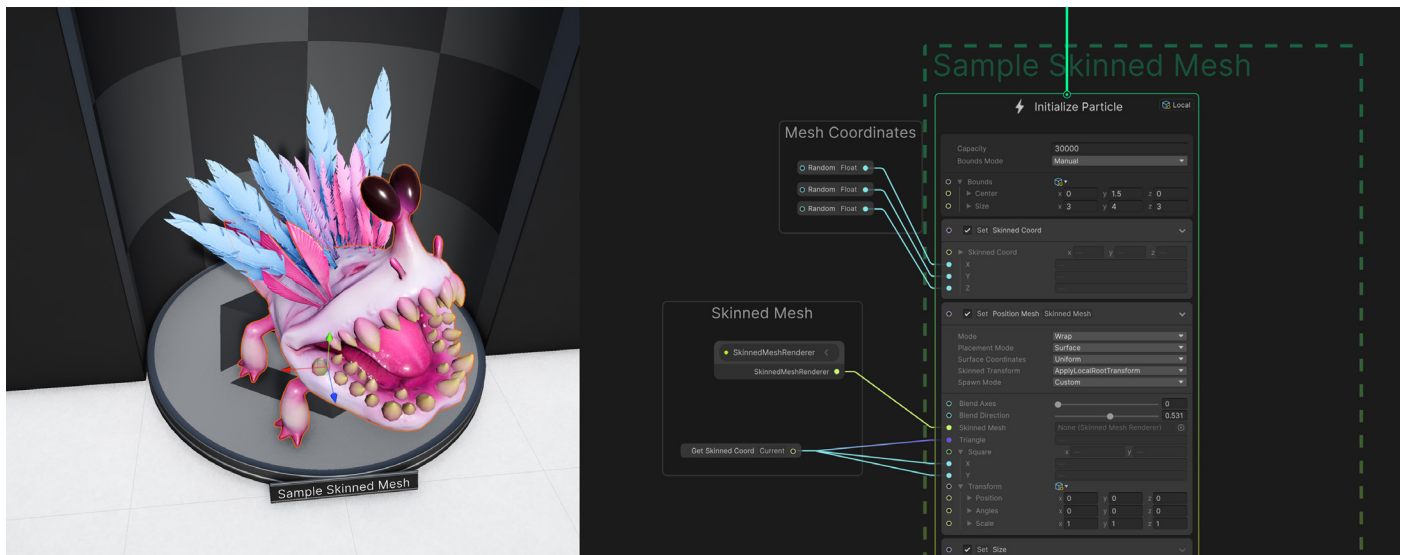
2D 画像にパーティクルをスポーンする。

また、**符号付き距離フィールド (SDF)** をサンプリングすることもできます。SDF は、2D あるいは 3D オブジェクトの形状と輪郭を表す手法です。SDF は、空間内の任意の点からオブジェクト表面の最も近い点までの距離を計算します。**SampleSDF** VFX Graph は、SDF を使用してパーティクルをメッシュの表面に沿って這わせる方法を示しています。



符号付き距離フィールドでパーティクルをスポーンする。

リグ設定されたキャラクターや小道具にエフェクトを適用する必要がある場合は、スキinnedメッシュをサンプリングして、表面位置、頂点色、UV、法線などに関する情報を取得します。**Sample Skinned Mesh** VFX Graph は、クリーチャーの背中に羽をスポーンするために表面の UV を取得しています。



スキinnedメッシュをサンプリングすると、このエフェクトが得られる。

衝突とインタラクティブ性

VFX Graph は GPU 上でパーティクルをシミュレートするため、そのパーティクルは通常のリジッドボディコライダーと衝突できません。ただし、ボックス、円錐、平面、スフィアといった幾何学的プリミティブや複雑な形状と、[Collide with Signed Distance Field](#) ブロックや [Collide with Depth Buffer](#) ブロックを介して衝突できます。



Collision Simple VFX Graph

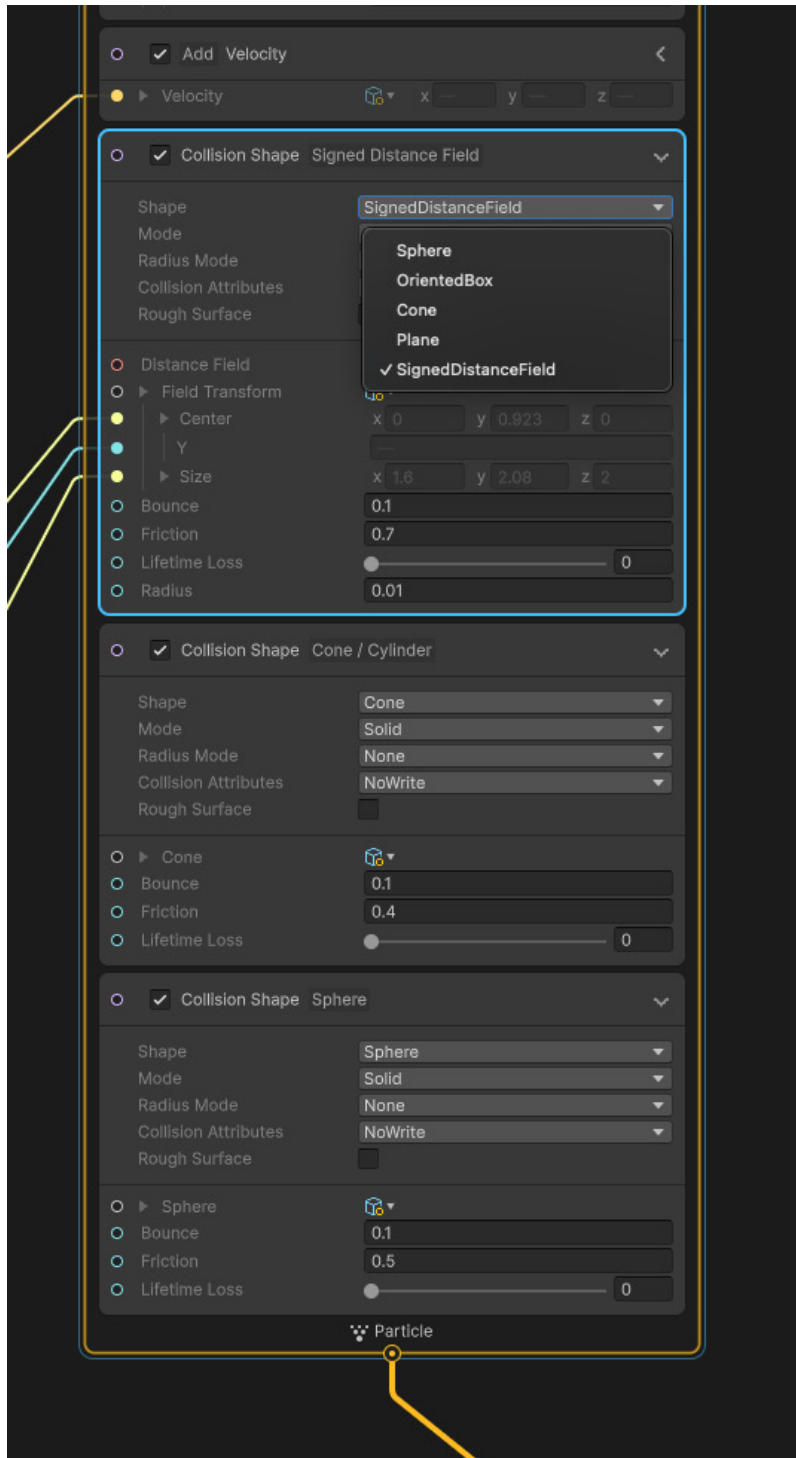
Collision Simple VFX Graph を開いて、基本的な衝突グラフロジックの設定方法を確認してください。

Collision Properties VFX Graph は、Bounce、Friction、Lifetime Loss、Roughness などのプロパティがパーティクルの衝突応答にどのように影響するかを示しています。



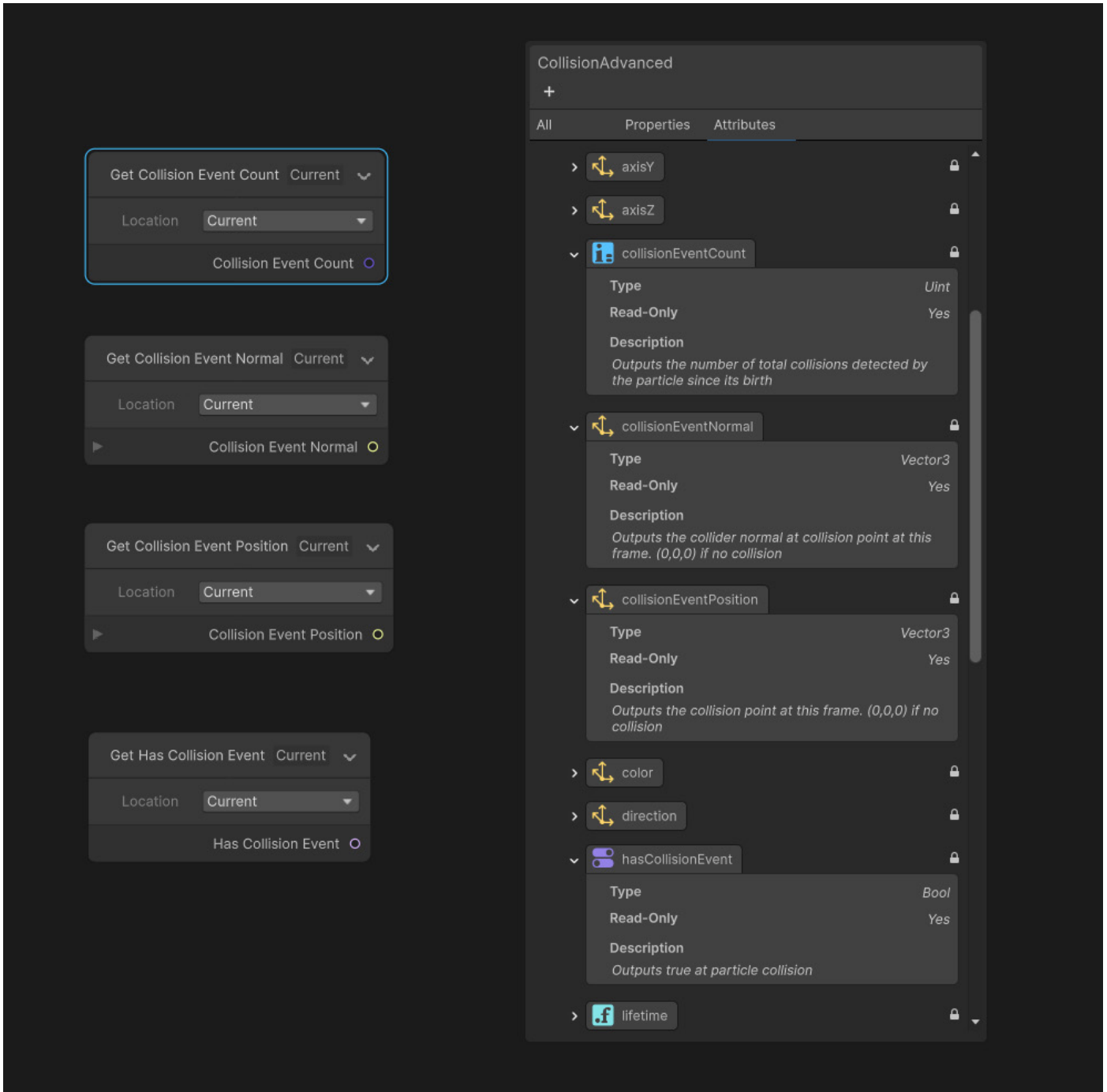
Bounce、Friction、Roughness は衝突に影響する。

Unity 6 では、個々の **Collider ブロック** が Collision Shape ブロックに統合され、形状を簡単に切り替えることができるようになりました。いくつかのオプションが追加され、安定性と精度の両方が改善されています。1つのコンテキスト内で複数の Collision Shape ブロックをつなぎ、目的のエフェクトが得られるようにします。



Collision Shape ブロックを使用してコライダーの形状を切り替える。

いくつかの新しい衝突属性 (**collisionEventCount**、**collisionEventNormal**、**collisionEventPosition**、**hasCollisionEvent**) は衝突イベントの詳細情報を提供し、より正確な制御とインタラクティビティを可能にします。



新しい衝突属性を使用すると、パーティクルの衝突をより正確に制御できる。

単純な衝突形状では環境との十分な衝突精度が得られない場合、符号付き距離フィールドは複雑なジオメトリを近似するのに適したソリューションです。**Collision Advanced** VFX Graph は、パーティクルと手の彫刻との衝突に [Collide with Signed Distance Field](#) ブロックを使用する方法を示しています。



符号付き距離フィールドは、衝突の複雑なジオメトリを近似できる。

デカルパーティクル

デカルを使用して、環境や、さらには動的なオブジェクト、スキンメッシュにも、テクスチャを投影できます。パフォーマンスに大きな影響を与えることなく、シーンのビジュアルを複雑にすることができます。一般的なデカルには次のようなものがあります。

- 弾痕、焦げ跡
- 汚れ、すず
- 落書き、ポスター、看板
- わだち、足跡
- 植物、コケ

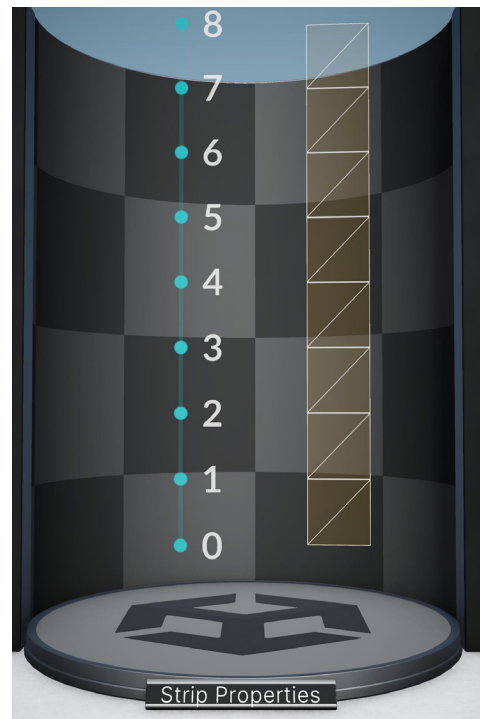
Output デカルを使用すると、パーティクルをデカルとしてレンダリングし、そのプロパティをベースカラーマップ (アルベド)、法線マップ、マスクマップを使用してサーフェスに投影できます。この例では、アニメーション化された Skinned Mesh Renderer にデカルを投影する方法を示しています。



デカールは動的なオブジェクトにもビジュアルの複雑さをもたらす。

パーティクルストリップ

パーティクルストリップシステムは、パーティクル間にクアッドを描画することでリボンやトレイルのようなエフェクトを作成できる、パーティクルがリンクされたグループです。**Strip Properties** テンプレートから、次の 2 つのシンプルなパーティクルストリップを見てみましょう。左側は線、右側は幅の広いリボンです。



Strip Properties テンプレートは、2 つのパーティクルストリップを作成する。

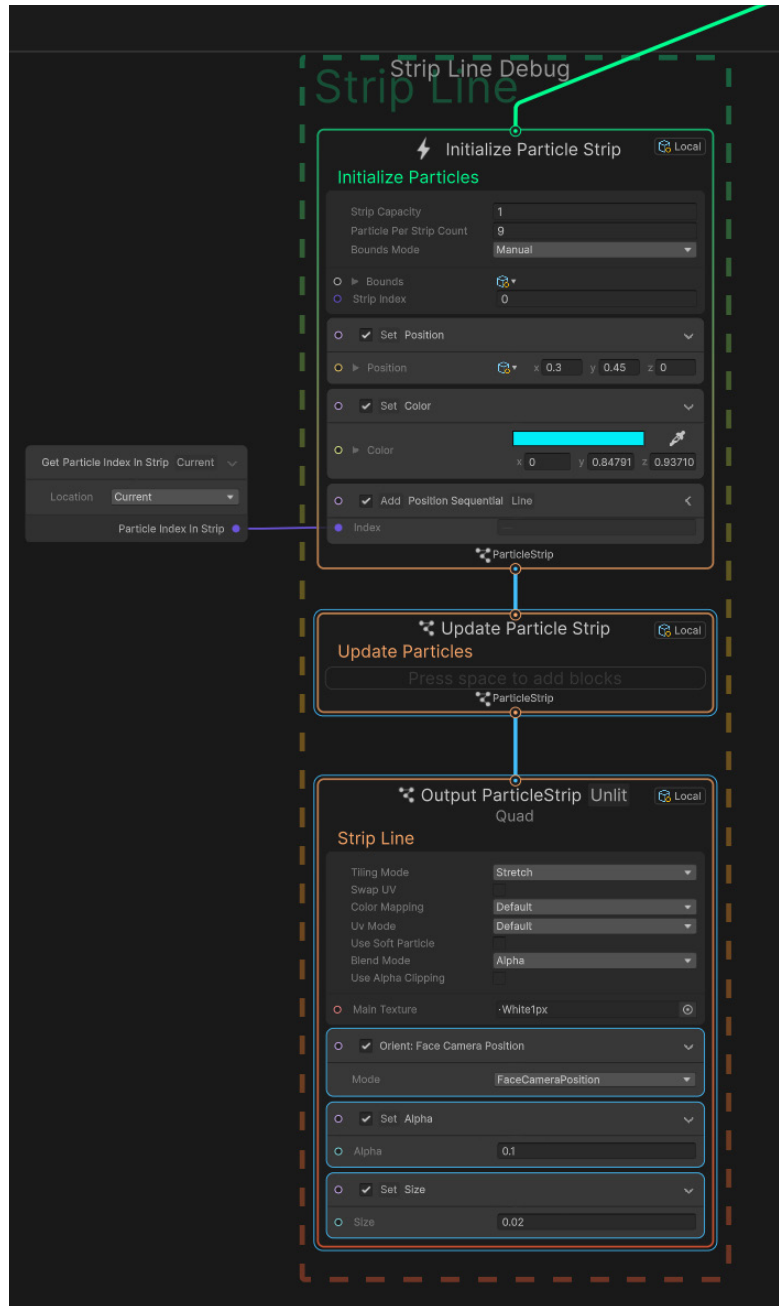
ストリップを生成するには、Initialize Particle Strip コンテキストを使用します。ここでは、Strip Capacity と Particles Per Strip Count を設定します。これらの設定により、システムが処理するストリップの最大数と、各ストリップを構成するパーティクルの数が決まります。どちらの設定もストリップの外観と動作に影響を与えます。

グラフ内では、パーティクルストリップ固有の属性がいくつかあります。

- **Strip Index** は、各パーティクルが属しているストリップを定義します。このようなシンプルなセットアップでは、すべてのパーティクルが 1 つのパーティクルストリップに沿って落下するため、Strip Index は 0 になります。
- **ParticleIndexInStrip** 属性は、Particle Strip バッファ内でのパーティクルの位置を表す uint です。0 からストリップ内のパーティクルの総数（この例では 0 から 8）までの値です。
- この **SpawnIndexInStrip** 属性は、ストリップ内の Spawn Index を表す uint です。ParticleIndexInStrip 属性とは異なり、SpawnIndexInStrip は一意の ID ではありません。同じストリップの異なるフレーム上に生まれた 2 つのパーティクルは、同じインデックス値を持つことができます。

この例では、シングルバーストを使用してすべてのパーティクルが同じフレーム内にスポーンされるため、SpawnIndexInStrip と ParticleIndexInStrip の値は同一です。

Initialize ParticleStrip コンテキストは、Update ParticleStrip コンテキストに接続します。最後に、Output ParticleStrip コンテキストを使用してパーティクルストリップをレンダリングします。ここでは、Stretch、Repeat Per Segment、Custom などの Tiling Mode オプションを使用して、ストリップに沿ったテクスチャのマッピング方法を制御できます。

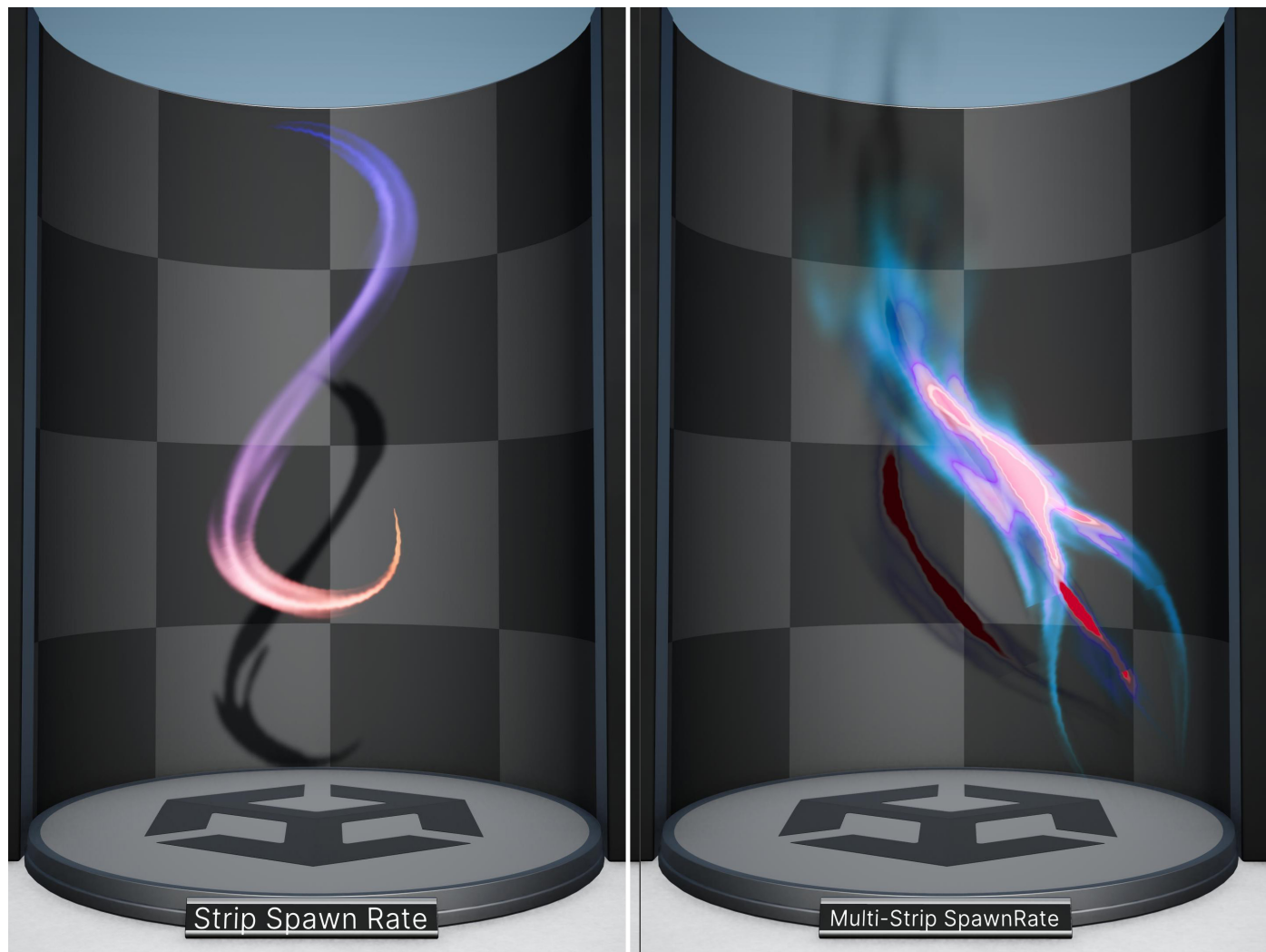


パーティクルストリップの設定。

パーティクルをスポーンする方法はいくつかあります。パーティクルストリップを扱う場合、VFX Graph の設定方法に影響を与える可能性があります。これらの例では、さまざまな設定を紹介しています。

Strip SpawnRate テンプレートは、パーティクルの連続スポーンレートから 1 つのトレイルを作成する方法を示しています。

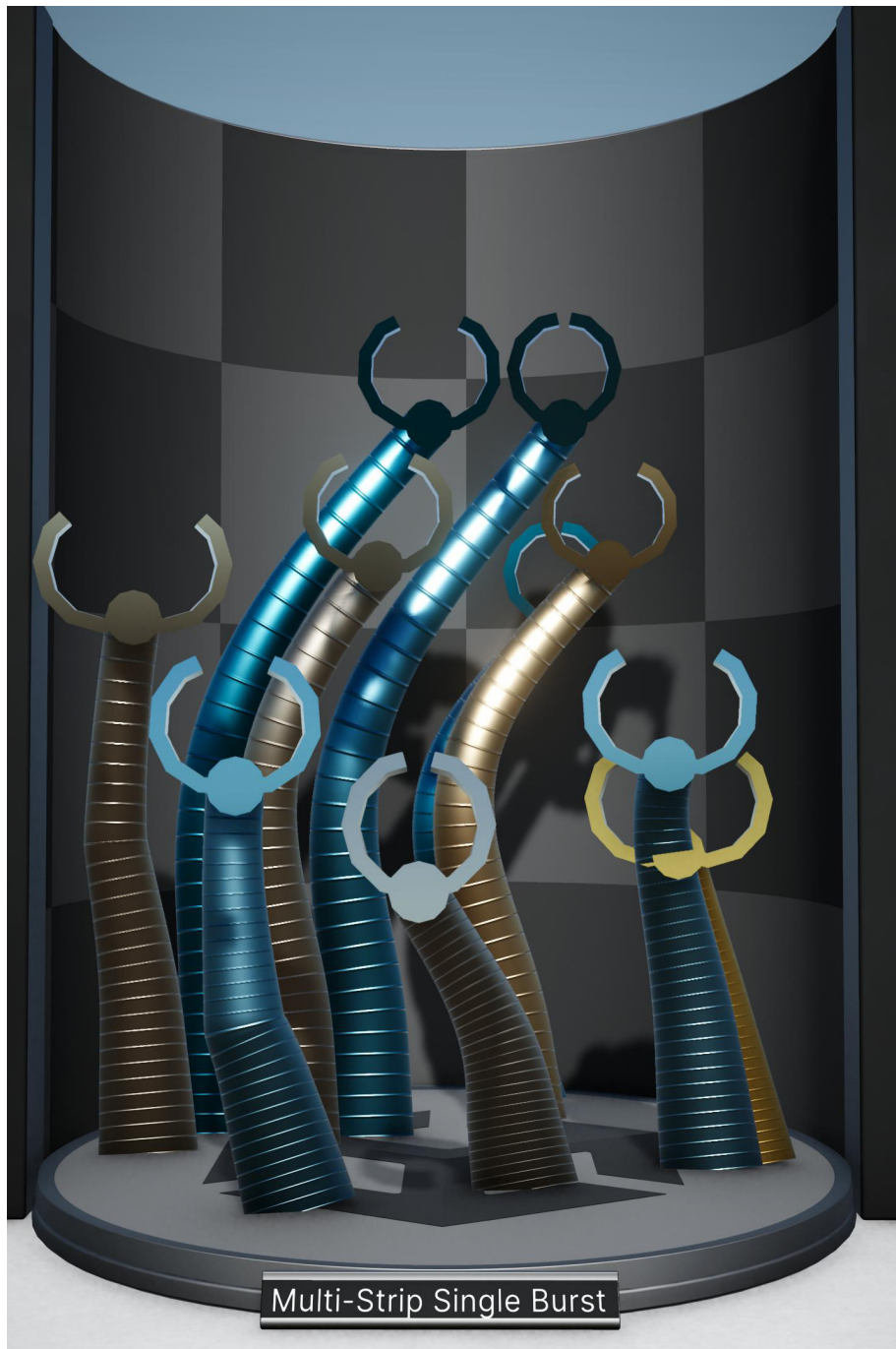
Multi-Strip SpawnRate テンプレートは、連続スポーンレートから複数のトレイルを作成する方法を示しています。



これらの例では、トレイルの作成に連続スポーンレートを使用している。

この例では、1つのバーストから複数のロボットアームを生成します。Strip Index の設定方法については、**Multi-Strip Single Burst** テンプレートを参照してください (Spawn Index をストリップごとのパーティクル数で割る)。

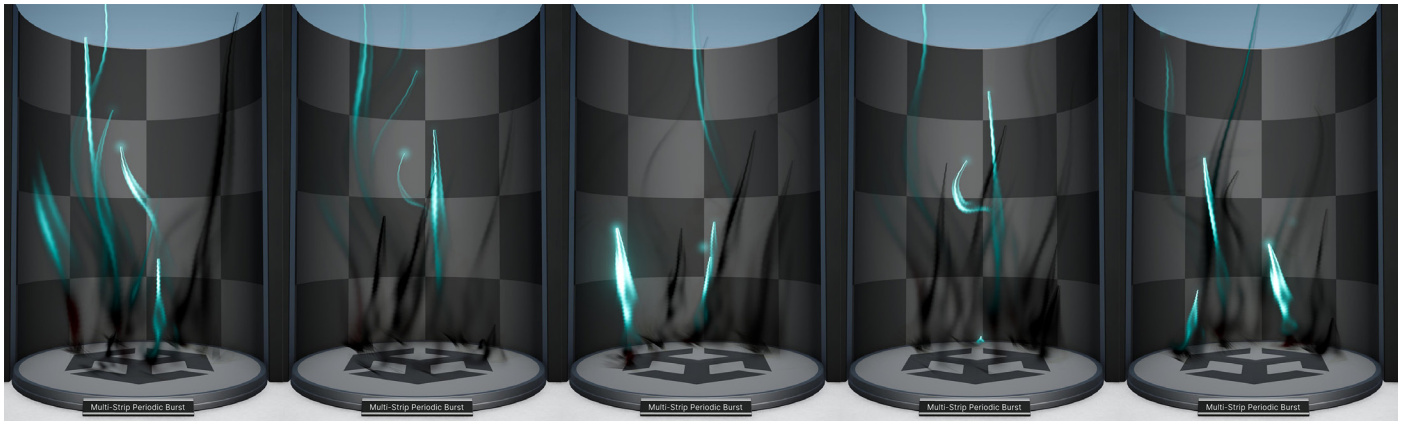
ロボットの腕はパーティクルストリップですが、端にある爪の部分は、Lit Mesh Particle です。



単一のバーストを使用して複数のパーティクルストリップを作成している。

一方、**Multi-Strip Periodic Burst** の例は、システム内の周期バーストごとに新しいトレイルを作成する方法を示しています。Loop Index は Spawn コンテキスト外で使用するために texIndex として保存されます。この値は初期化時に Strip Index として使用され、その後“ストリップ比率”に正規化されます。

このストリップ比率を計算することは、パーティクルストリップを扱うときに役立ちます。この例では、伸びたり、サイズを変えたり、重力や乱流などの力に反応したり、といった動作や外観に影響しています。

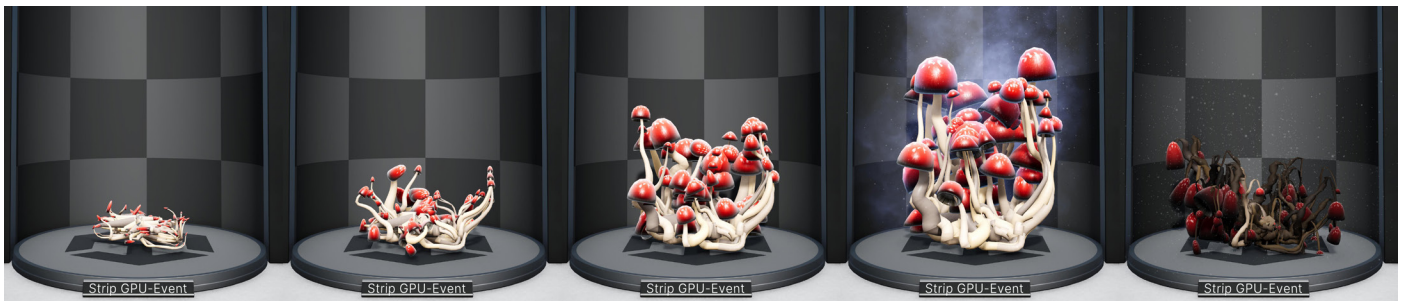


ストリップ比率を計算してパーティクルストリップの動作を管理する。

この **Strip GPU イベント** の例は、キノコが成長する VFX です。キノコの傘 (Mushroom Hat システム) はパーティクルメッシュをレンダリングし、キノコの柄 (Mushroom Foot システム) はパーティクルストリップから作られています。

一方のシステムが他方の“親”のように機能し、GPU イベントを使用して次のシステムをトリガーします。

このメカニズムは、キノコが完全に成長すると埃の雲や煙を生成し、エフェクトに加えます。



GPU イベントがキノコを成長させる。

Multi-Strips GPU イベントテンプレートは、パーティクルストリップを使用してより複雑なエフェクトを作成する方法を示しています。この例では、各ヘッドフォンプラグはパーティクルメッシュで、親 1 つごとに 1 ストリップという通常のパーティクル生成方法を回避して、そのパスに沿ってパーティクルをスポーンします。

各プラグは床からスピーカーの下にあるジャックまで Bezier の経路をたどり、有機的に動く蛇行したケーブルのトレイルが残ります。プラグが正常に接続されると、緑色の光のパーティクルが追加され、エフェクトにダイナミックなタッチを加えます。



パーティクルストリップがヘッドフォンプラグの後ろを蛇行する。



VFX Graph のサンプル (HDRP)

Visual Effect Graph サンプルは、VFX Graph に関連するさまざまなシナリオを取り上げています。サンプルグラフの一部はケーススタディとして見ることもできるので、プロジェクトをダウンロードして実践的なリファレンスとして役立ててください。

注：このサンプルは HDRP のみをサポートしており、URP には対応していません。

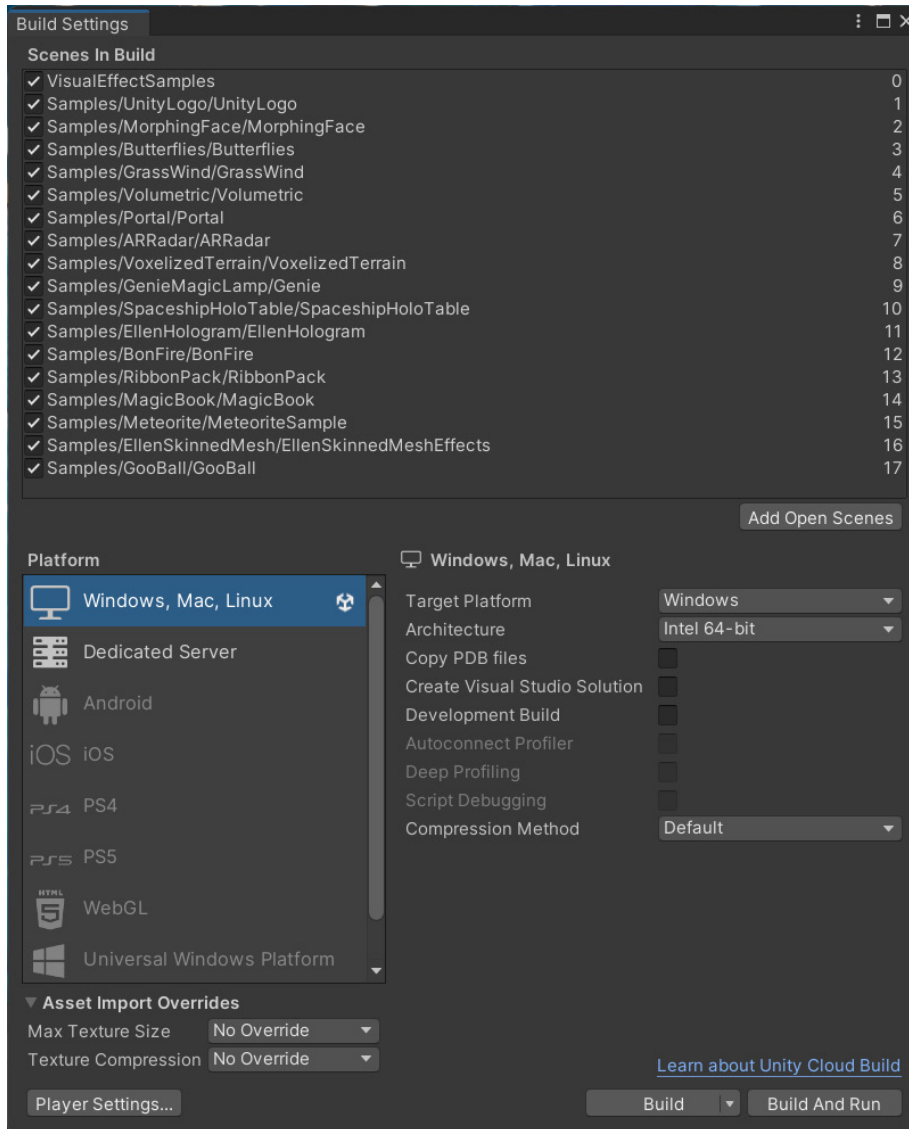


リポジトリで入手できる VFX Graph サンプルの 1 つ

[Release](#) タブに移動すると、これらのサンプルのスナップショットのほか、ビルド済みのバイナリへのリンクも見つかります。あるいは、リポジトリ全体のクローンを作成することもできます。

各サンプルは、プロジェクトの **Assets/Samples** フォルダー内のサブディレクトリに表示されます。メインの **VisualEffectsSample** シーンはルートにあります。

プレイヤーを構築する必要がある場合は、**Build Settings** でこのメインシーンがインデックス 0 に設定されていることを確認してください。その後で、予定している他のすべてのシーンを追加します。



Build Settings

VFX Graph Samples の各シーンパーツには、独自のエフェクトが用意されています。その一部を詳しく見ていきましょう。

Smoke Portal

Smoke Portal は、ボリュメトリックな濃い煙が渦を巻き、神秘的なポータルを形成します。このエフェクトは、Houdini でシミュレートされた煙のアニメーションと、新しい **6 ウェイスモークライティングマテリアル** (後述) を使用したリアルタイムライティングを組み合わせたものです。この手法により、VFX Graph で 6 軸からバイクされたライトマップを使用できます。

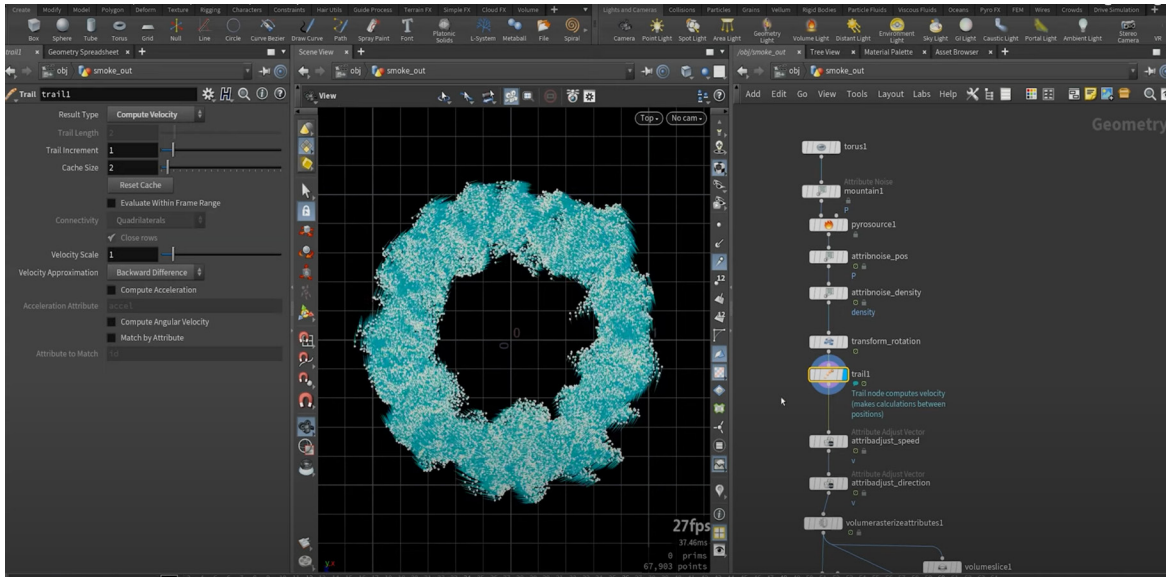
メインポータル周りには、幻想的な外観を際立たせる追加要素があります。揺らめく炎、火花、浮遊する岩や、ポータル周囲の空間をゆがめる微かな歪みエフェクトなどです。



Smoke Portal では、6 ウェイスモークライティングマテリアルを紹介している。

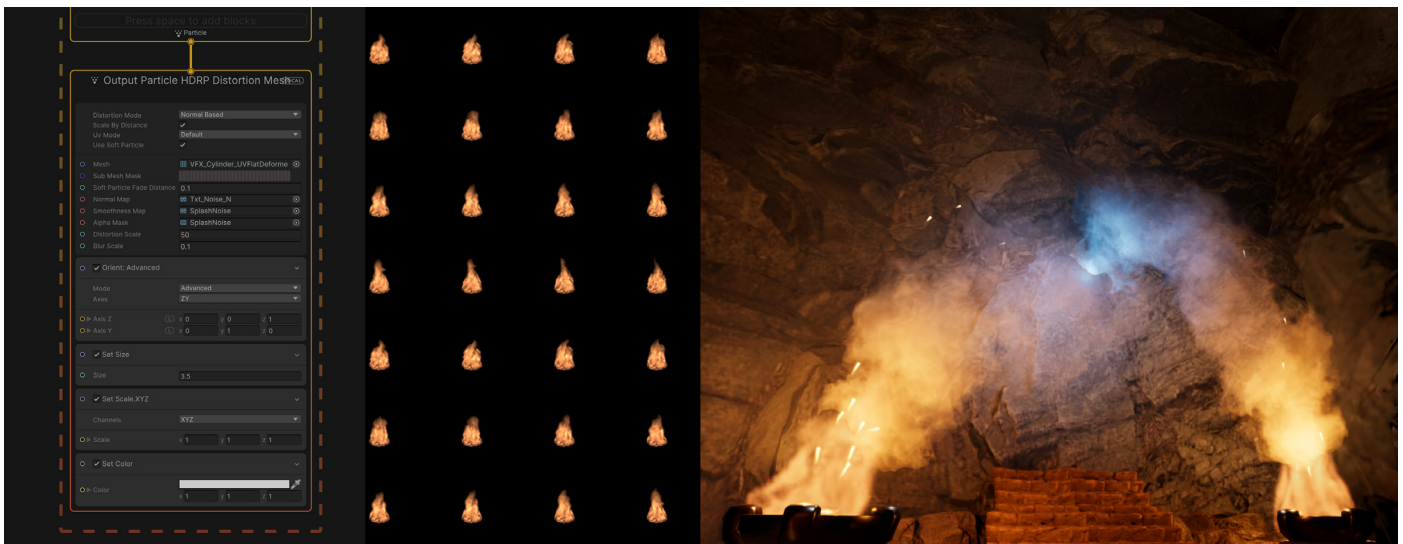
メイキングビデオでは、このサンプルの完全なワークフローを説明します。作成方法の重要なステップをいくつか紹介します。

- **Houdini シミュレーション**: 煙のポータル VFX は、Houdini からエクスポートされた煙のシミュレーションを含むレイヤーで構成されています。元の渦は、Houdini のノイズを持つ円環から始まります。密度、温度、速度の属性によって煙のシミュレーションが行われ、8 x 8 のフリップブックテクスチャとしてエクスポートされます。



煙のポータルは Houdini のシミュレーションとして始まる。

- **炎と追加エフェクト**: 炎のエフェクトはフリップブックのテクスチャを使用して開発され、スモークリングとブレンドされてリアルに表現されています。岩や火の粉がシーンに追加され、重力を感じ、エフェクトが周囲と一体化します。



炎はフリップブックテクスチャである。



- **6 ウェイライティング**:6 ウェイライティング機能では、6 つの異なる軸からベイクされたライトマップを使用します。この手法では、スプライトを使用することで、ボリュメトリックレンダリングにかかる高い計算コストを回避します。リアルタイムライトは煙と相互作用し、真のボリュメトリックレンダリングを行わずにリアルなライティングと影を作り出すことができます。
- **フリップフロップブレンディング**: モーションベクトルは、フリップブックのフレームをブレンドするのに役立ちます。[TFlow](#) (Asset Store より) は、エクスポートされたフリップブックテクスチャからモーションベクトルを生成するのに役立ちます。テクスチャの解像度を下げる場合、TFlow モーションベクトルマップを使用すると、品質低下を補うのに役立ちます。パラボラ状のメッシュにフリップブックテクスチャを適用することで、ボリウムと環境との統合性を高めています。
- **プロシージャルクリスタル**:VFX Graph は、周辺にさまざまなクリスタルを形成するのに役立ちます。[インスタンスング](#)は、VFX クリスタルを単一のバッチにグループ化することで、計算負荷を軽減します。



VFX Graph のインスタンスングは、クリスタル要素の最適化に役立つ。

- **HDRP ライティング**: スポットライトは炎の周囲に焦点を合わせたイルミネーションを提供し、エリアライトはパスに沿ったアンビエントライティングを完成させます。動的ライティングエフェクトは、ライトフリッカースクリプトを使用してライトの位置にアニメーションを追加し、シーンをよりリアルにします。ポイントライトの半径は小さく抑え、ビジュアルクオリティを損なうことなくセットアップを最適化します。

- **最適化** : いくつかの戦略により、VFX Graph のパフォーマンスが向上します。この戦略には、ポータルフリップブックのテクスチャの解像度を 8K から 2K に下げること、アルファクリッピングを使用して透明なオーバードローを減らすこと、クリスタル要素に [VFX Graph のインスタンス機能](#) を活用すること、などが含まれます。

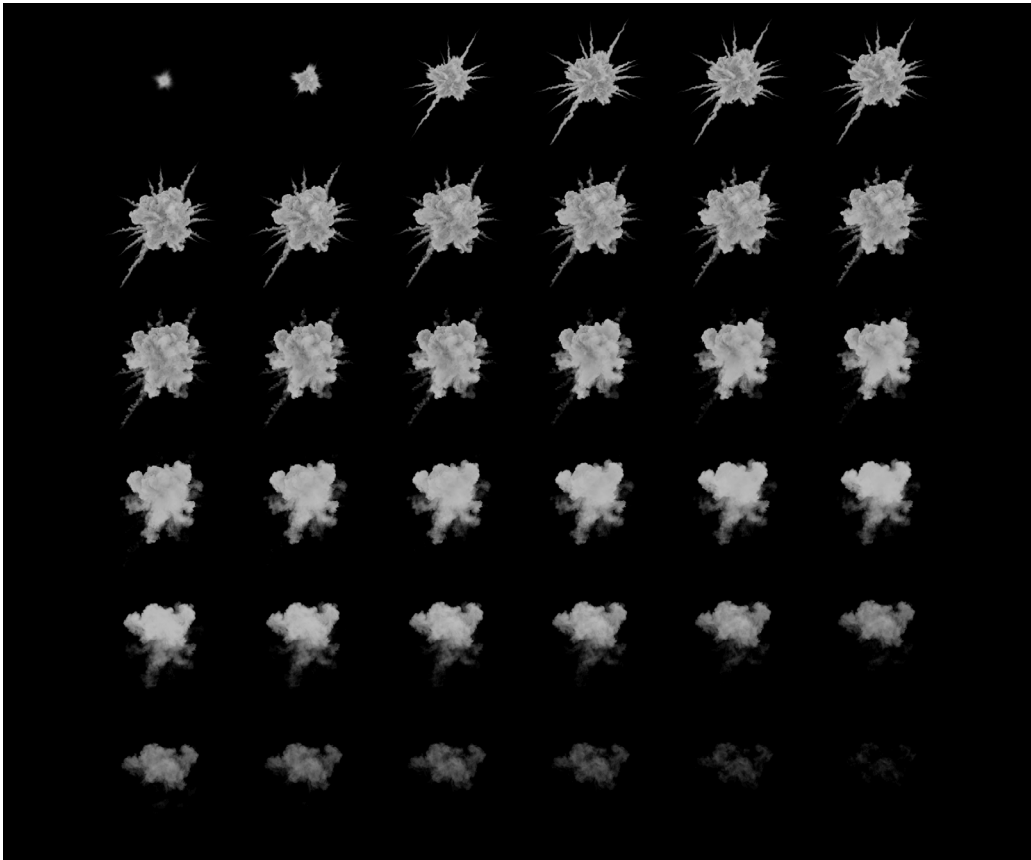


このポータルは、複数の手法を組み合わせで最終的なエフェクトを生み出している。

詳細については、元の[解説動画](#)をご覧ください。

リアルタイムスモークのための 6 ウェイライティング

多くのゲームにおいて、真のボリュメトリックレンダリングの実行は負荷が高すぎるため、リアリティのある煙のリアルタイムレンダリングは難しい挑戦となります。これに対処するため、多くの開発者はフリップブックテクスチャを使用しています。フリップブックテクスチャは通常、テクスチャを $n \times n$ のグリッドに分割します。ただし、表示される解像度は常にほんの一部なので、解像度とメモリが犠牲になります。



スモーク用の典型的な 8x8 フリップブックテクスチャ

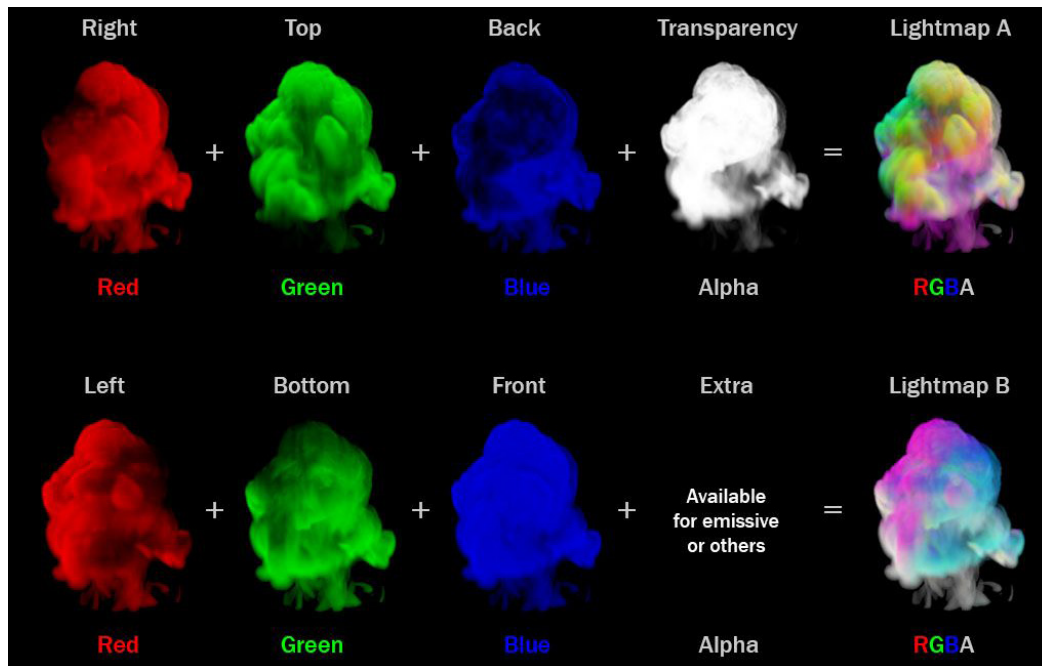
よくある質問です。フラットな 2D ジオメトリに基づいて煙を照らすにはどうすればよいでしょうか？ バリエーションを持たせるためにテクスチャを追加すると、メモリバジェットがすぐに底をつき、テクスチャの品質が低下し、爆発のバリエーションが減少します。法線マッピングや完全にベイクされたカラーマップなどの他の方法は、リアルな見た目や柔軟性に欠けることが多いです。

そこで登場するのが、6 ウェイライティングです。この方法により、ベイクしたシミュレーションからのスモークレンダリングが可能になり、さまざまなライティング条件でうまく機能します。コスト効率の高いプロセスで、煙のポリュメトリックな感触を近似できます。

6 ウェイライティングの秘密は、6 つの異なる方向（上、下、左、右、前、後）から照らしたときの煙の見え方をキャプチャするライトマップのセットを使用することにあります。

その後、これらのライトマップは 2 つの RGBA テクスチャにベイクされます。最初のテクスチャの RGB チャンネルには上、左、右のマップが、アルファチャンネルには下のマップが格納されます。

2 番目のテクスチャの RGB チャンネルには、前、後と、追加のマップが格納され、アルファチャンネルはオプションのエミッシブマスクに利用できます。



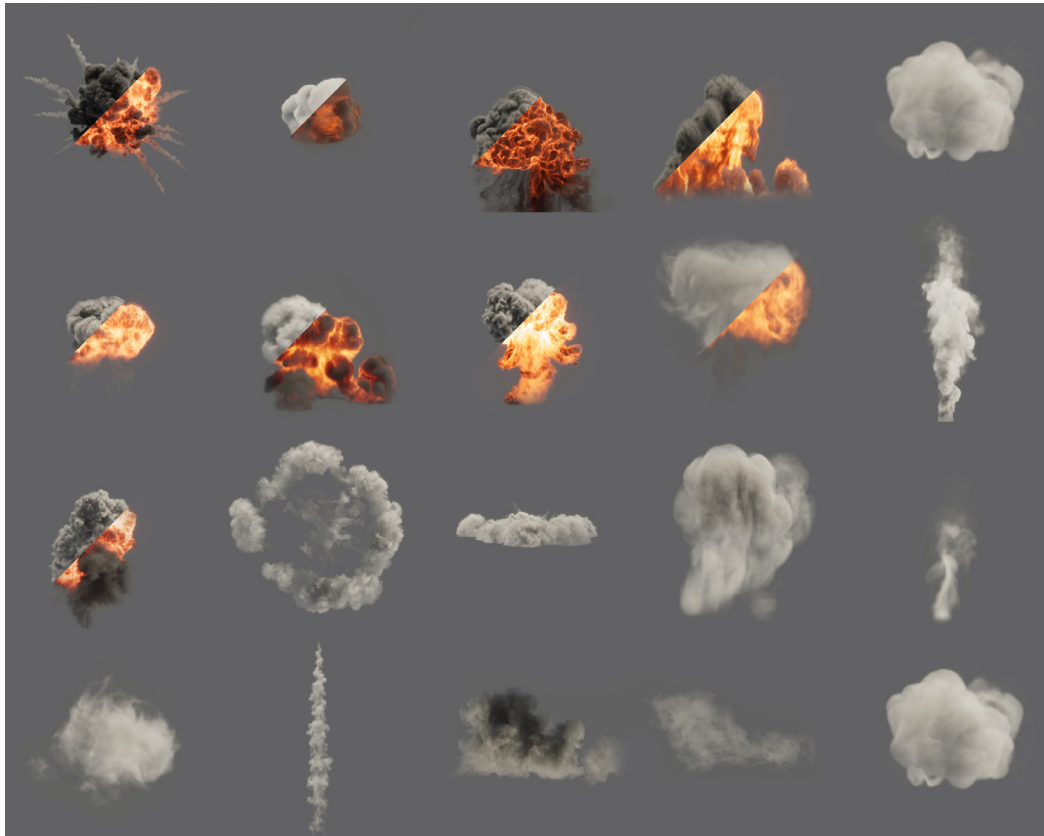
6 つのライトマップは異なるライト方向を捉える。

レンダリング時、シェーダーは各パーティクルに対するライトの方向に基づいて 6 つのライトマップをブレンドします。つまり、すべての種類のライトからの直接光と、ライトプローブからの間接光、およびその他のグローバルイルミネーション手法を使用して、さまざまなライティング条件に合わせて煙を動的にシェーディングできます。



6 ウェイライティングには次の利点があります。

- **複雑な光の相互作用と散乱のエフェクト**：煙の中のリアルな光の散乱をシミュレートすることで、視覚的な複雑さが増し、煙をより自然に見せることができます。
- **環境とビジュアルの統合の改善**：動的シェーディングは、煙が周囲の環境と調和し、ライティング条件の変化に適応するのに役立ちます。



6 ウェイライティングは、さまざまなライティング条件下で多様な煙のレンダリングを可能にする。

- **コストパフォーマンスに優れたレンダリング**：リアルな見た目でありながら、6 ウェイライティングはパフォーマンスに負担をかけず、リアルタイムアプリケーションに適しています。
- **メモリ効率**：この方法では、異なるライティング条件でライトマップテクスチャを再利用することで、異なるライティング条件で複数のテクスチャを使用するよりもメモリを節約できます。



6 ウェイライティングは、リアルタイムのスモークエフェクトをレンダリングするためのビジュアルクオリティ、パフォーマンス、メモリ使用量のバランスを取る、エフェクトツールキットで役立つ手法です。

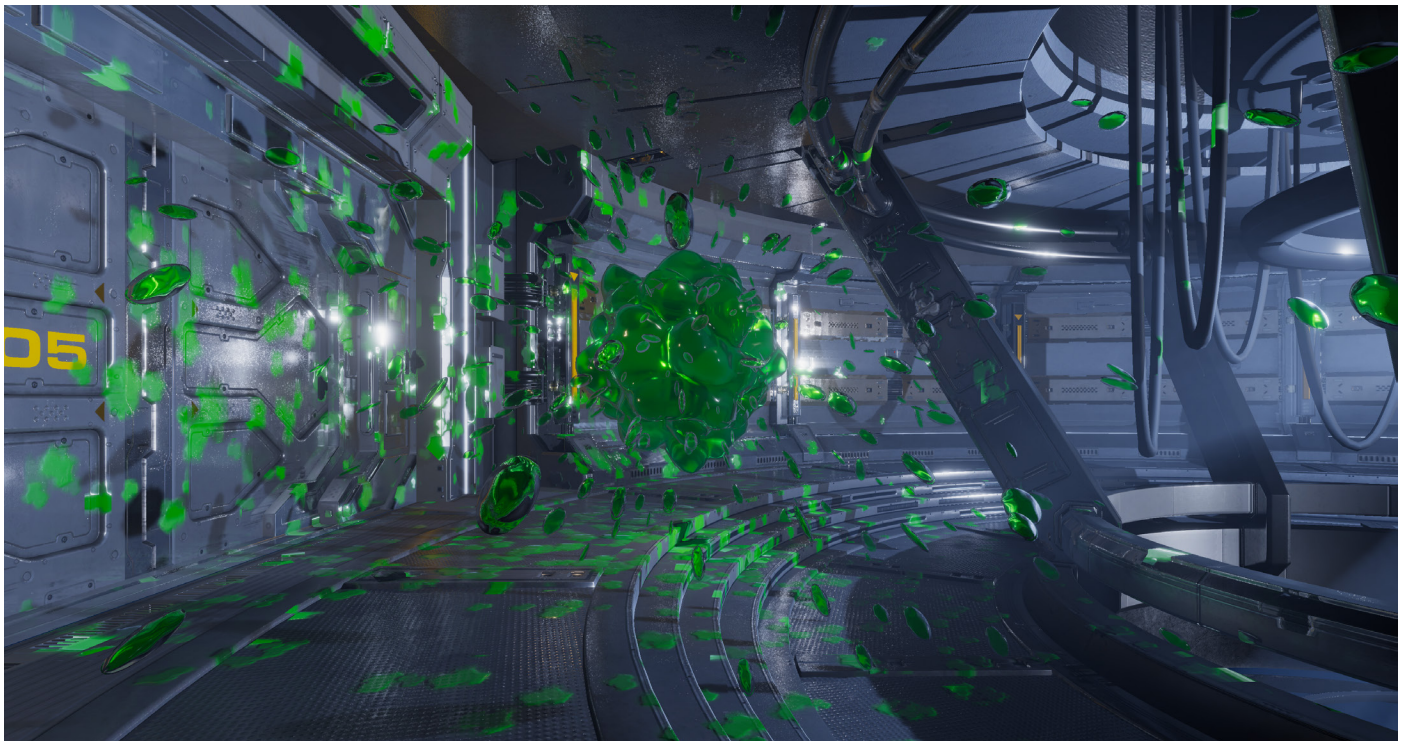


リアルな煙や爆発には 6 方向ライティングを使用する。

[VFX Graph: Six-way lighting workflow](#) (VFX Graph: 6 ウェイライティングのワークフロー) の動画で、このテクニックを解説しています。詳細については、こちらの[ブログ記事](#)も参照してください。

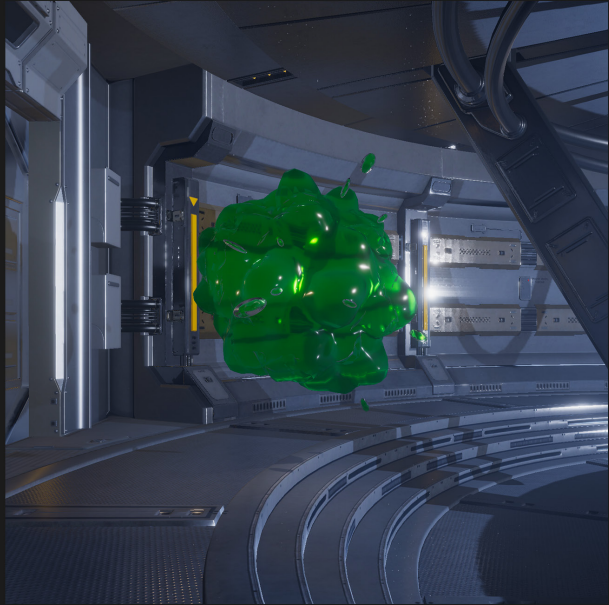
GooBall

Unity でペンキを飛び散らしたいと思ったことはありませんか？ この遊び心のある Sci-Fi デモには、周囲の環境と相互作用するネバネバした物質をシミュレートするデカールが組み込まれています。他の制作例と同様に、このマルチレイヤーエフェクトは、複数のシステムを利用して最終的な見た目を実現しています。

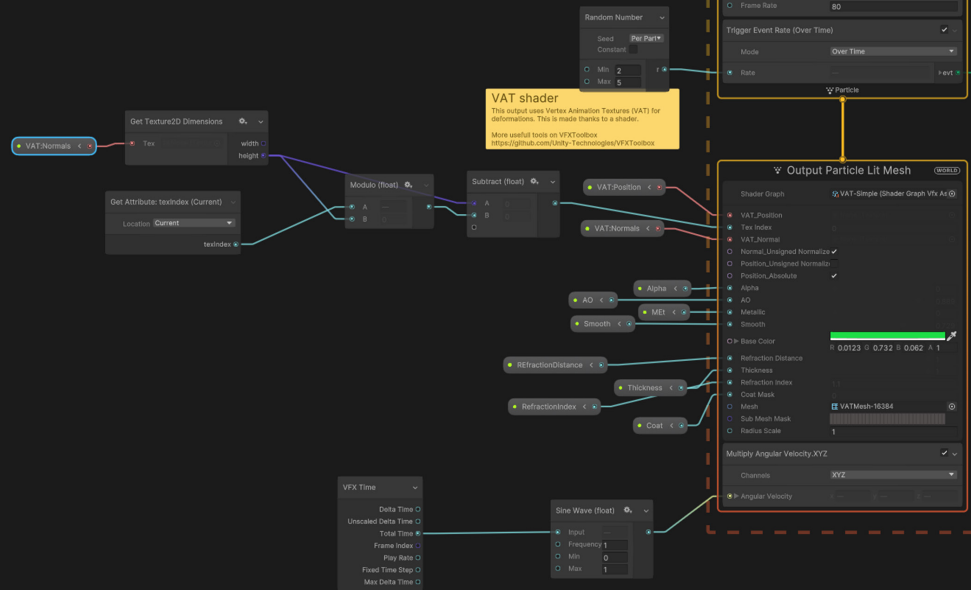


GooBall エフェクト

中央の塊（プロブ）は、流体の動きのような印象を与える **Vertex Animation Texture (VAT) Shader Graph** から始まります。シェーダー内のスクロールするテクスチャシートは、メッシュの 3D ポイントを常に波打たせており、Sci-Fi のネバネバした物体に最適です。

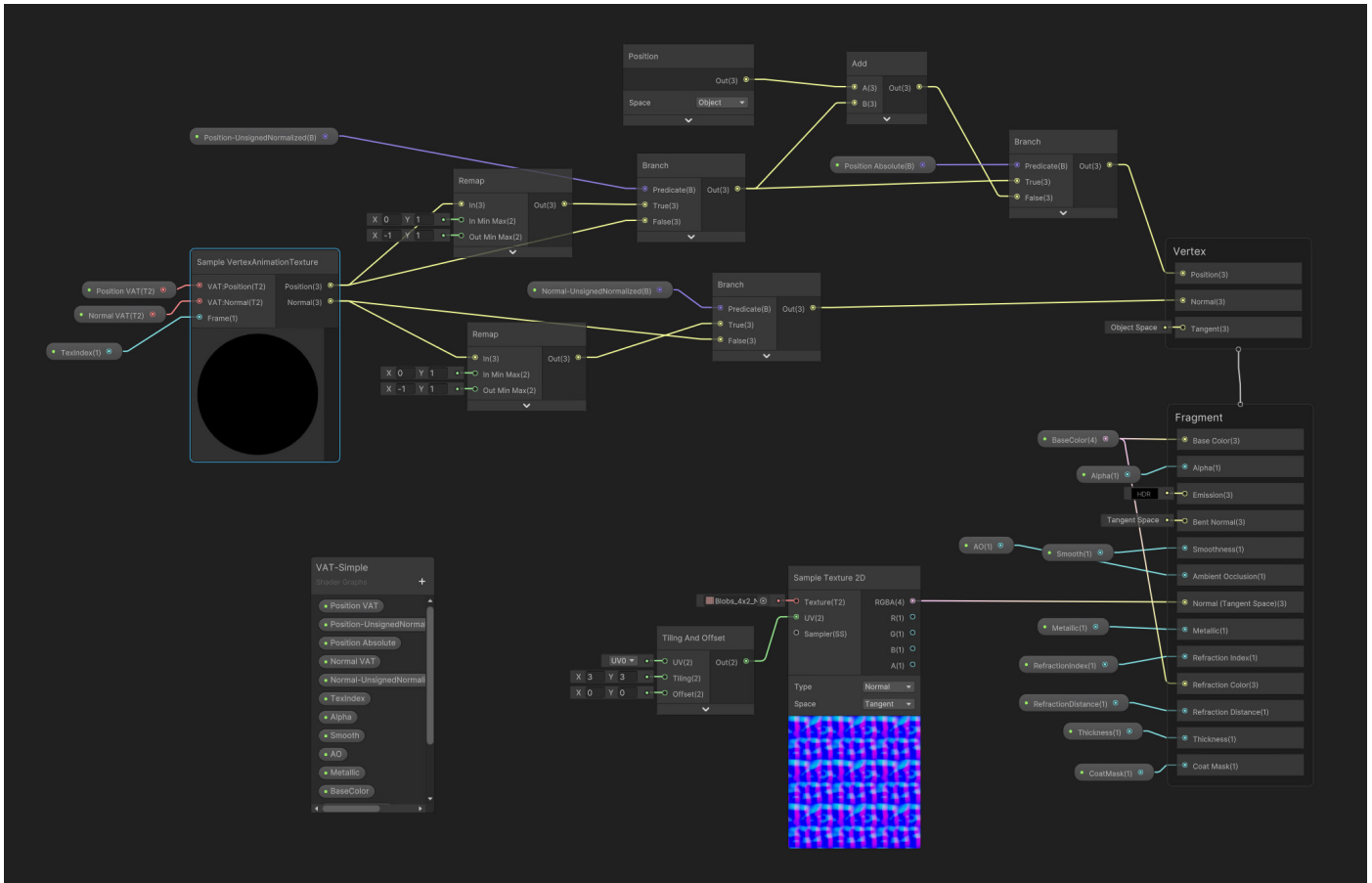


Main blob



メインプロブ

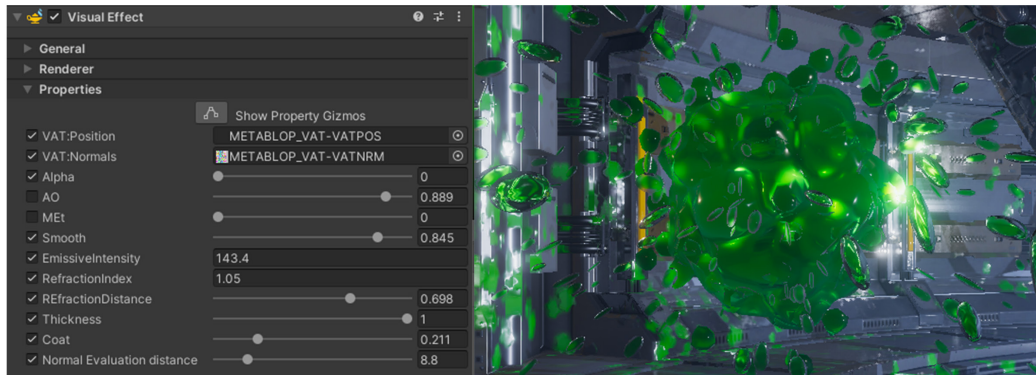
Shader Graph は、**Output Particle Lit Mesh コンテキスト**に直接接続するため、シェーダーに影響を与える入力ポートがいくつかあります。VAT Shader Graph をご覧ください。



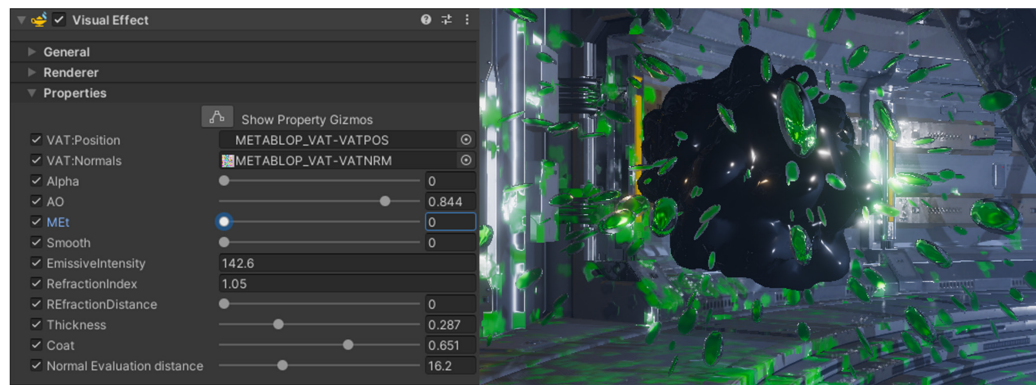
VAT Shader Graph

頂点モーションに加えて、Shader Graph は緑色の透明なガラスのようなマテリアルの外観も作成します。これでネバネバの見た目が変わります。

Shader Graph は VFX Graph と密に連携しています。この例では、VFX Graph の Blackboard から GooBall のシェーディングパラメーターの一部を編集できます。**Inspector** に移動してスライダーとフィールドの統一されたセットにアクセスし、ネバネバの外観を調整します。



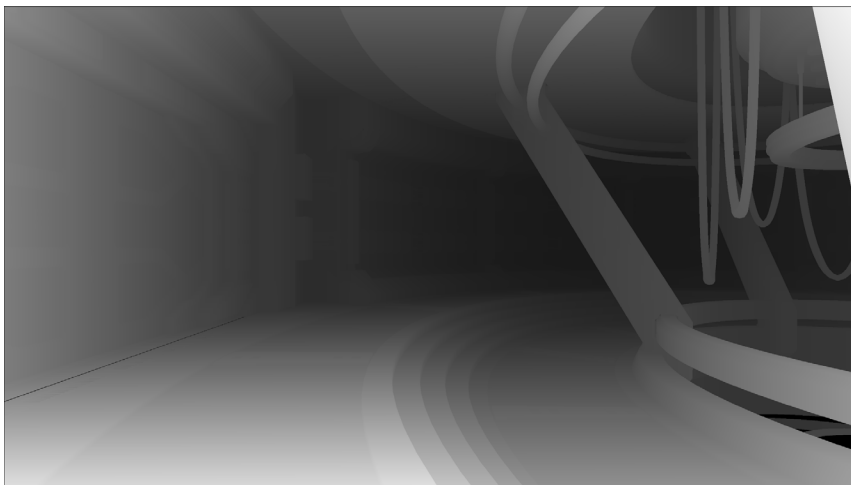
Adjust shader parameters



Blackboard のプロパティを調整してネバネバの外観を変更する

メインのプロップは GPU イベントをトリガーし、スフィアの表面にいくつかのパーティクルをスポーンします。適切なマップを使用すると下向きの力が働き、ネバネバの滴が周期的にポタポタ落ちます。

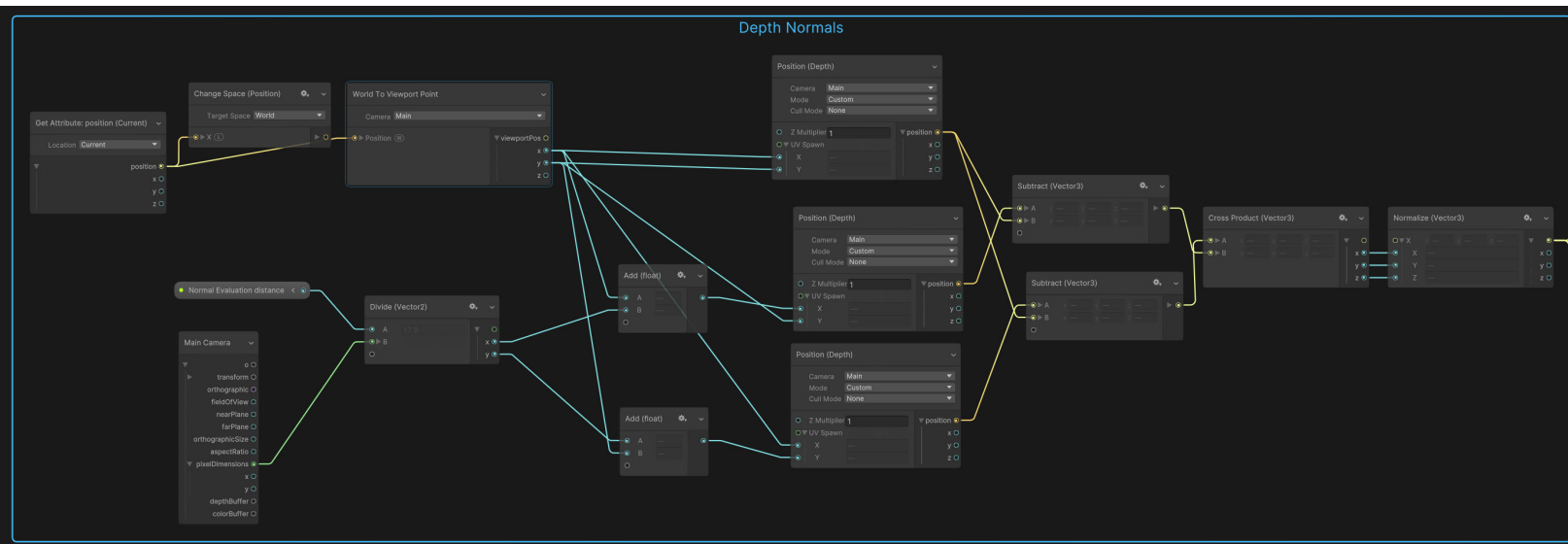
VFX Graph は床の Collider と直接相互作用しませんが、カメラの**深度バッファ**によって環境を近似できます。精度が問題にならない場合に有効です。



カメラの深度バッファの可視化

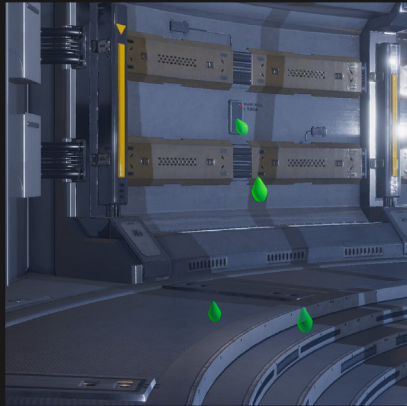
滴が**深度バッファ**に到達すると、GPU イベントがトリガーされます。このイベントは、デカールを処理する別のシステムに、位置、色、サイズの属性を渡します。

ジオメトリに対してスプラットを正しく向けるために、バッファの深度法線は次のようにして計算されます。



深度法線

次に、すべてのデカールが正しい方向を向くように Z 軸属性に渡します。新しい水たまりのテクスチャが作成されたことで、各デカールは段差のある床の表面にリアルに飛び散ります。



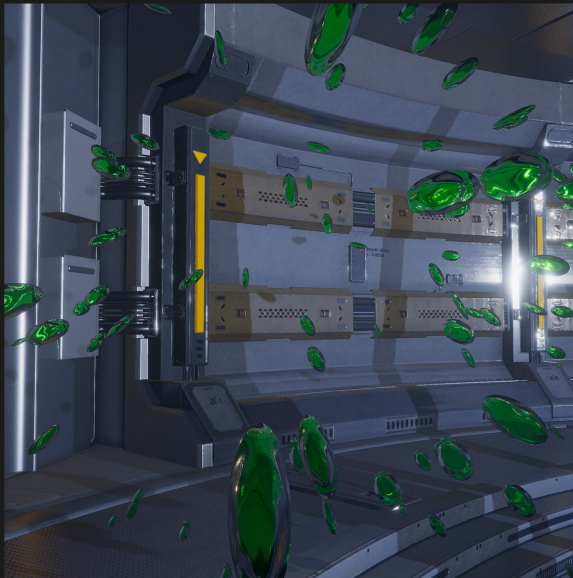
Drops

Drop decals

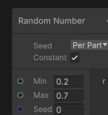
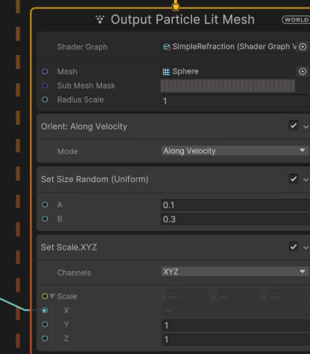
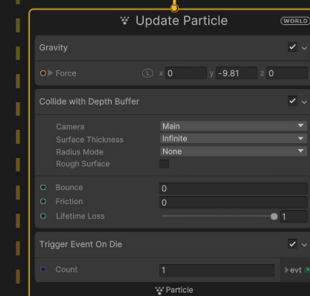
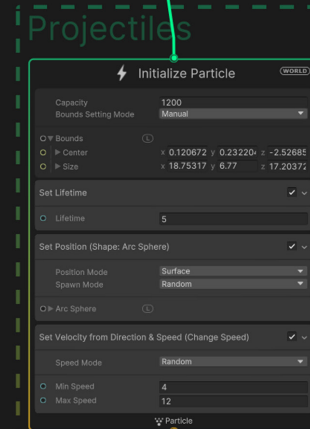
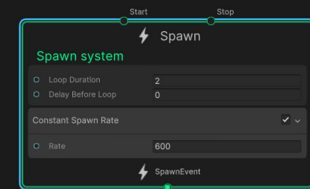
球面的な表面からランダムに飛び散ることによって、シーンはより混沌としたものになります。これは、VFX のパワーを改めて示すものです。

VFX Graph は、複雑なシミュレーションを計算し、フレームバッファを読み出します。ただし、C# へのパーティクルデータの取り込みや基盤となる物理演算システムへの接続はサポートされていません。

- 深度バッファ
- プリミティブ表現 (スフィア、ボックス、円環など)
- 符号付き距離フィールド、Point Cache、ベクトル場などの 3D テクスチャ

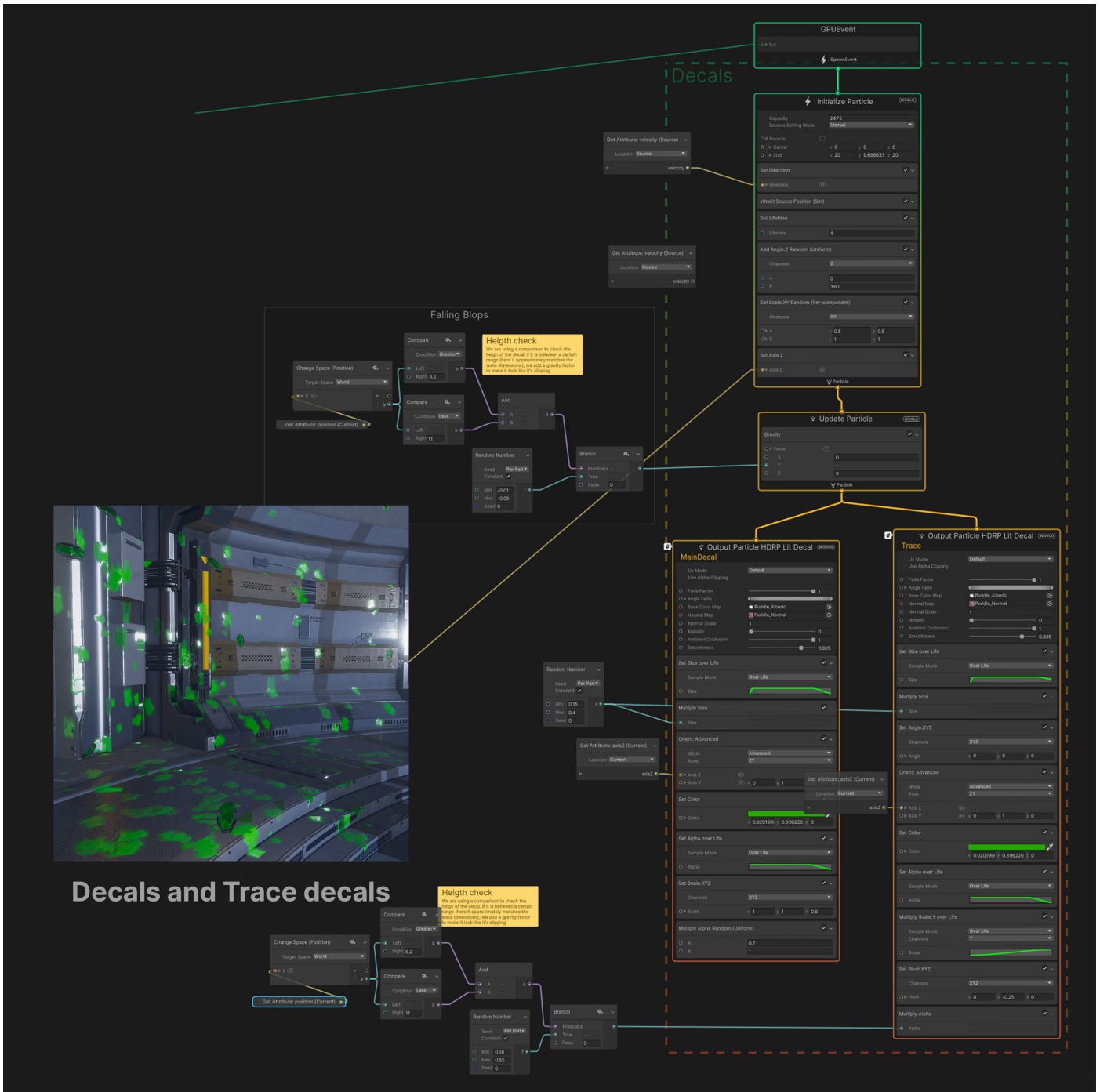


Projectiles



GooBall からカオス的に飛び散る

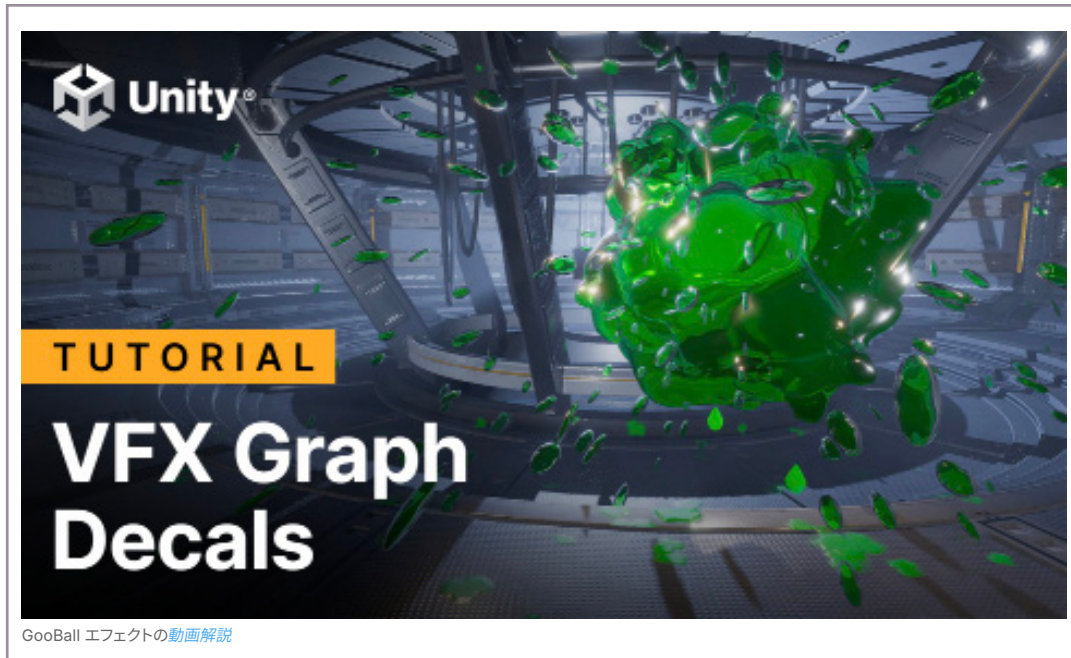
飛び散ったものが部屋のジオメトリに衝突すると、同様の手法でスプラットが計算されます。ただし今回は、2 つ目の Trace デカルが壁の高さの範囲内に収まっているかどうか確認する必要があります。収まっていると、Y 方向の長さがわずかに変わり、金属的で滑らかな表面をスライムが滑り落ちるような錯覚が完成します。



Splat デカルと Trace デカルでエフェクトを完成させる。

これと同じエフェクトの再現を目指してなくても、GooBall シーンは次の方法を示しています。

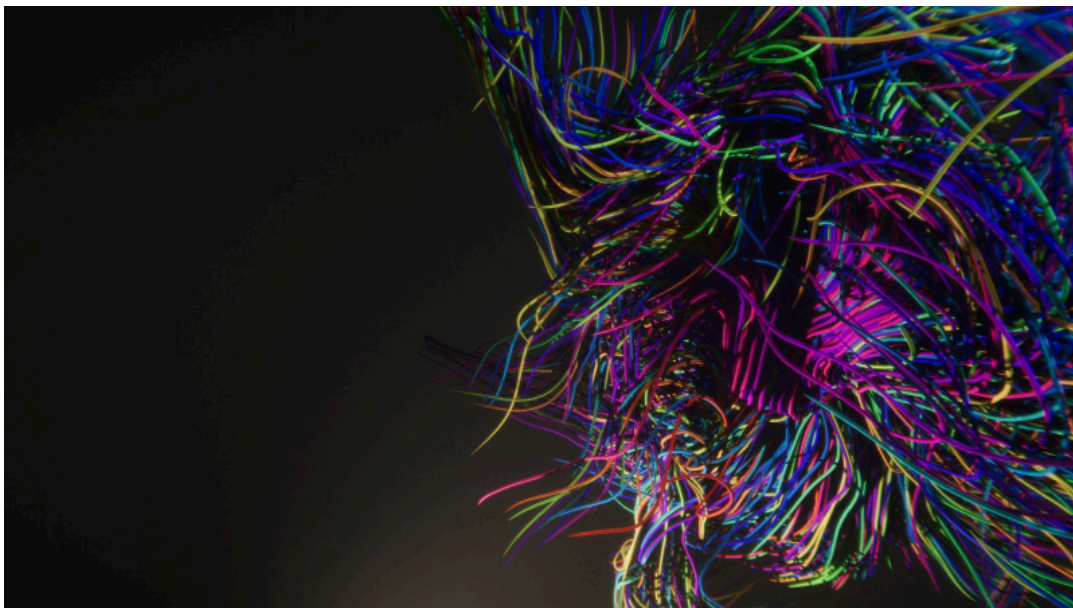
- Shader Graph を Output コンテキストに組み込む
- GPU イベントを使用して同じグラフ内の他のシステムをトリガーする
- 深度法線を使用して環境上にデカルを適用する



リボンパック

この抽象的なエフェクトは、**パーティクルストリップ**、つまりパーティクルのつながりを線やクアッドのストリップとしてレンダリングする方法を示しています。

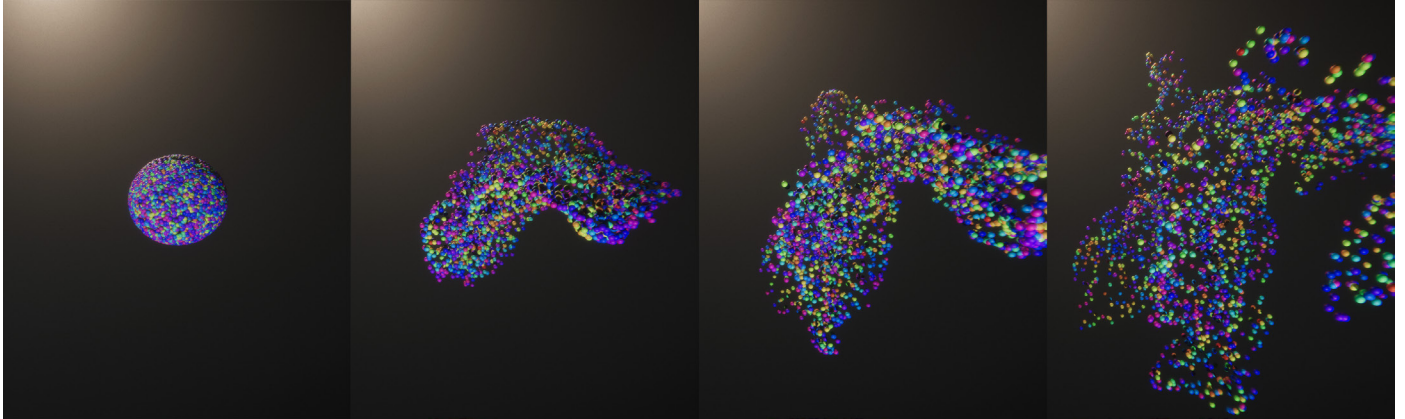
多くの場合、アニメーション化されたトレイルをシミュレートできます。GPU イベントを使用して他のパーティクルでガイドし、トレイルの各点がどのように独立して変化し、風、力、乱流を適用できるかに注目してください。



Ribbon Pack の特徴はパーティクルストリップである。

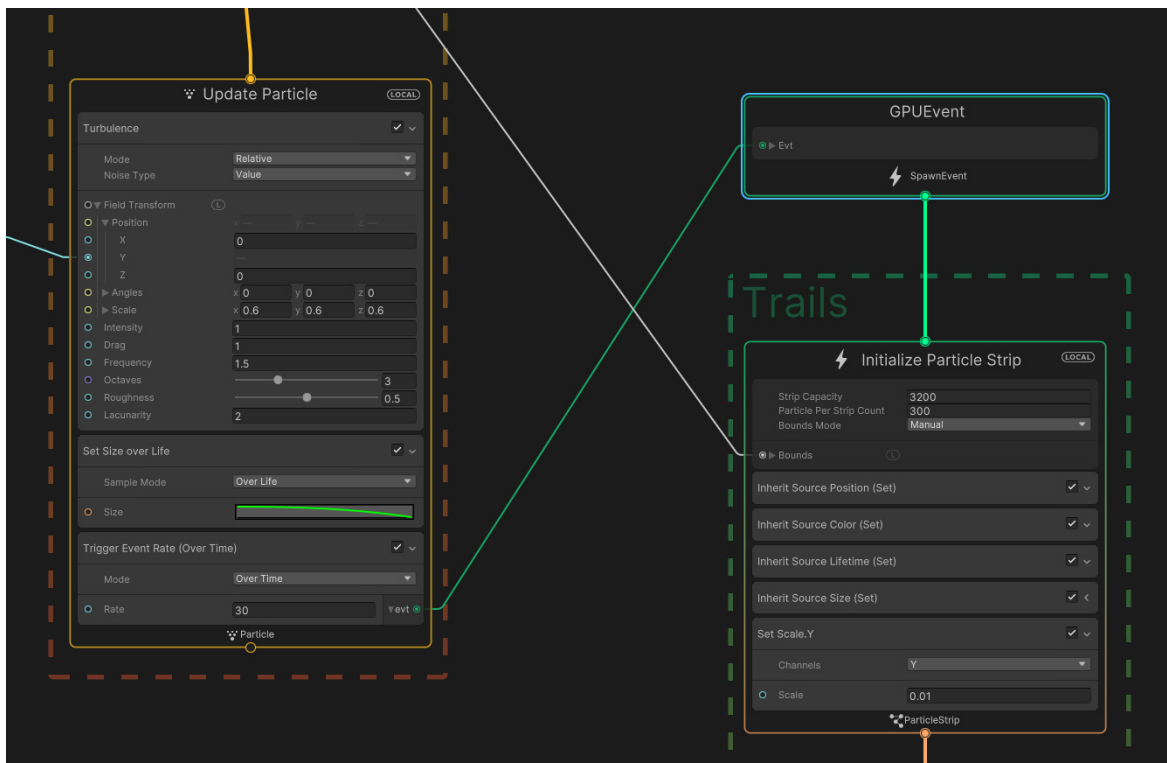
Ribbon Pack グラフは、球面的な円弧の表面に多色のパーティクルをスポーンします。**Noise** を使用して有機的な動きを追加したり、利用可能な**ブロック**を編集して各パーティクルストリップのテクスチャマッピング、スポーン、向きをカスタマイズしたりできます。

このエフェクトを従来メッシュパーティクルでレンダリングすると、次のようになります。



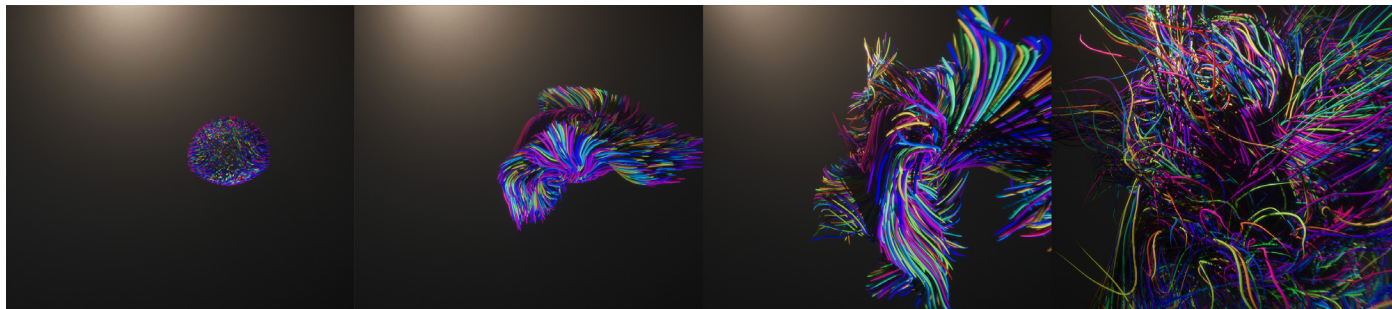
パーティクルストリップなしの Ribbon Pack のレンダリング

代わりに、最初のシステムは非表示になり、画面には表示されません。**Trigger Event Rate** ブロックは、GPU イベントを呼び出すために使用されます。



Trigger Event Rate ブロックは GPU イベントを呼び出す。

これは、パーティクルストリップを初期化、更新、レンダリングする一連のコンテキストにメッセージを送信します。その結果、ケーブルやファイバーが柱状に絡み合います。



多数のリボン

パーティクルストリップにはさまざまな用途があります。例えば、魔法の光条、武器のトレイル、ワイヤーなどを思い浮かべてください。Magic Book のサンプルでは、各ビームの周囲を飛び交うトレイルにパーティクルストリップを使用しています。



Magic Book サンプルのパーティクルストリップのトレイル

それらはまた、次のセクションで説明する、Meteorite サンプルの草の葉でも見られます。

Meteorite サンプル

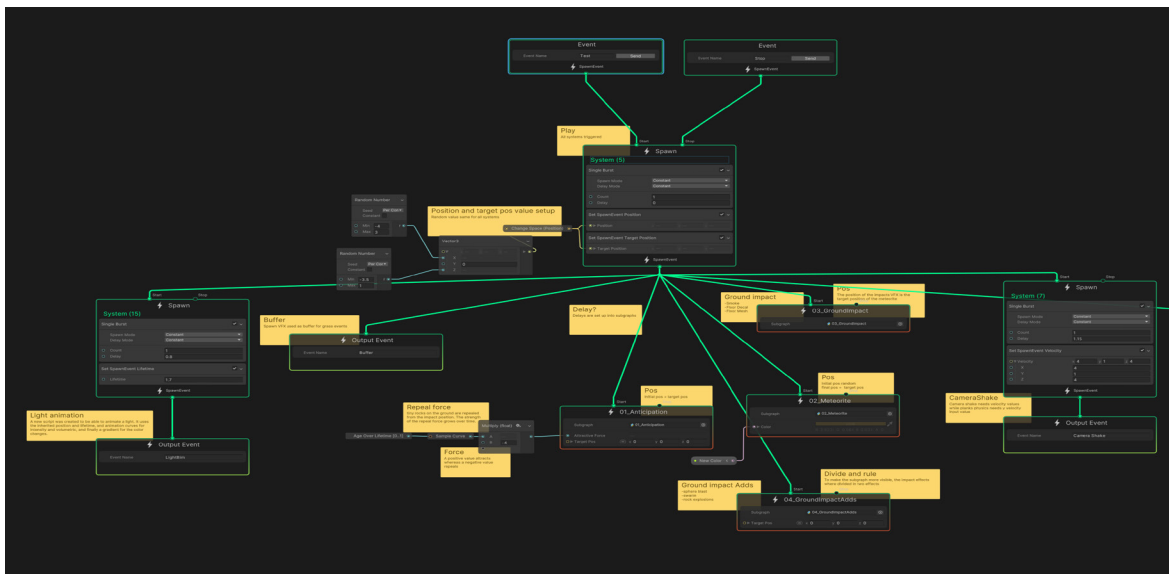
Meteorite サンプルでは、いくつかのエフェクトを組み合わせ、地球に衝突する隕石の衝撃を際立たせます。**Timeline** オブジェクト上の **MeteoriteControl** スクリプトは、キーの押下やマウスのクリックを監視し、隕石エフェクトを有効にします。

このシーンでは、地面と周囲の樹木が爆発に反応します。しかし心配は無用です。このエフェクトの作成により VFX の生き物が傷つけられることはありませんでした。



Meteorite サンプル

ここでは、**MeteoriteMain** という 1 つのグラフが他の複数の要素を駆動します。グラフ自体はきれいに整理されています。



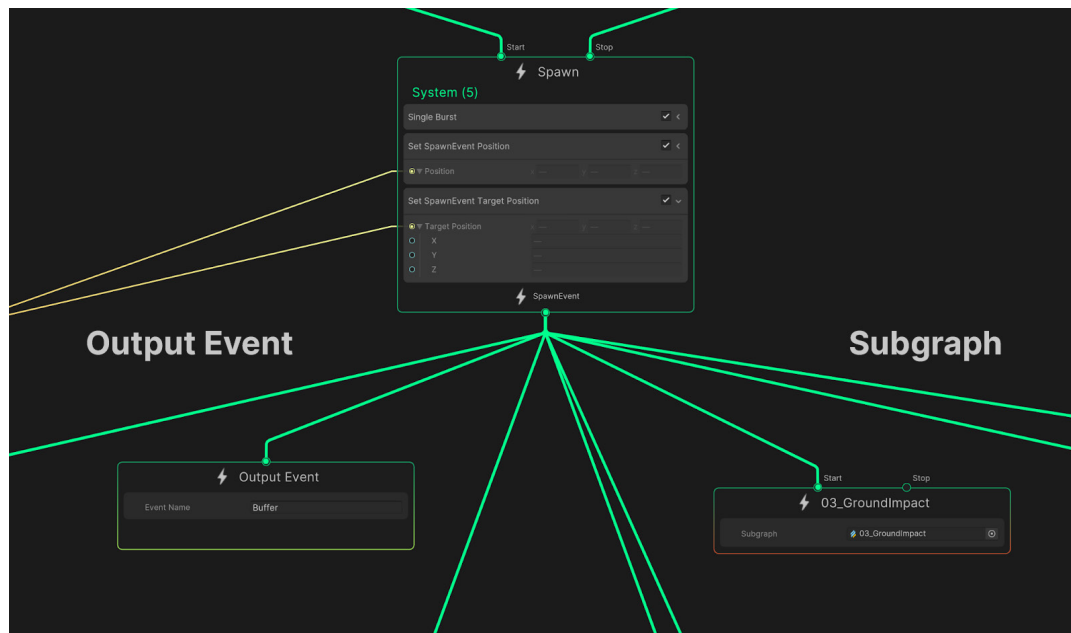
MeteoriteMain グラフ

隕石が空から落ちるたびに、エフェクトの連鎖反応が発生します。これは、他の多くのエフェクトをトリガーする中央の Spawn コンテキストで構成されています。

- 振動する岩と隕石を予感させる光のエフェクト
- 衝突する隕石と軌道の跡
- 衝突により視聴者を驚かすアニメーションライトのバースト
- 地面と相互作用するリジッドボディの“板”とがれき
- 衝撃力をさらに感じさせるカメラの揺れ

注：MeteoriteMain グラフの大部分はサブグラフに分割されているため、比較的すっきりと見えます。Spawn イベントは Start ポートの多くに直接接続します。個々のサブグラフを見ることで、具体的な実装を確認できます。

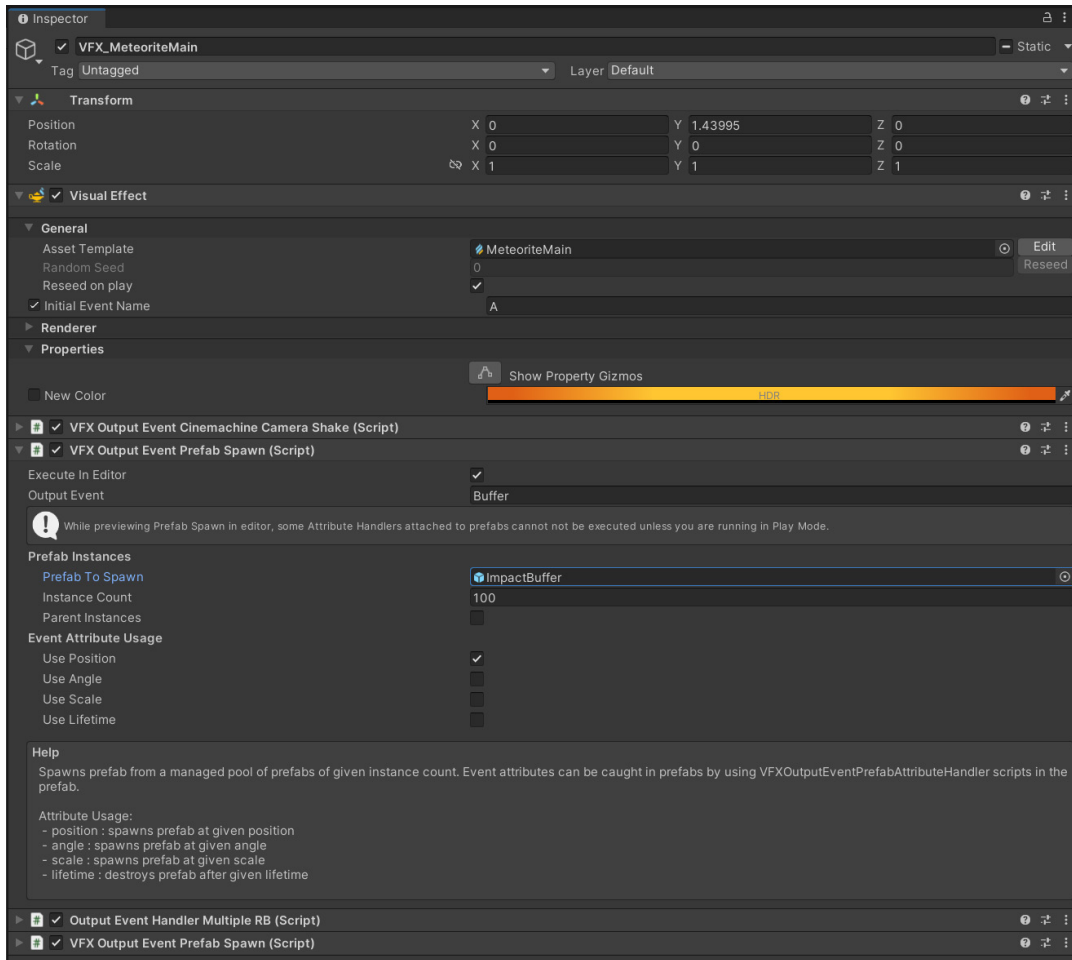
この構造により、多くのディテールが扱いやすい小さなパーツに振り分けられています。グラフの操作が簡単になり、わかりづらい一連のノードで苦勞することがなくなります。



Spawn イベントをサブグラフと Output イベントに接続する。

Output イベントは、グラフ外の他のコンポーネントと通信するために使用します。特に、ライトアニメーション、カメラの揺れ、板の破片には独自の Output イベントがあります。

VFX_MeteoriteMain というゲームオブジェクトを選択すると、これらのイベントを受信して応答するさまざまな Output イベントハンドラースクリプトが表示されます。



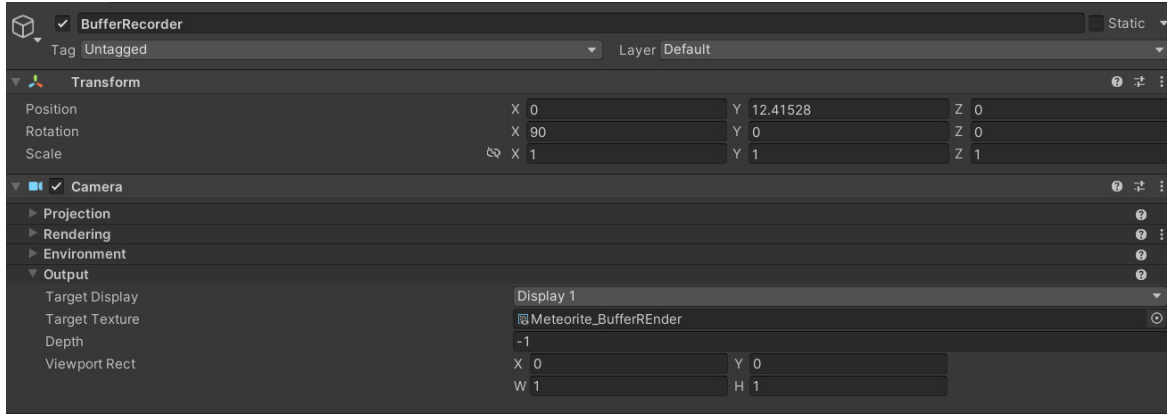
ゲームオブジェクトの Output イベントハンドラー

一方、グラフの **Buffer Output イベント**は、独自の VFX Graph を持つ **ImpactBuffer** というプレハブをスポーンします。地面のデカールを赤、青、緑のカラーチャンネルに分割してアニメーション化します。**ImpactBuffer** エフェクトは次のようになります。



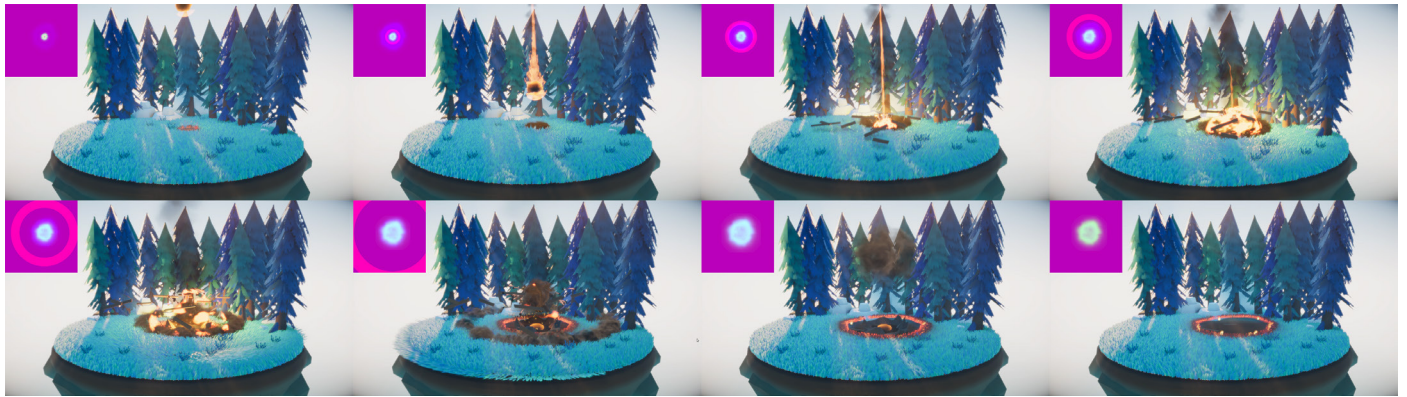
ImpactBuffer プレハブ (赤、青、緑)

必要なのは、それを 2D で記録することだけです。**Buffer Recorder** という別のカメラが真下を見下ろし、**Meteorite_BufferRender** というレンダーテクスチャを生成します。このテクスチャバッファは、**GrassStrip** という別のグラフにデータを送信します。



BufferRecorder

テクスチャは GrassStrip グラフにプラグインされ、パーティクルストリップの動きとレンダリングを駆動します。テクスチャバッファがどのようにエフェクトを生み出しているかをご覧ください。

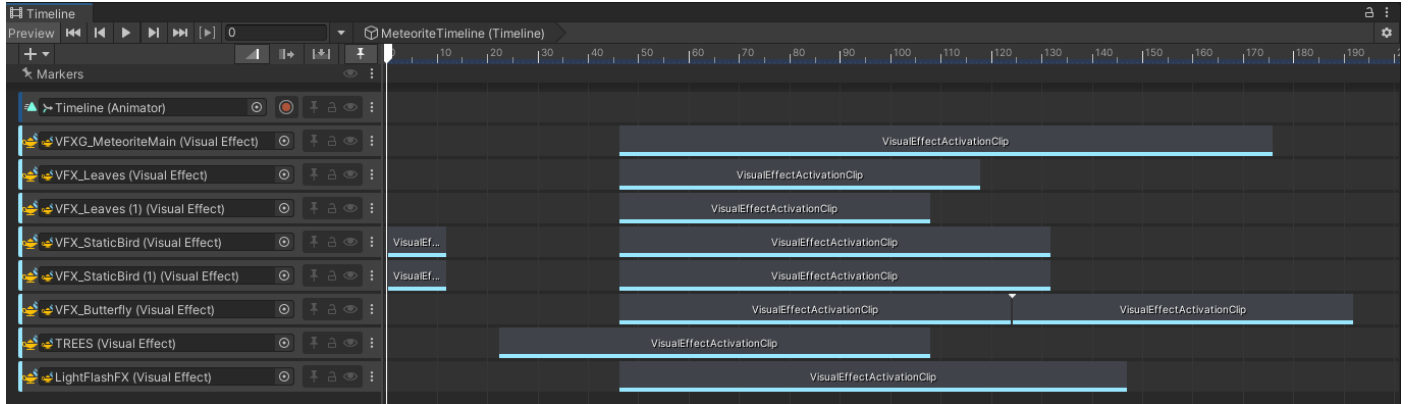


テクスチャバッファが、草のエフェクトを制御している。

メインの隕石以外にも、次のような二次的なエフェクトもあります。

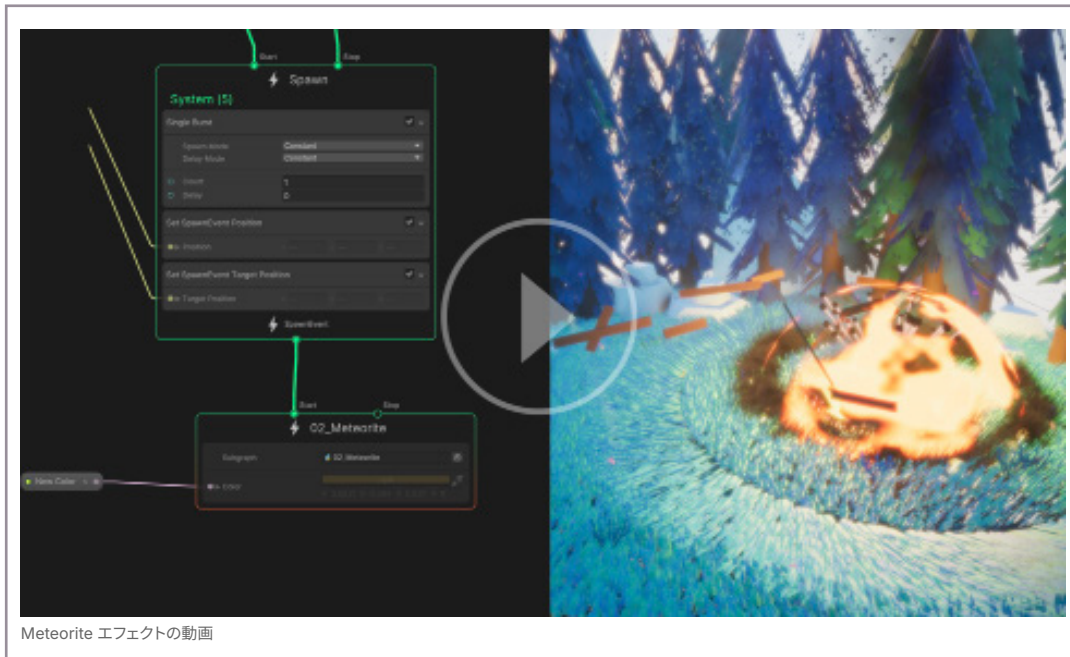
- 木から落ちる葉
- 前景と背景に舞う鳥
- 消えたり現れたりする蝶

これらの二次的なエフェクトは、複数の **Visual Effect Activation Track** を含む **Meteorite Timeline** に依存しています。MeteoriteControl スクリプトが **Playable Director** を開始すると、各二次エフェクトは所定のタイミングに従って再生されます。



Meteorite Timeline

Timeline と Output イベントを設定する方法については、[インタラクティブ性の章](#)を参照してください。

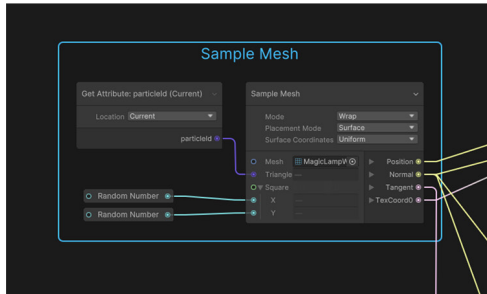


Meteorite エフェクトの動画

メッシュサンプリングエフェクト

メッシュサンプリングは、メッシュからデータを取得し、その結果をグラフで使用できるようにする実験的な手法です。次のいずれかを使用してメッシュをサンプリングします。

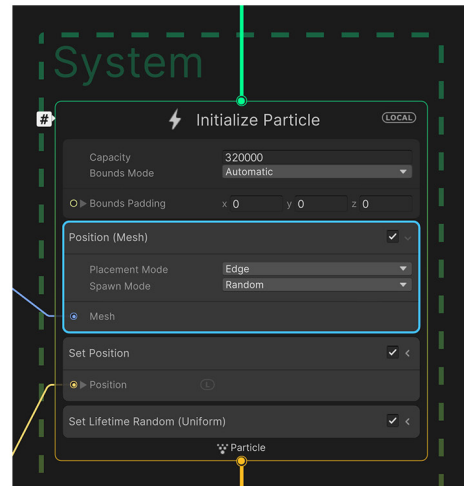
- [Position \(Mesh\) ブロック](#)
- [Sample Mesh オペレーター](#)



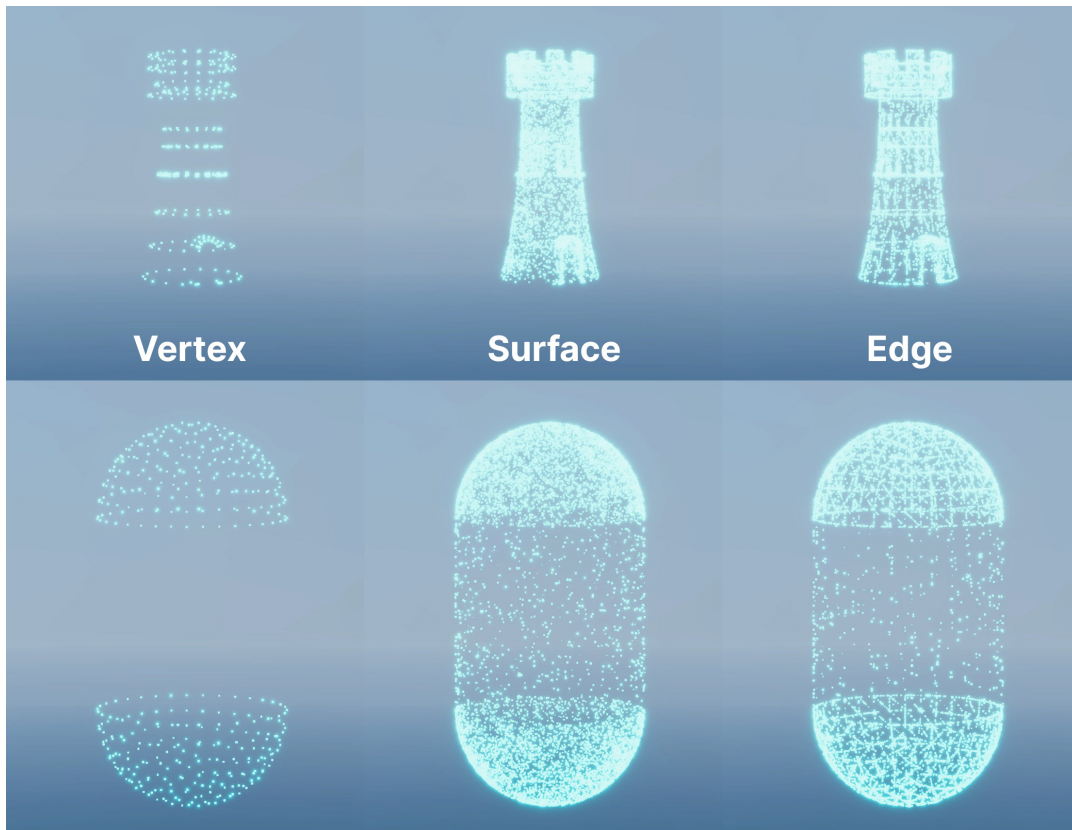
Sample Mesh Operator

Sample Mesh オペレーターと Position (Mesh) ブロック

Placement Mode は、**Vertex** (頂点)、**Edge** (エッジ)、**Surface** (表面) に設定できます。**Position (Mesh) ブロック**がシンプルなメッシュで動作している様子を示します。

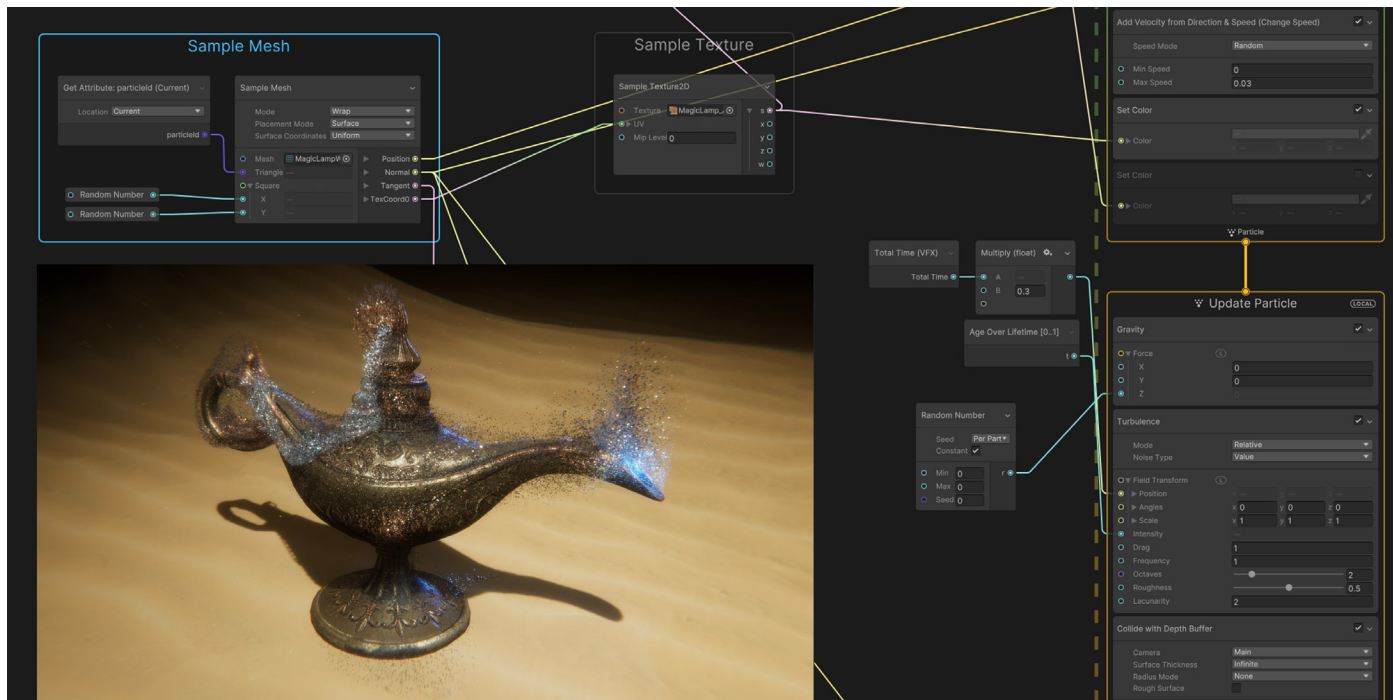


Position (Mesh) Block



Vertex、Surface、Edge のサンプリング

このバリエーションは、サンプルのオリジナルの Magic Lamp シーンを変更します。いくつかの追加作業により、**Sample Mesh オペレーター** を使用してランプ表面のパーティクルを初期化し、パーティクルが浮かんでいるときの向きを変えることができます。Magic Lamp の **Sample Texture** を取得すると、パーティクルの色がメッシュ面に滑らかに統合されます。



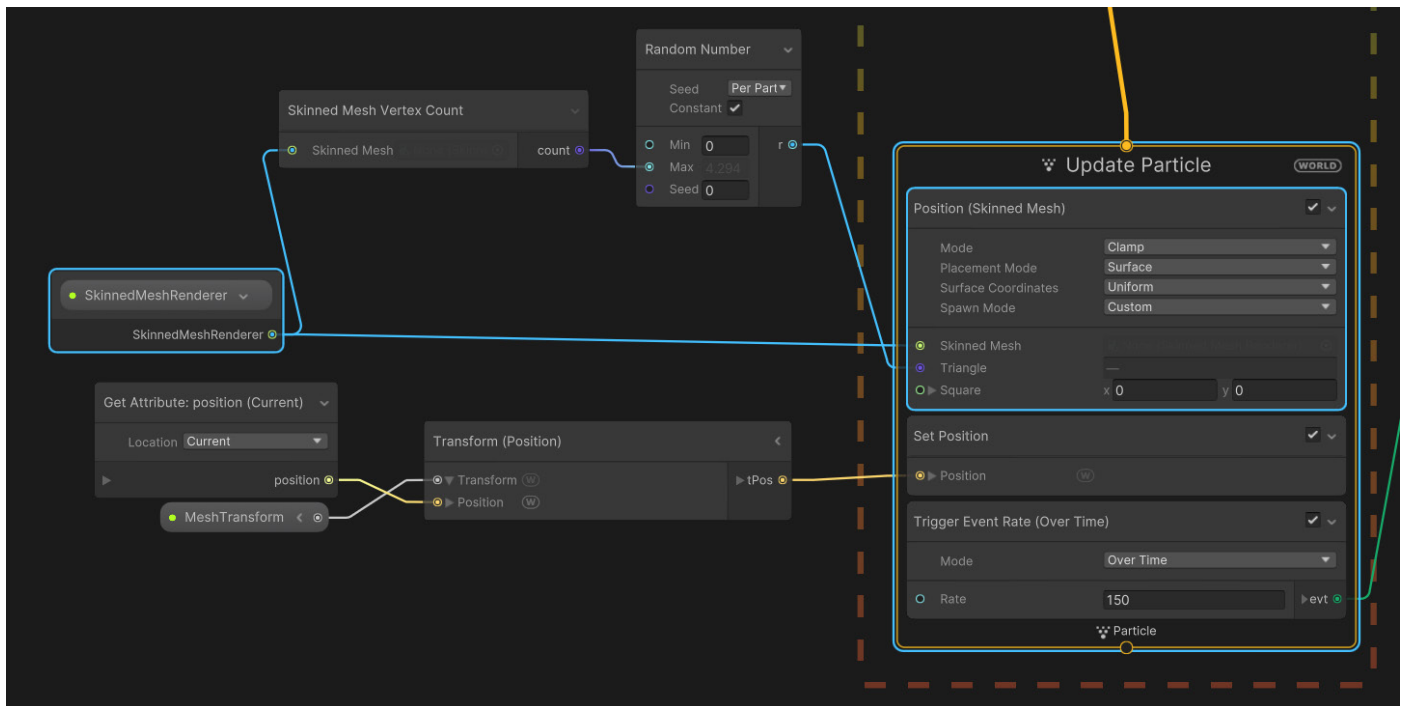
Magic Lamp のサンプルシーン

Skinned Mesh サンプリング

ゲームのキャラクターにエフェクトを適用すると、面白くて目を引く要素をゲームプレイに追加できます。心臓を貫かれて崩れる吸血鬼が必要ですか？それとも炎がよいでしょうか？ [Skinned Mesh Renderer](#) をサンプリングすることで、どちらも可能です。

EllenSkinnedMeshEffects というサンプルエフェクトは、そのタイトルのヒロインのさまざまなエフェクトを示しています。ただし、スキンメッシュデータをサンプリングする一般的なプロセスは、それぞれのケースで似ています。

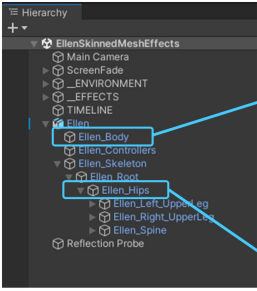
- **Position (Skinned Mesh) ブロック**をコンテキストに追加します (目的のエフェクトに応じて、**Initialize**、**Update**、**Output**)。
- **Blackboard** に **Skinned Mesh Renderer** プロパティを表示し、Skinned Mesh フローポートに接続します。
- **Surface**、**Edge**、**Vertex** データへの追加アクセスが必要な場合は、[Skinned Mesh オペレーター](#) を使用します (使用例については、Hologram グラフと Disintegration グラフを参照してください)。

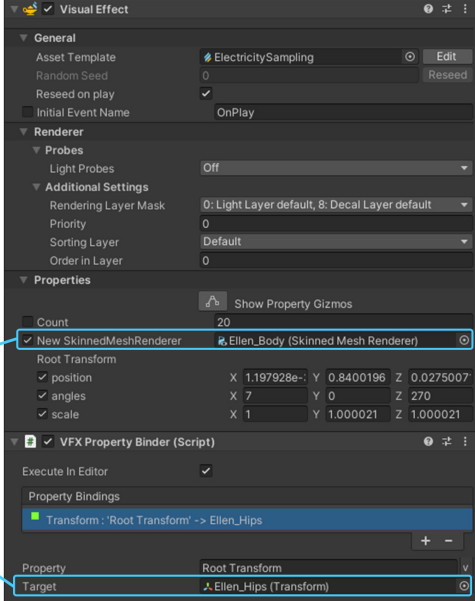


グラフで Position (Skinned Mesh) を設定する

- **Inspector** で、シーンの **Skinned Mesh Renderer** を使用して**プロパティ**を設定します。
- **プロパティバインダー** ([インタラクティビティの章](#)を参照) を使用して、キャラクターのスケルトン内のエフェクトの位置と向きを変換します。この **Transform** とそれに対応する **Set Position ブロック**を、最適な結果を得るために **World** 空間に設定します。

Select from Hierarchy





Skinned Mesh GameObject

Root Transform

Use Properties and Property Binder

プロパティと VFX プロパティバインダーを設定する

また、スケルトン自体の **Base Transform** に対応する **Mesh Transform プロパティバインダー** も追加する必要があります。¹これはキャラクターによって異なりますが、この場合はキャラクターの股関節をつなぎます。これで、パーティクルを Skinned Mesh Renderer に追従させることができます。



EllenSkinnedMeshEffects の例

¹ Unity 2021 LTS はこちらをご覧ください。Unity 2022.2 以降では、この手順は必要ありません。



キャラクターやオブジェクトにエフェクトを作成するときに、Skinned Mesh サンプリングの汎用性を活用しましょう。サンプルでは、Ellen のキャラクターが登場します。

- Sci-Fi ホログラフィック投影として
- 煙と火の粉を伴った炎
- 濡れて、肌から滴り落ちる水滴
- 色のついた軌跡を残す猛ダッシュ
- 電気ショック
- 灰への変化

これらはそれぞれ、Skinned Mesh Renderer をサンプリングした個別のグラフです。それぞれの実装の詳細を調べると、パーティクルとキャラクターメッシュの相互作用を確認できます。

その他の例

VFX Graph Sample により、さらに多くの可能性を探ることができます。プロジェクトの他のシーンもぜひご覧ください。それぞれ、特定のビジュアルエフェクトを作成するための異なる手法が使用されています。

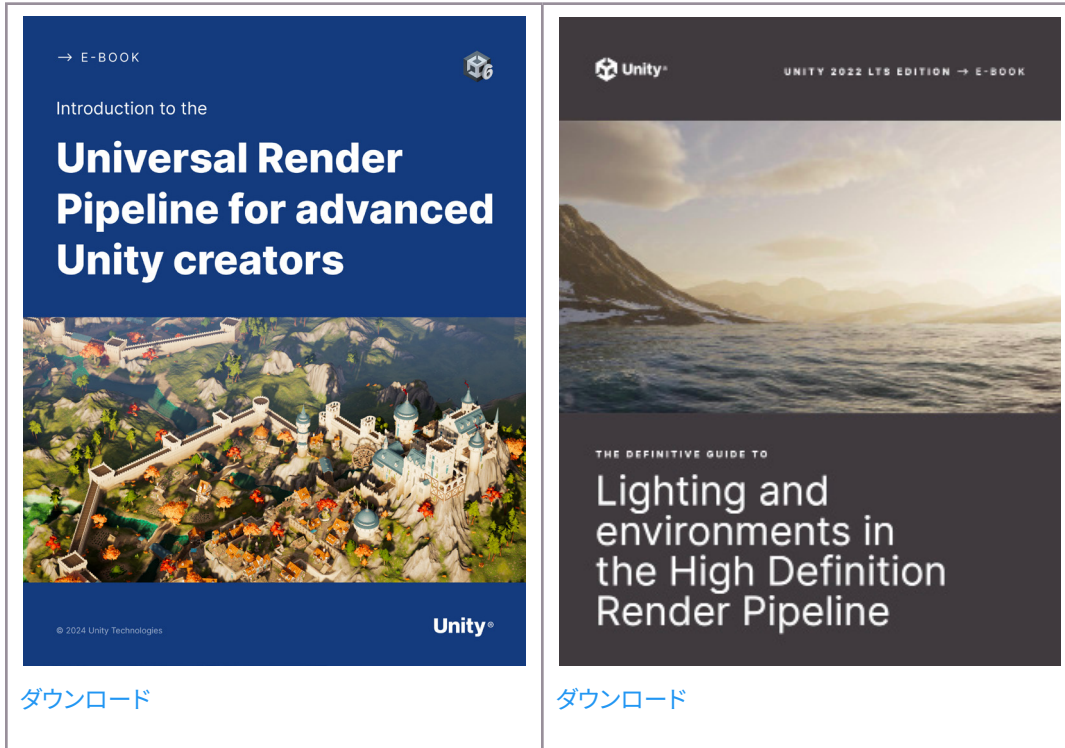
レンダーパイプラインと VFX Graph

選択したレンダーパイプラインは、VFX Graph で利用可能な出力コンテキストに影響を与えます。デザインのニーズとターゲットプラットフォームによって、アプリケーションに最適なレンダーパイプラインが最終的に決まります。

ユニバーサルレンダーパイプライン (URP) は、ローエンドのモバイルからハイエンドのコンソールや PC まで、幅広いデバイスのパフォーマンスに合わせて最適化されています。簡素化されたライティングとシングルパスフォワードレンダリングにより、合理的な機能セットを提供します。URP は VFX Graph をサポートしていますが、一部の高度な機能が制限または利用できない場合があります。

HD レンダーパイプライン (HDRP) は、ハイエンドの PC とコンソール向けに最適化されており、ディファードレンダリングパスを使用して複雑なライティングとシェーディングを処理します。HDRP には、レイトレーシング、ボリュメトリックライティング、サブサーフェススキャタリング、スクリーンスペースリフレクションなどの高度な機能が備わっています。ただし、HDRP はパフォーマンスのオーバーヘッドが高く、プラットフォームのサポートも限られているため、ローエンドデバイスやモバイルプラットフォームには適していません。

レンダーパイプラインのすべての機能の比較は、[こちら](#)で確認できます。



デバイスサポート

VFX Graph では、デバイスの互換性のためにコンピュートシェーダーが必要です。モバイルデバイスでのコンピュートシェーダーのサポートは、ブランド、モバイル GPU アーキテクチャ、オペレーティングシステムによって大きく異なります。プラットフォームがコンピュートシェーダーをサポートしていない場合は、Unity の[ビルトインパーティクルシステム](#)をお勧めします。

レンダーパイプラインの互換性の詳細については、[Unity VFX Graph システム要件](#)を参照してください。

URP 用の VFX Graph

VFX Graph は、コンピュートシェーダーに対応したデスクトップ、モバイル、XR プラットフォーム向けのユニバーサルレンダーパイプライン (URP) をサポートしています。注：URP では、Visual Effect Graph は[ガンマ 色 空間](#)をサポートしません。

Unity 6 では、以前は HDRP でのみ利用可能だった機能のいくつか、URP に加えられています。

Lit 出力

VFX Graph が URP で Lit 出力に対応するようになりました。これらを使用して、シーンのライティングに直接反応するエフェクトを作成できます。



URP に Lit 出力が追加された。

VFX Graph で URP デカルをスポンできるように、Shader Graph を使用して HDRP と URP の両方のデカルをカスタマイズできます。

2D Renderer と Shader Graph のサポート

VFX Graph は、Shader Graph の 2D サブターゲットをサポートし、パーティクルをスプライトとしてレンダリングするために使用できます。新しいサブターゲットを見つけるには、**Create > Shader Graph > URP** に移動します。この機能はユニバーサルレンダーパイプライン専用です。



URP 2D Renderer による VFX Graph Rain エフェクト

デカール

VFX Graph でデカールをスポーンし、HDRP でも URP でも、Shader Graph を使用してカスタマイズできます。これを使用して、弾丸の衝撃、足跡、表面の損傷、その他環境に動的なエフェクトを追加します。

6 ウェイスモークライティング

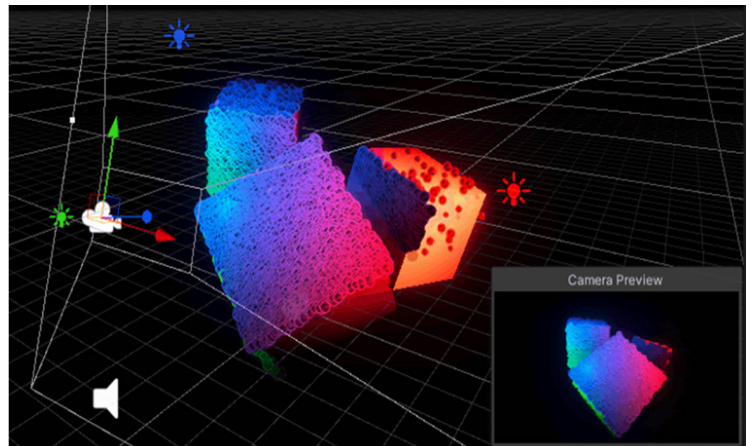
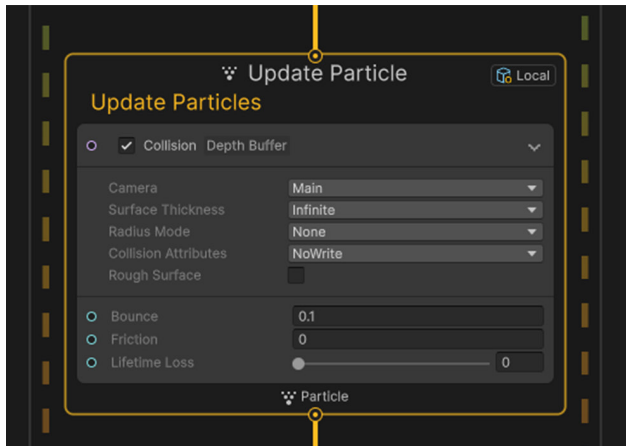
リアルなスモークエフェクトを実現する 6 ウェイライティングは、SideFX の Houdini、Blender、JangaFX の EmberGen などのツールのライトマップを使用してカスタマイズできます。このライティング手法により、高い計算コストをかけることなく、動的なリアルタイムライティングで煙のパーティクルをボリュームトリックスモークに近似できます。6 ウェイライティングは、炎や爆発などの煙のようなエフェクトに使用できます。シーンライティングは、パーティクルシミュレーションを環境に統合するのに役立ちます。

モーションベクトル

VFX Graph のパーティクルは URP でモーションベクトルを生成でき、Temporal Anti-Aliasing (TAA) や Motion Blur などのエフェクトに役立ちます。URP は不透明なパーティクルのみをサポートしますが、HDRP では透明と不透明の両方を使用できます。

カメラバッファ

URP カメラバッファをサンプリングして、シーンの深度と色を取得できるようになりました。この機能を使用すると、GPU 上で高速な衝突判定を実行したり、深度バッファに対してパーティクルをスポーンしたり、シーンの色を継承したりできます。例えば、VFX によるインパクトやスプラット、パーティクルに溶け込むキャラクター、土台となるオブジェクトに応じて色が変化するエフェクトを作成できます。



カメラバッファは GPU 上で衝突判定を実行できる。



HDRP 用の VFX Graph

レイトレーシングやボリュメトリックレンダリングなど、一部の VFX Graph 機能は HDRP でのみ使用できます。アプリケーションでハイエンドなグラフィックスが必要な場合は、以下の HDRP 専用機能を検討してください。

レイトレーシングのサポート

HD レンダーパイプラインで、以下のレイトレーシング機能を持つ VFX Graph パーティクルを使用します。

- **レイトレーシングによるリフレクション**：これは、[Screen Space Reflection](#) の代わりに、より正確なリフレクションのためにオフスクリーンデータを使用できます。
- **レイトレーシングによるシャドウ**：レイトレーシングは、従来のシャドウマップに代わって、より正確でリアルなシャドウを作成できます。
- **レイトレーシングによるアンビエントオクルージョン**：これは HDRP のスクリーンスペースアンビエントオクルージョンに代わるもので、画面外のデータを使用できる、より正確なレイトレーシングソリューションです。
- **レイトレーシングによるグローバルイルミネーション**：これは間接光をより正確に表し、表面で反射する光を考慮しています。

RTX サポートにより、レイトレーシングパスで VFX をレンダリングする機能が追加され、レイトレーシングによるリフレクションで VFX を考慮できるようになりました。



RTX サポートにより、レイトレーシングパスで VFX Graph をレンダリングする機能が追加された。

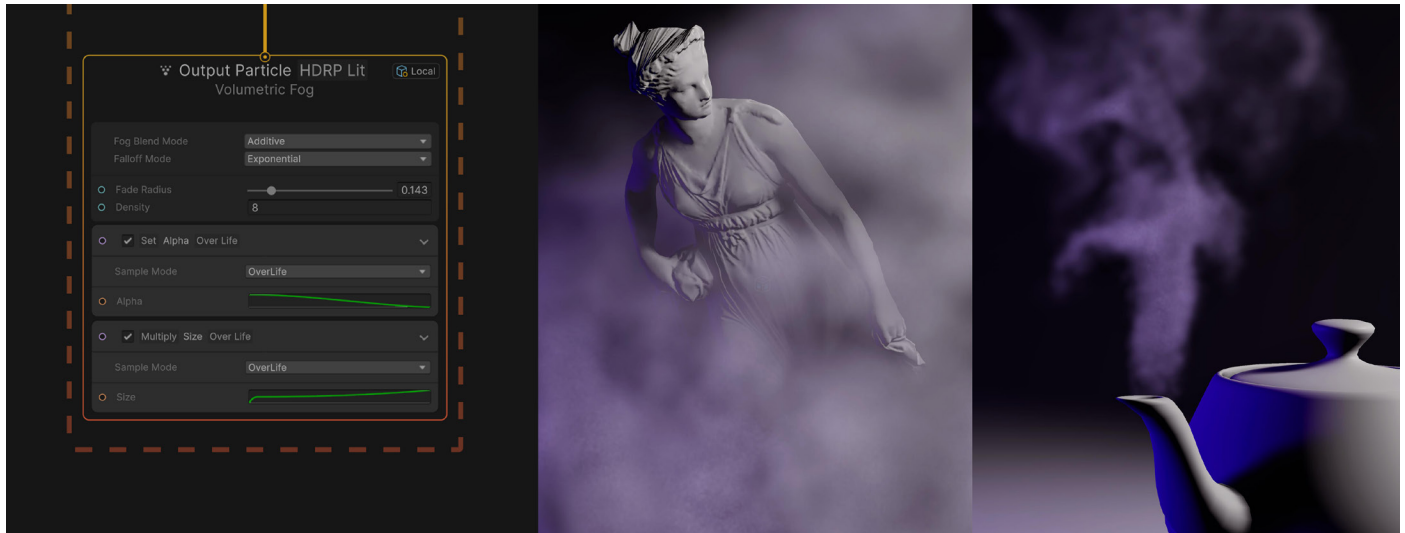
クアッド (四角形)、三角形、八角形を使用する VFX Graph でレイトレーシングを有効にできます。VFX Graph は、メッシュやストリップを使用したレイトレーシングをサポートしていません。

HDRP シーンでレイトレーシングを使用するには、[レイトレーシングの使用を開始する](#)を参照してください。



ボリュメトリックフォグの出力

Output Particle HDRP Lit Volumetric Fog 出力ノードは、HDRP で **Local Volumetric Fog** をサンプリングします。これを使用して、動的な雲、煙、炎のエフェクトをシミュレートしたり、ボリュメトリックフォグをより動的にしたりできます。



HDRP はボリュメトリックフォグ (Volumetric Fog) の出力をサポートしている。

VFX Graph との互換性の詳細については、Unity [Graphics product roadmap](#) (グラフィックス製品ロードマップ) をご参照ください。

Shader Graph の統合

ビジュアルエフェクトは現実世界の法則に逆らうことが多く、標準の URP シェーダーや HDRP シェーダーが提供するものを越えた、独自のシェーディングとレンダリングが必要になります。Sci-Fi フォースフィールドや魔法のオーラは現実世界には存在しないので、あらかじめ用意されているシェーダーでは十分表現できない可能性があります。このようなシナリオでは、[Shader Graph](#) を使用してシェーダーをカスタマイズできます。

Shader Graph 統合の主な利点の 1 つは、シェーダーの動作をパーティクルレベルで制御できることです。これにより、パーティクルごとに異なる値を使用してバリエーションや色のランダム化などの動的なエフェクトを作成でき、非常に複雑なビジュアルを実現できます。

ビルトイン出力

カスタムシェーダーについて詳しく見る前に、Unity にはさまざまなレンダーパイプライン用に最適化された、次のようなさまざまな Output コンテキストが組み込まれていることを理解してください。

- **HDRP/URP Lit 出力** : この出力は、リアルなライティングとシェーディングによるパーティクルのレンダリング用に最適化されています。Lit 出力は、ビジュアルエフェクトがシーン内のライティングに反応する必要があるシナリオで便利です。
- **Unlit 出力** : この出力は、シンプルな非点灯エフェクト用に設計されており、シーンのライティングとは相互作用しません。ライティング計算が不要な定型化されたエフェクトや 2D のようなエフェクトに使用します。
- **Six-Way Smoke Lit** : これは HDRP Lit および URP Lit 出力の Material Type オプションです。この出力を使用すると、[6 ウェイライティング手法](#)で、ボリュメトリックのようなリアルなシェーディングを実現できます。

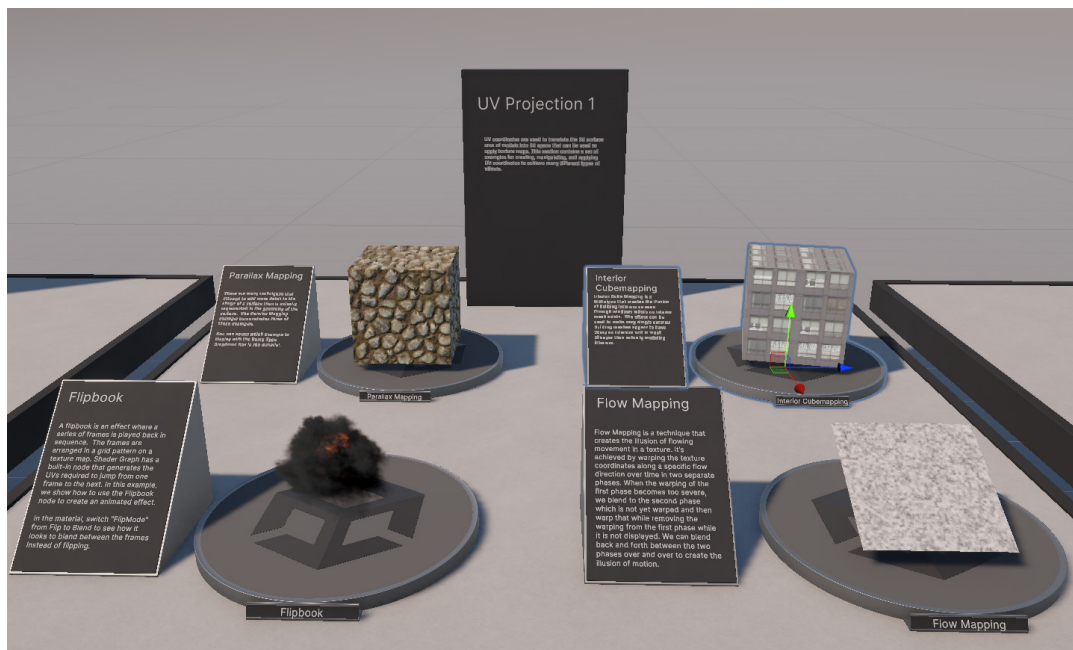
- **HDRP Distortion 出力** : この HDRP 出力は、背景を歪めるエフェクトを生み出し、かげろう、さざ波、ガラスの屈折などの現象をシミュレートします。
- **Decals 出力** : この URP および HDRP 向けの出力は、シーン内のサーフェスへのテクスチャの投影に使用され、弾痕、落書き、表面の摩耗などのディテールを既存のジオメトリに直接追加できます。

これらの出力には、フレームブレンディング、UV フリップブック、エミッシブプロパティなどの機能を含む、統合されたシェーダー機能が備わっています。これにより、カスタムシェーダーをゼロから構築することなく、洗練されたエフェクトを作成できます。

シェーダーエフェクト

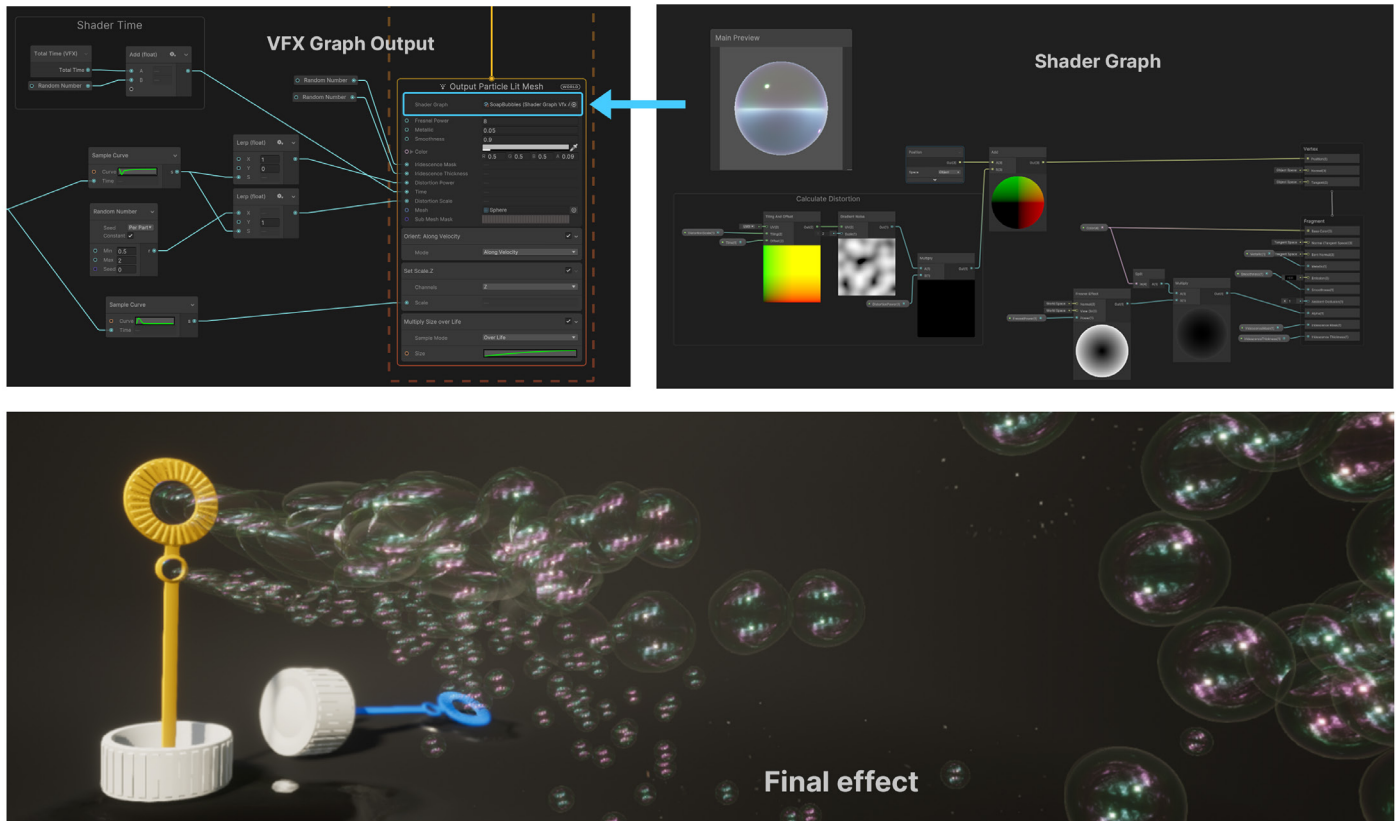
Shader Graph を使用すると、テクニカルアーティストはグラフネットワークを使用してカスタムシェーディングを構築できます。Shader Graph とシェーダーオーサリングはそれぞれが独立したトピックですが、シェーダーに関する一定の知識があれば、VFX Graph の使用を補完できます。シェーダーを使用すると、光と色を操作してエフェクトを強化できます。

Shader Graph を使用すると、UV を歪めたり、アニメーション化したり、サーフェスの外観をプロシージャルに変更したりできます。シェーダーは、フルスクリーンのイメージフィルターのように機能しますし、ワールドの位置、法線、カメラからの距離などに基づいてオブジェクトの表面を変更するのも便利です。Shader Graph のビジュアルインターフェースは、リアルタイムのフィードバックによりイテレーションをスピードアップします。



シェーダーは UV を使用して表面の外観を変更できる。

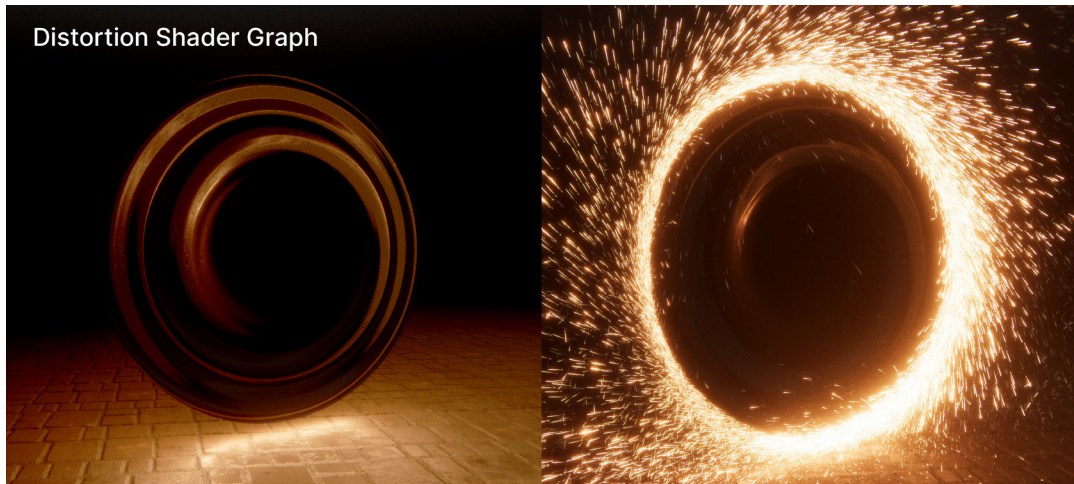
シェーダーは、揺らめく炎や、水中を移動するオブジェクトに反応する水などの動的なエフェクトを作成するために使用できます。雲、煙、流体などの自然現象の場合、シェーダーは複雑で反復しないパターンを作成し、奥行きとバリエーションを加えるのに役立ちます。



Shader Graph と HDRP の統合

シェーダーが VFX Graph を補完する他の方法をいくつか紹介します。

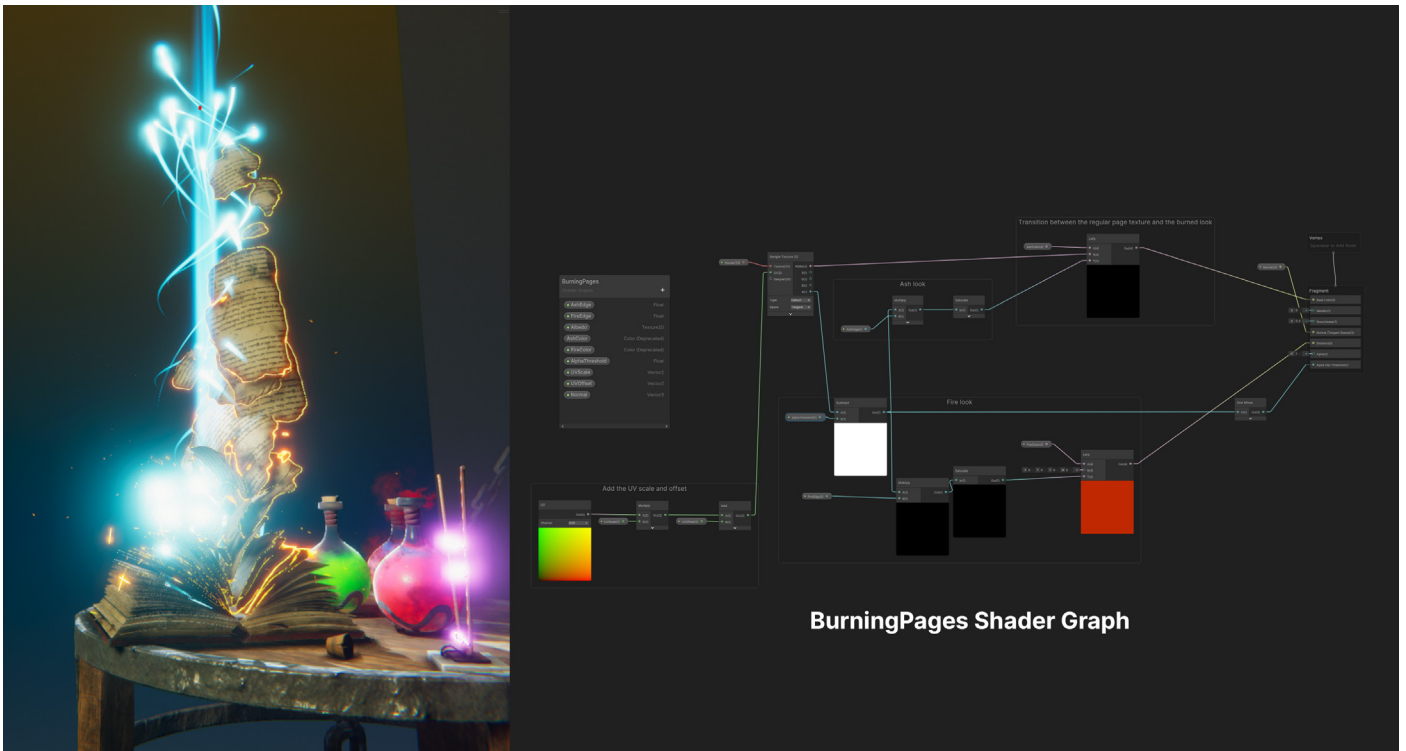
- **ディストーション (歪み) のエフェクト** : シェーダーはピクセルを操作して光の屈折をシミュレートできます。ディストーションシェーダーを使用して、たき火やジェットエンジンなどの熱源に波状のエフェクトを作成します。歪みは、水やその他の液体のさざ波の見た目を模倣することもできます。
- 例えば、[Visual Effect Graph Samples](#) の Portal エフェクトは、Shader Graph を使用してポータル内部の中央を歪めます。



シェーディングは光の屈折をシミュレートできる。

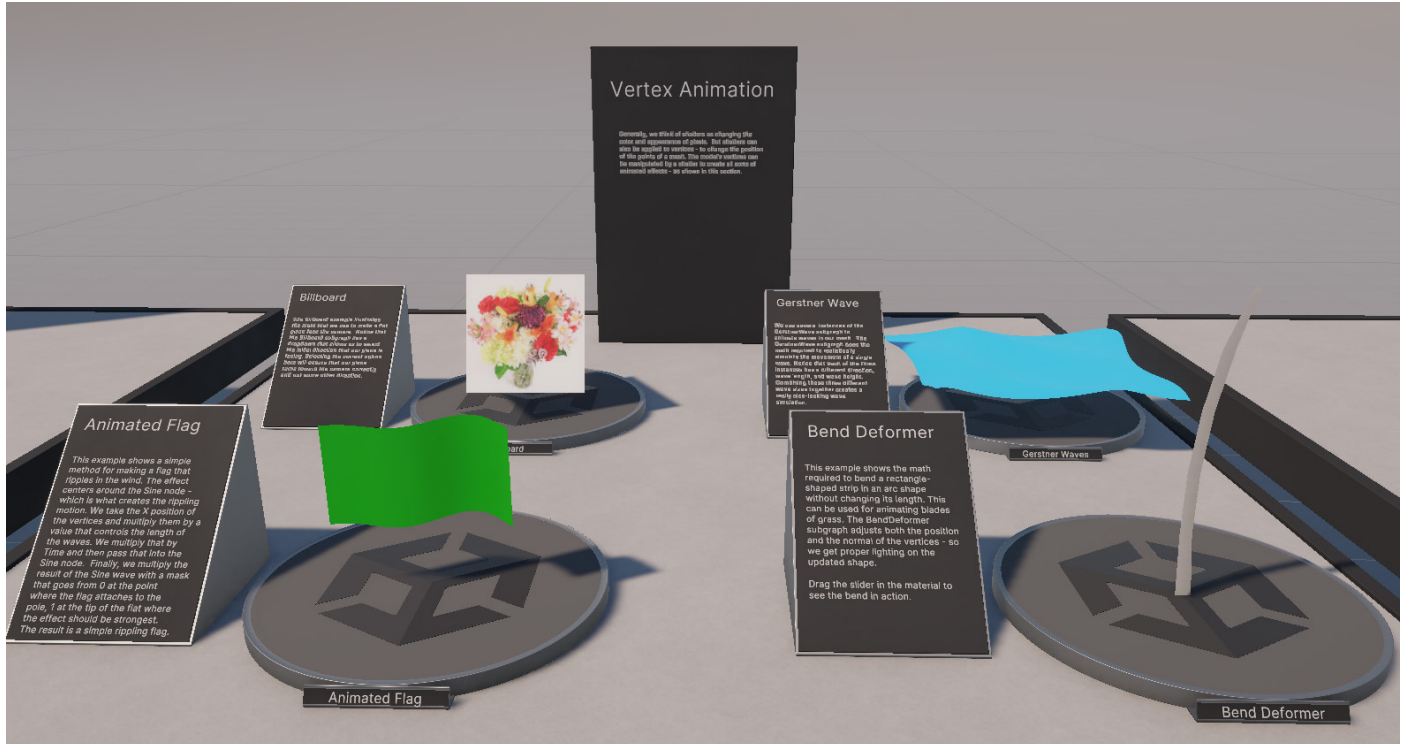
- **アルファ浸食**：オブジェクトやテクスチャをディゾルブさせることは、テクスチャを不均一にフェードインやフェードアウトさせる一般的な方法であり、何かが煙や灰の中に消えていったり、隠れた構造が現れたりするエフェクトを作成できます。テレポーテーションやホログラムなど、分解するものを考えてみてください。

例えば、Magic Book サンプルでは、特殊なシェーダーを使用して、飛んでいるページが燃えさしへとディゾルブするように見えます。



シェーダーはディゾルブエフェクトを作成できる。

- **頂点アニメーション**：頂点アニメーションを使用して、有機的な動きや動的な表面をシミュレートします。頂点シェーダーは、メッシュの頂点を動かして GPU 上で直接アニメーションを作成できます。例えば、あるメッシュを別のメッシュにモーフしたり、パーティクルストリップを弧状に曲げたり、波打つ旗や火の玉の尾のように表面を波立たせたりできます。

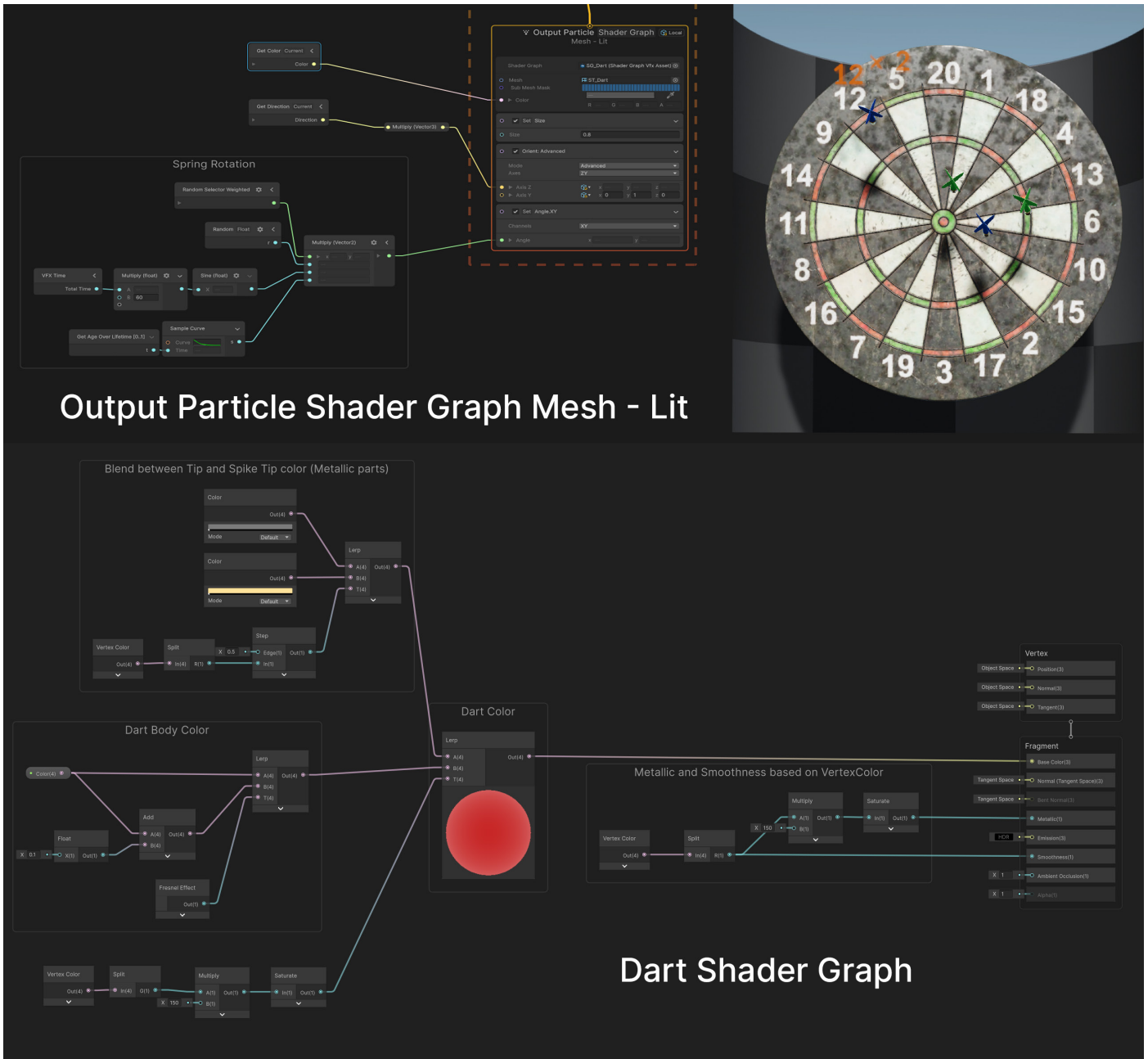


頂点アニメーションは有機的な動きをシミュレートできる。

- **ライティングとシェーディング**: 高度なライティングエフェクトでリアリズムやスタイライズを加えましょう。VFX Graph は、光り輝くシェーダーや、反射、屈折、影などの複雑なライティングインタラクションにより、しばしば効果が高まります。

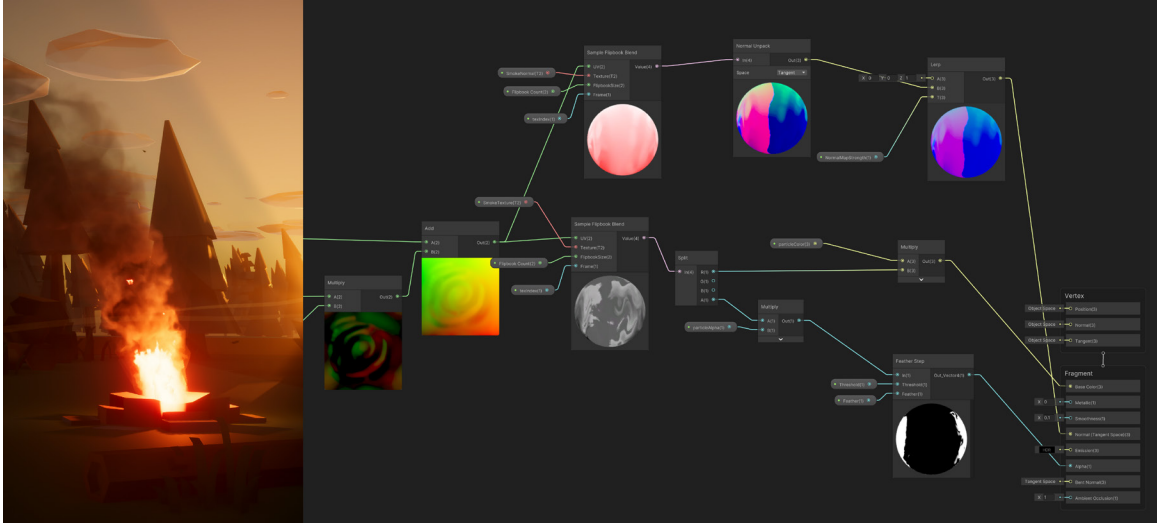
例えば、ガラス、水、金属を模倣する魔法のエフェクトがある場合、シェーダーは正確な反射をシミュレートする必要があります。トゥーンシェーディングゲームを作っていますか？ Shader Graph は、アニメのようなスタイルのエフェクトをレンダリングするのに役立ちます。

この例では、ダーツは Shader Graph を使用したパーティクルで、弾力性のある発射物のように、光沢のあるメッシュをレンダリングしています。



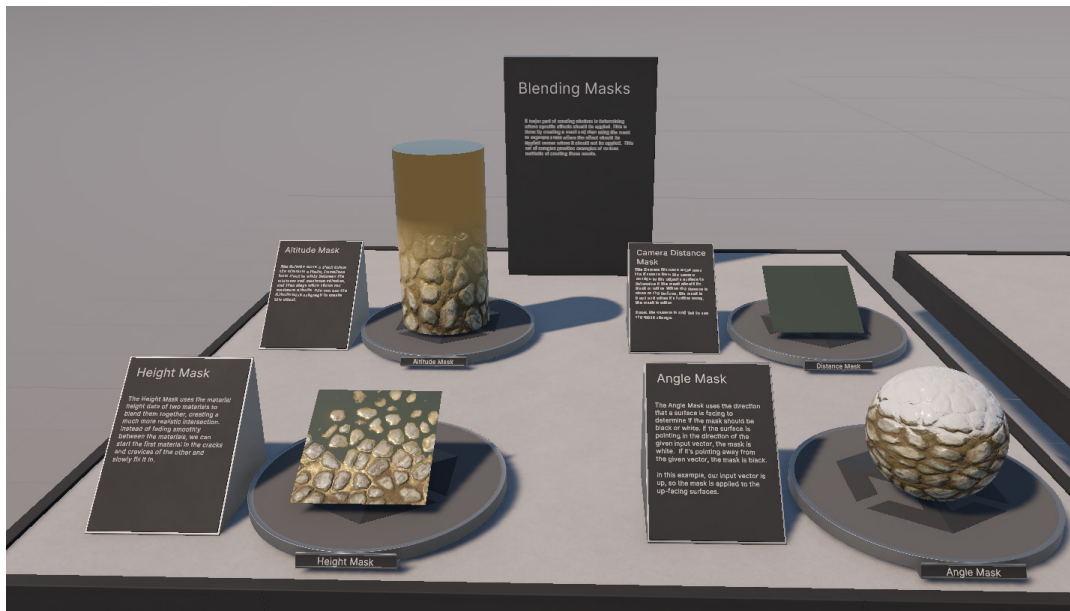
シェーディングはメッシュパーティクルを強化する。

- **色の遷移**：シェーダーは、時間、位置、あるいはユーザー入力に基づいて色を補間できます。ランプやグラデーションを使用すると、サーフェス全体やアニメーションシーケンスを通じて色を滑らかに遷移させることができます。炎が熱い青色の中心から赤と黄色の外層へと変換するなど、エフェクトをよりダイナミックにします。



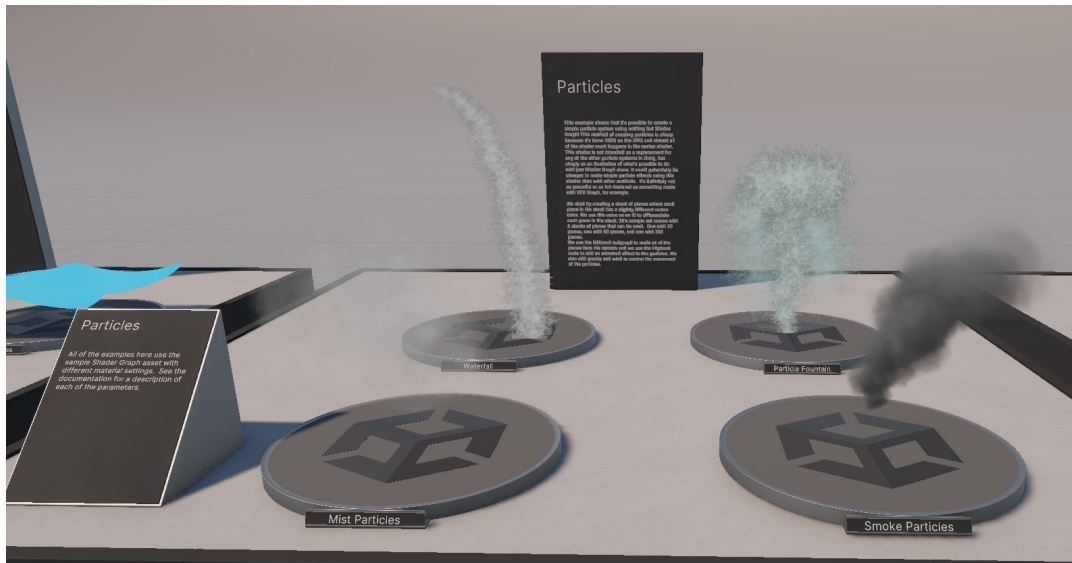
色の遷移はエフェクトをより動的にすることができます。

- **ブレンディングマスクと透明度**：さまざまなレベルの透明度が必要なエフェクトは、カスタムシェーディングの恩恵を受けることができます。Shader Graph を使用してピクセルのアルファ値を制御します。ブレンディングマスクは、高さ、角度、カメラまでの距離に基づいてフェードさせることができます。この手法は、ガラスや氷など、光を透過させる必要があるものにも便利です。



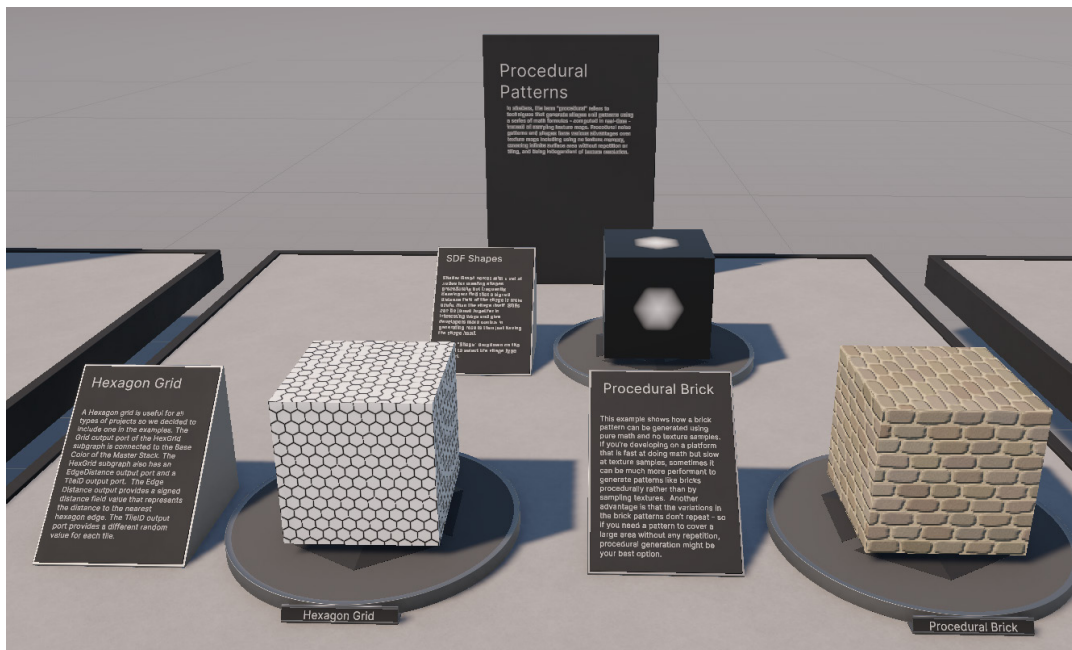
ブレンディングマスクは距離や角度によってフェードすることができる。

- **パーティクルシェーディング**: シェーダーは、小さなビジュアルディテールを追加したり、さまざまなパラメーターに基づいて外観を操作したりすることで、パーティクルエフェクトに複雑さを加えることができます。例えば、Shader Graph を使用して花火のパーティクルの色、サイズ、明るさを制御します。または、シェーダーを適用して魔法の呪文の色を変えたり、発光をアニメーション化したりできます。



シェーダーはパーティクルの色を変えたりアニメーション化したりできる。

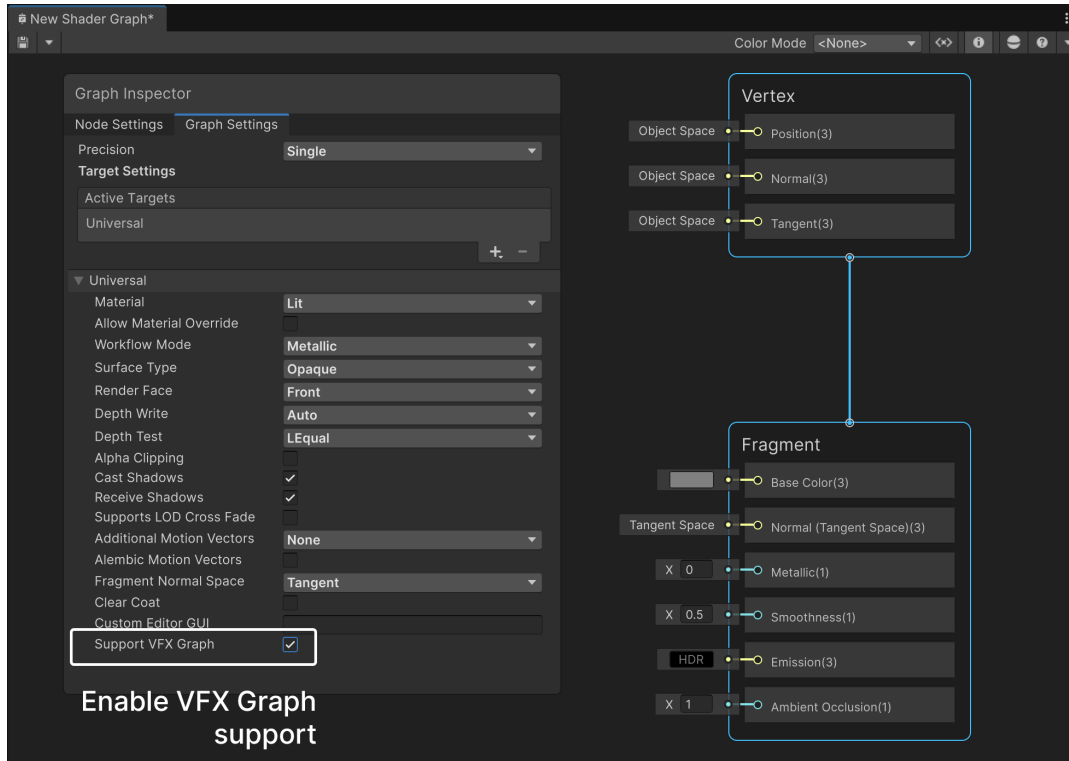
- **プロシージャルパターンとテクスチャ**: シェーダーはテクスチャをプロシージャルに生成できるため、テクスチャを事前に用意する必要はありません。これにより、サイズの大きなテクスチャファイルの必要性を減らし、メモリ使用量を削減しつつ独自のパターンを作成できます。テクスチャは数学関数に基づいているため、カメラの近くにおいてもピクセレーションやぼやけがなく、ディテールを表現できます。



シェーダーは、テクスチャとパターンをプロシージャルに生成できる。

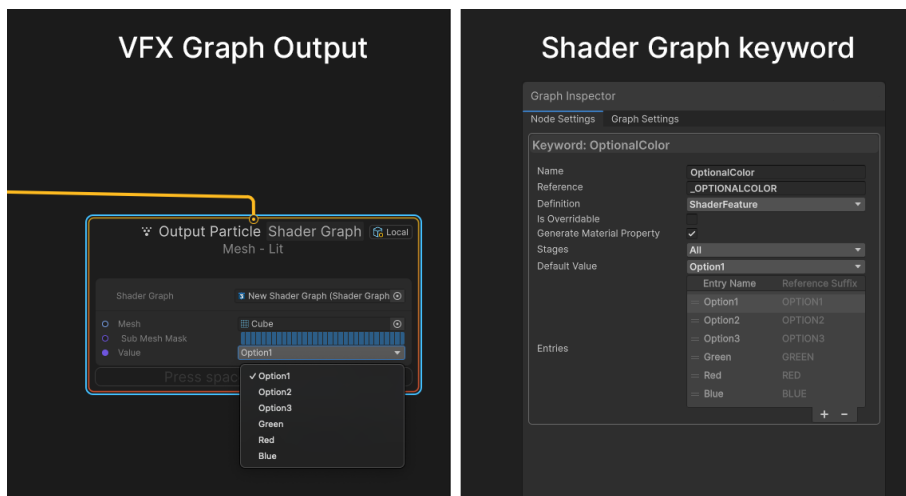
Shader Graph と VFX Graph

新しい Shader Graph を作成するときは、**Support VFX Graph** オプションをオンにし、VFX Graph の Output の対応するフィールドに割り当てます。これにより、シェーディングと VFX のパラメーターをまとめることができます。



シェーダーの Support VFX Graph を有効にする。

VFX Graph に Shader Graph キーワードとの連携機能が追加されました。これにより、1 つの Shader Graph を作成して、複数の VFX Graph で使用できます。次に、VFX Graph Out Particle Mesh ノードのキーワードに基づいて機能を有効にします。

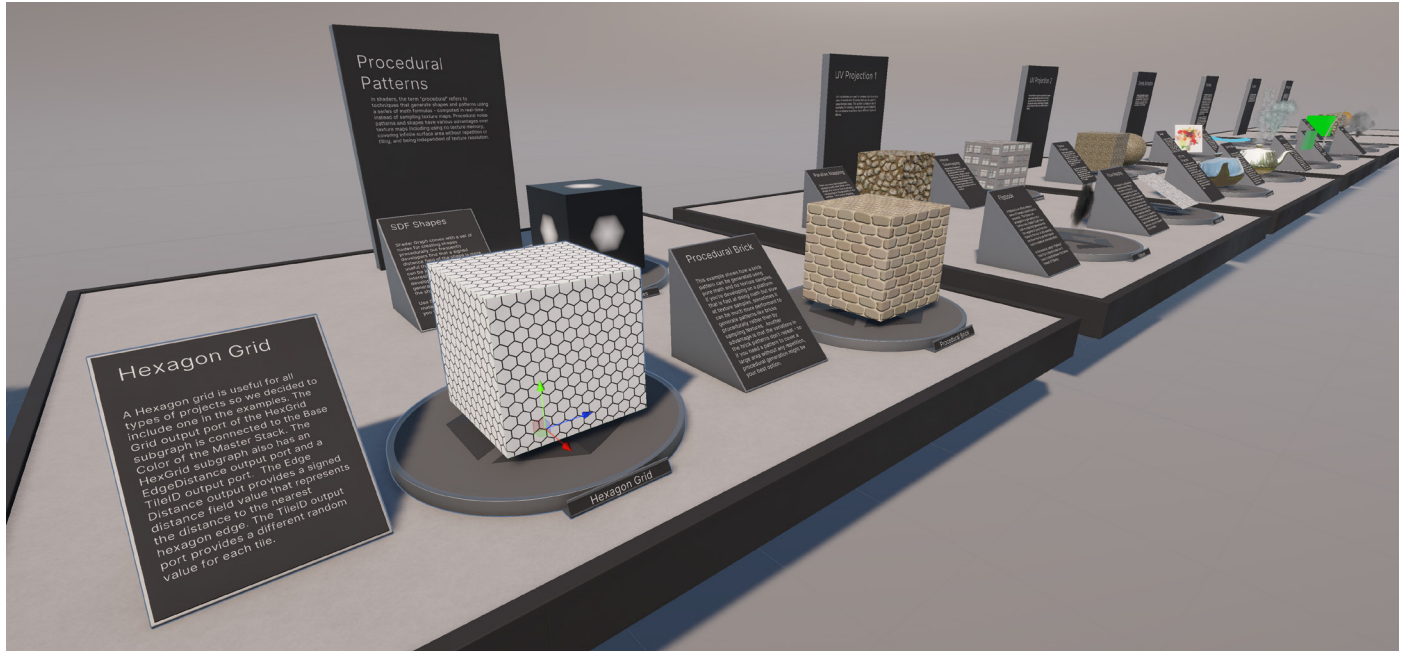


VFX Graph は、Shader Graph keyword に基づいた動作を有効化できる。

VFX Graph の Learning Templates でサンプルを探すことも、Package Manager で利用可能な Shader Graph Feature Examples でより一般的なサンプルを探すこともできます。

Shader Graph Feature Examples

[Shader Graph Feature Examples](#) のサンプルコンテンツは、Shader Graph で一般的な手法とエフェクトを実現する方法を示す、Shader Graph アセットのコレクションです。このサンプルパックの目的は、ユーザーが特定のエフェクトを実現するために必要なものを確認し、サンプルを提供して簡単に習得できるようにすることです。



Shader Graph Feature Examples を見てみよう。

インタラクティブ性

ビジュアルエフェクトには、多くの可動パーツが関係することがよくあります。それらをアプリケーション内の正しいポイントに接続することは、ランタイム時に連携するために極めて重要です。

接触して爆発する砲弾や、マウスポインターから飛び出す稲妻が必要な場面など、Unity シーンの他の部分とのインタラクションにはこれらのツールが役立ちます。

- **イベントバインダー** : シーン内で起こるさまざまなことを検知し、ランタイムに特定のアクションに反応します。
- **Timeline** : Activation Track を使用してビジュアルエフェクトをシーケンス化し、選択したタイミングでグラフにイベントを送信できます。あらかじめ設定されたタイミング (カットシーン中のエフェクト再生など) で正確に制御できます。
- **プロパティバインダー** : シーンやゲームプレイの値を Blackboard の **Exposed プロパティ** にリンクすることで、シーンの変化にエフェクトがリアルタイムで反応するようにします。
- **Output イベント** : VFX Graph からスクリプトやその他のシーンコンポーネントにメッセージを送信するために使用します。

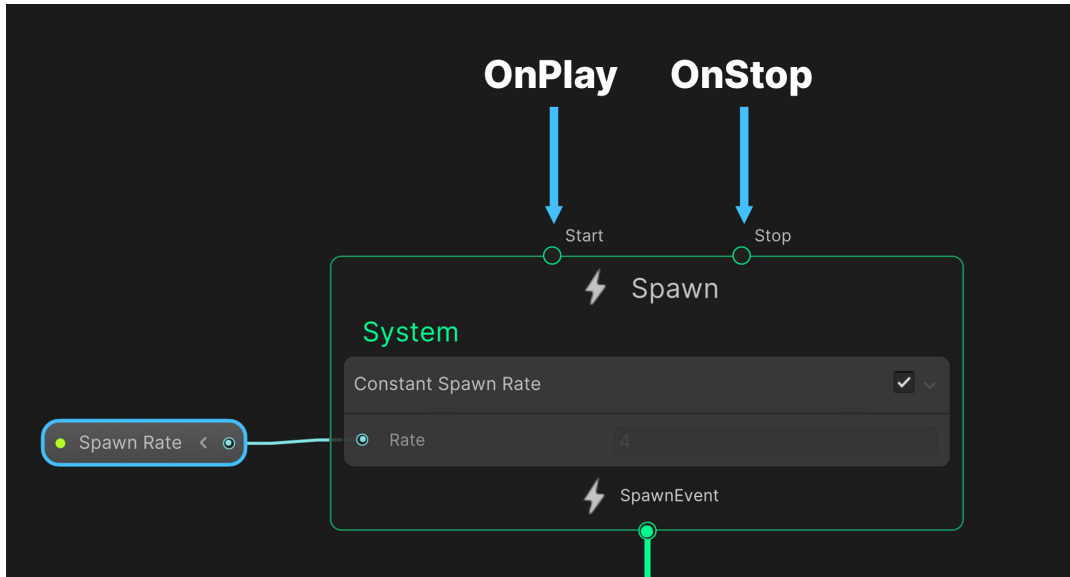
これらのツールはそれぞれ、ゲームオブジェクトと VFX Graph の橋渡しをするうえで非常に重要なテクニックであるため、詳しく見ていきましょう。

イベントバインダー

イベントバインダーは、VFX Graph 内から**イベント**を起動できる **MonoBehaviour** スクリプトです。シーン内のマウスアクション、衝突、トリガー、可視性イベントにエフェクトが反応するようになります。

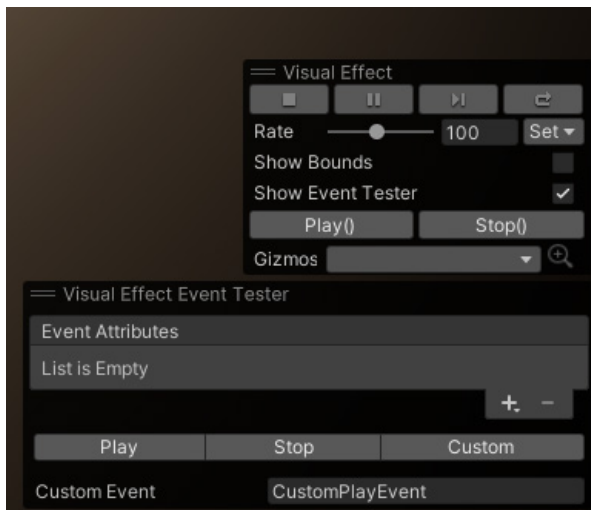
OnPlay および OnStop イベント

デフォルトでは、VFX Graph の Spawn コンテキストには OnPlay イベントまたは OnStop イベントがあります。各 Spawn コンテキストの Start および Stop フロースロットは、特定のイベントに接続しない場合に暗黙的にこれらを受け取ります。



OnPlay と OnStop は、暗黙的に Spawn コンテキストに流れる。

Scene ビューのゲームオブジェクトの再生コントロールに慣れている場合は、下部の **Play()** ボタンと **Stop()** ボタンで、それぞれ OnPlay イベントと OnStop イベントを送信します。



Play ボタンと Stop ボタンを使用して、OnPlay イベントと OnStop イベントを送信する。

イベントは、オブジェクト間でメッセージを送信するプロセスを容易にします。VFX Graph では、イベントは文字列として渡されます。OnPlay または OnStop を押しても、エフェクトはすぐには変化しません。特に上部の再生アイコンを使用した場合と比べると顕著です。これらは Spawn システムにシグナルを提供するだけです。

Visual Effect Event Tester ダイアログウィンドウを開けば、Play ボタンと Stop ボタンを同じエフェクトに使用できます。この柔軟性を利用して、**カスタムイベント**を定義し、Custom ボタンで呼び出します。

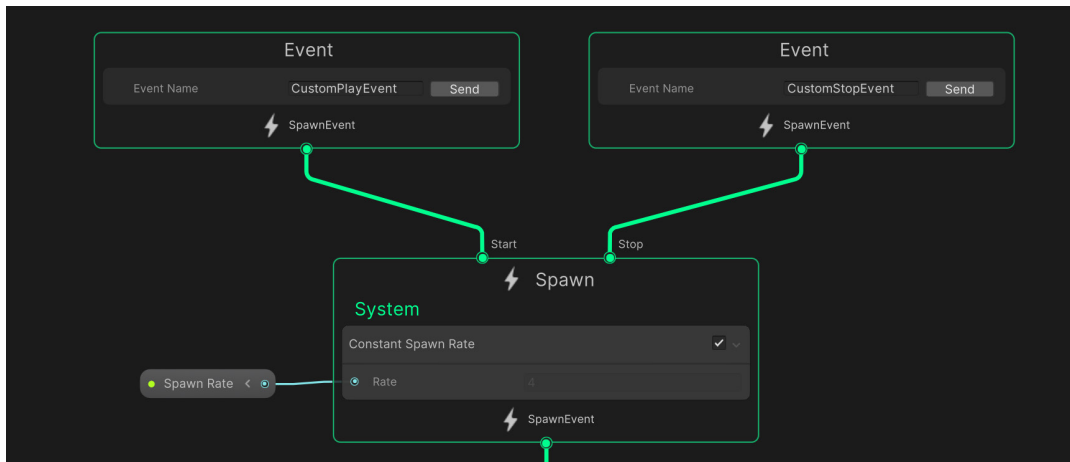


Visual Effect Event Tester

Node > Context > Event を使用して、独自のイベントをグラフに追加します。**Send** ボタンを押して、テスト時に手動でイベントを発生させます。

カスタムイベントの作成では、**Event Name** 文字列を変更し、ランタイム時に **Event Binder** か **Timeline Activation Clip** を使用してイベントを呼び出します。

この例では、**CustomPlay** と **CustomStop** という 2 つのイベントが追加されています。

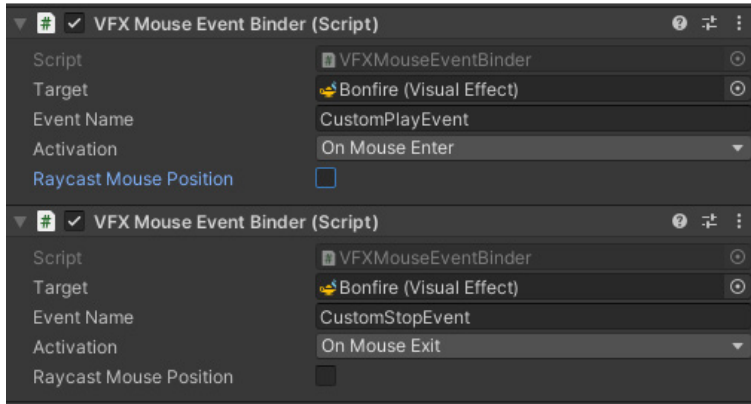


グラフでカスタムイベントを設定する方法

Mouse イベントバインダー

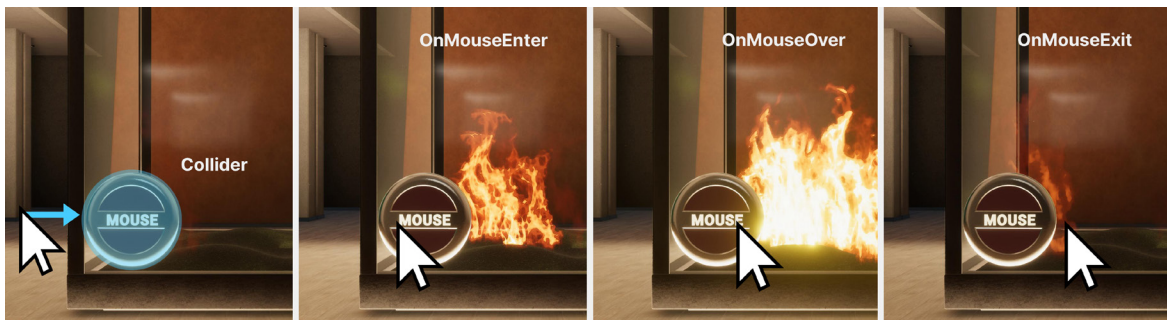
マウスポインターをクリック、ホバー、ドラッグすることで、VFX Mouse イベントバインダーを使用してグラフにメッセージを送信できます。必要なのは、コライダーを持つゲームオブジェクトだけです。

この例では、2 つのイベントバインダーが **CustomPlay イベント**と **CustomStop イベント**を Bonfire エフェクトに接続します。



Mouse イベントバインダーでエフェクトをアクティベートする

マウスポインターが画面上のコライダーに入ったら、CustomPlay イベントをグラフに送信します。このイベントは、炎、煙、火花の生成を開始します。マウスポインターが外れると、CustomStop イベントは Spawn コンテキストに停止を通知します。

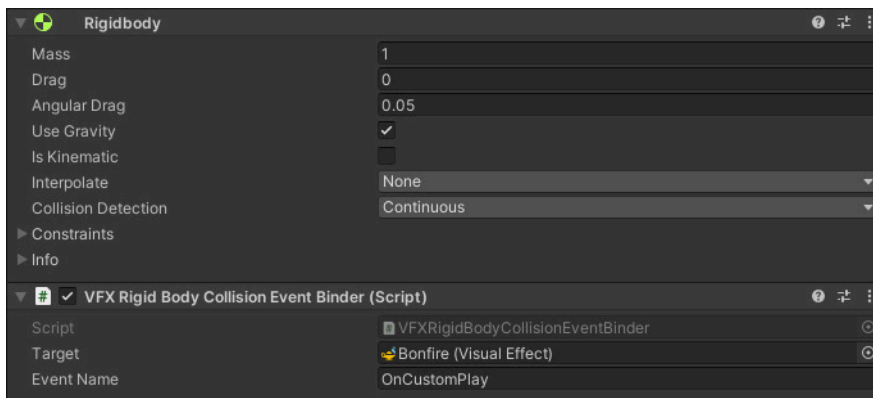


マウスポインターは、コライダーの出入り時にイベントを発生させる。

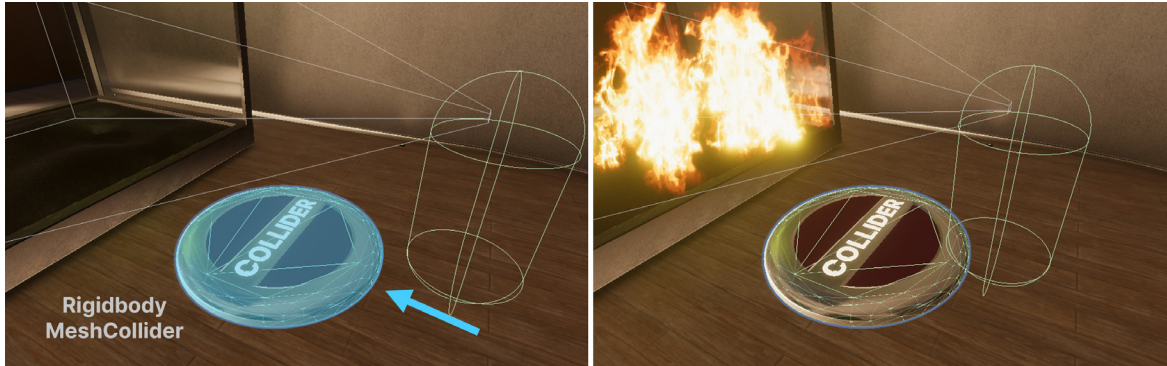
イベントは、標準のマウスアクション (**Up**、**Down**、**Enter**、**Exit**、**Over**、**Drag**) にバインドできます。**Raycast Mouse Position** オプションは、ポインターの 3D 位置を **イベント属性** として渡します。

Rigidbody Collision イベントバインダー

Rigidbody Collision イベントバインダーを使用すると、衝突が発生したときに物理オブジェクトがグラフにアラートを送信できます。**Rigidbody** (リジッドボディ) と **Collider** (コライダー) を使用して適切なゲームオブジェクトにアタッチします。



衝突が VFX Graph にイベントを送信する。



リジッドボディの衝突

接触したコライダーは、(指定されたイベント名で) イベントを生成し、メッセージとして **Visual Effect Target** に送信します。

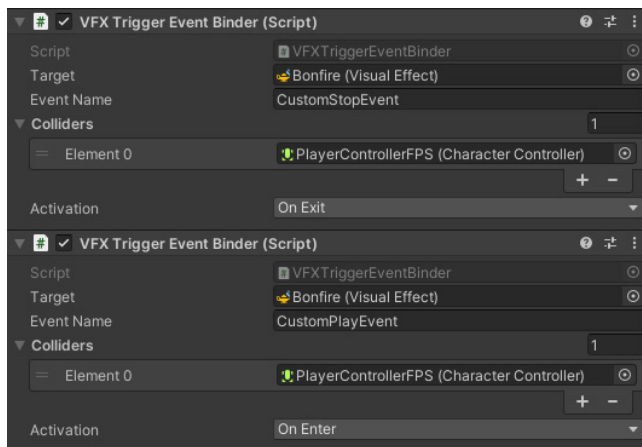
この例では、リジッドボディの**メッシュコライダー** がボタンとして機能します。プレイヤーが接触してから、エフェクトのスポーンが始まります。

このイベントバインダーがリジッドボディとの衝突に応答するので、さまざまな形式のインタラクティビティを作成できます。跳ね返るたびにパーティクルを放出するスフィアや、砲弾が当たると歪む力場を想像してみてください。

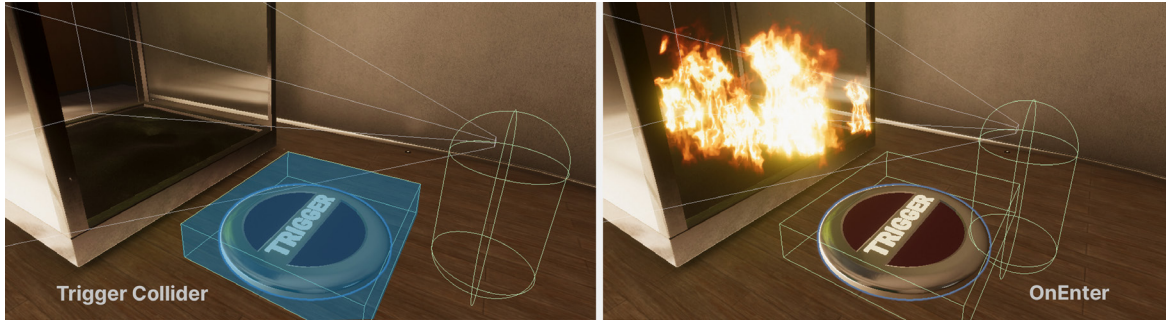
Trigger イベントバインダー

同様に、トリガーとして設定されたコライダーを使用して、ゲームオブジェクトに VFX Trigger イベントバインダーを追加することもできます。このコンポーネントは、**OnEnter**、**OnExit**、**OnStay** などの特定の **Activation イベント** を登録できます。Event Name のイベントを **Target** グラフに送信します。

これを使用すると、プレイヤーやその他のゲームオブジェクトがレベルの特定の部分に到達したときにエフェクトをトリガーできます。



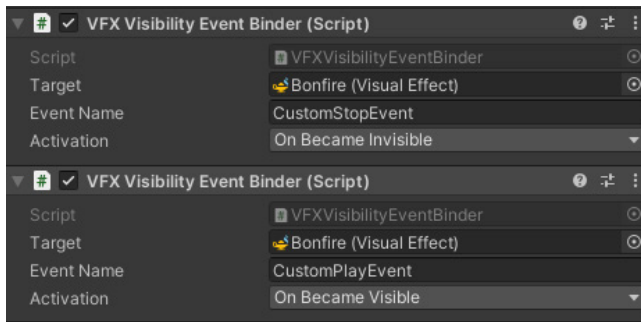
Trigger イベントバインダー



Trigger イベントバインダーは、特定のコライダーを検出する。

Visibility イベントバインダー

Visibility イベントバインダーを使用すると、オブジェクトが可視になるか非表示になるかに応じてイベントを発生させることができます。ここでは、何らかの種類のレンダラーでゲームオブジェクトにアタッチします。



Visibility イベントバインダー

特定のイベント名を、その **Activation (OnBecameVisible または OnBecameInvisible)** に基づいてターゲットグラフに渡します。これにより、レンダラーがカメラ錐台に出入りしたときにグラフに通知されるか、レンダラーのオンとオフが切り替わります。



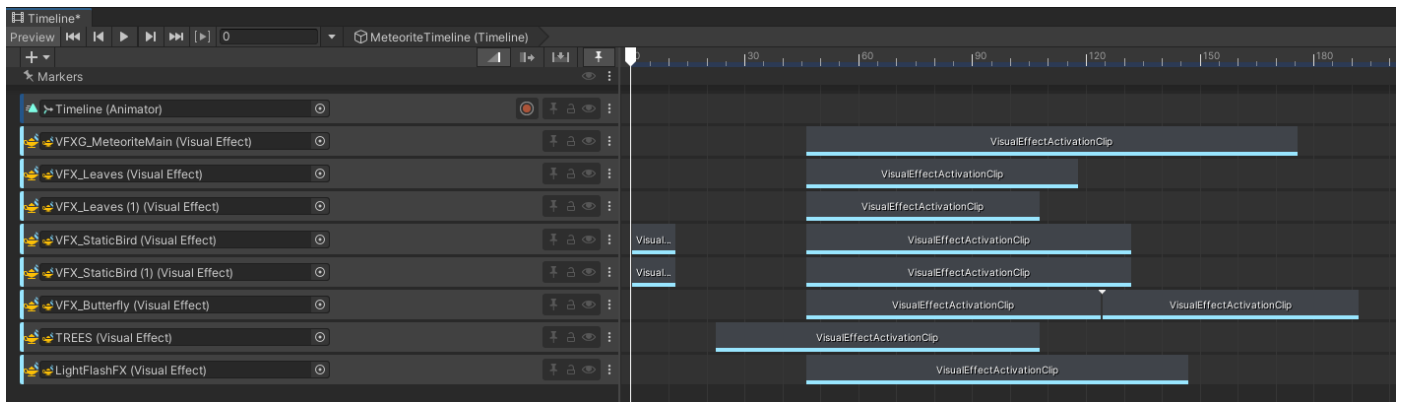
ビジュアルエフェクトは、ターゲットが可視になったときにのみアクティベートされる。



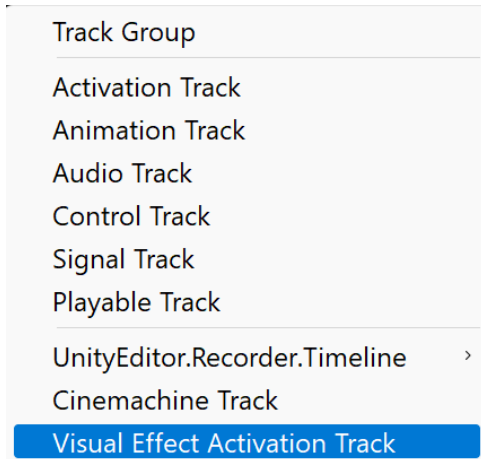
ビデオクリップ: イベントバインダー

タイムライン

Timeline は、正確なタイミングでビジュアルエフェクトのオンとオフを切り替える必要がある場合に、グラフと通信するためのもう 1 つの方法です。Visual Effect Activation Track を使用して、複数のエフェクトレイヤーを調整できます。

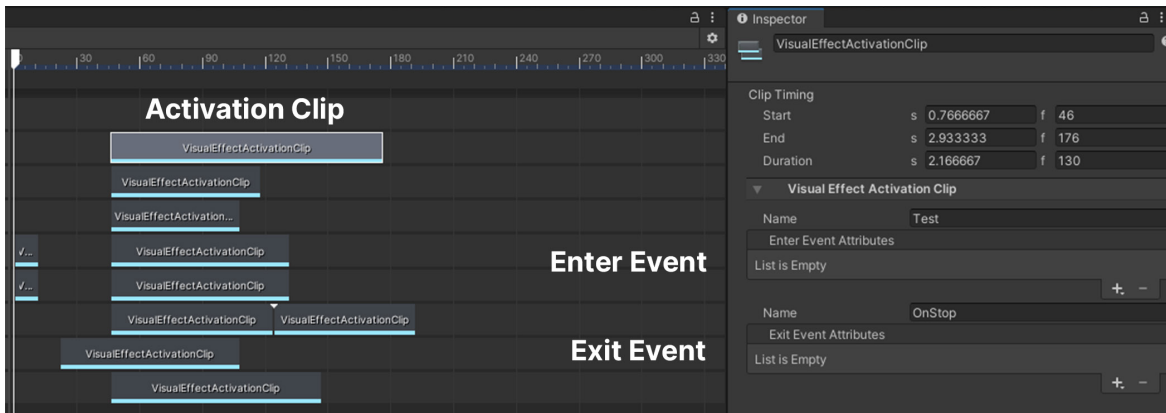


Visual Effect Activation Track

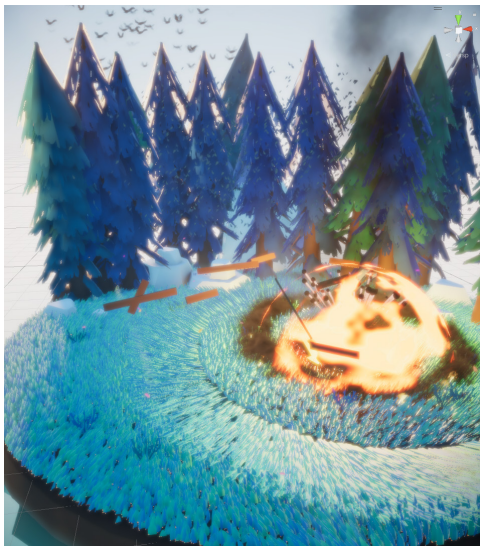


Timeline 内を右クリックして、新しい Activation Track を作成する。

Activation Track にエフェクトを割り当て、Timelineトラックで1つ以上の **Activation Clip** を作成します。各クリップは、最初と最後に1つずつ、2つのイベントを送信します。



各 Activation Clip は 2 つのイベントを VFX Graph に送信する。

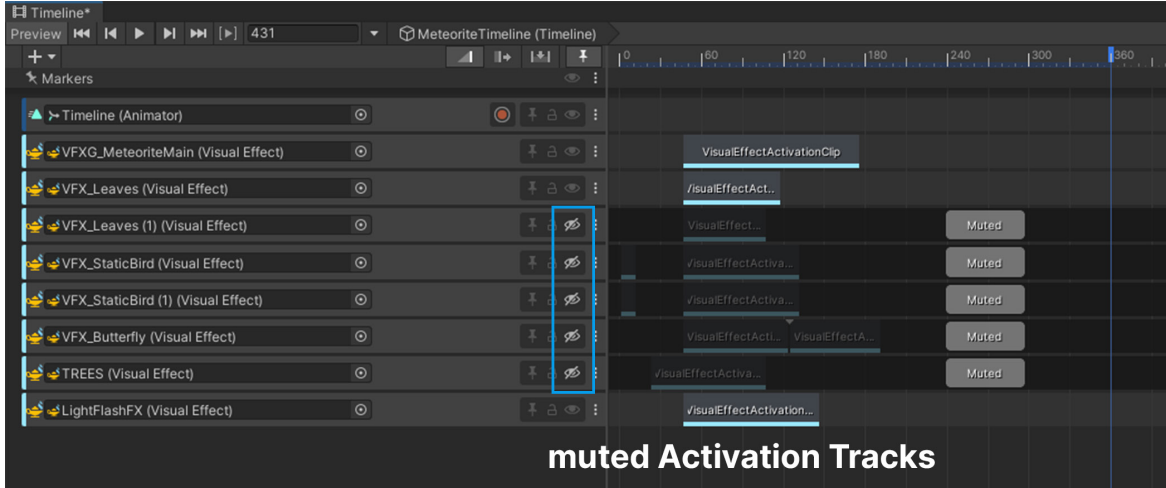


VFX Graph Samples プロジェクトの Meteorite サンプル

Meteorite サンプルでは、Timeline を使用して複数のエフェクトを連動させて再生する制御方法を説明しています。

Timeline は、エフェクト全体をまとめて作成するピースを整理するのに役立ちます。ここでは、各サブグラフに別個の Activation Track が使用され、Activation Clip を通じてイベントが渡されます。

Timeline 内でクリップをスライドさせることで、タイミングをインタラクティブに調整できます。その後、カスタム MeteoriteControl スクリプトによって Playable Director コンポーネントが呼び出されます。



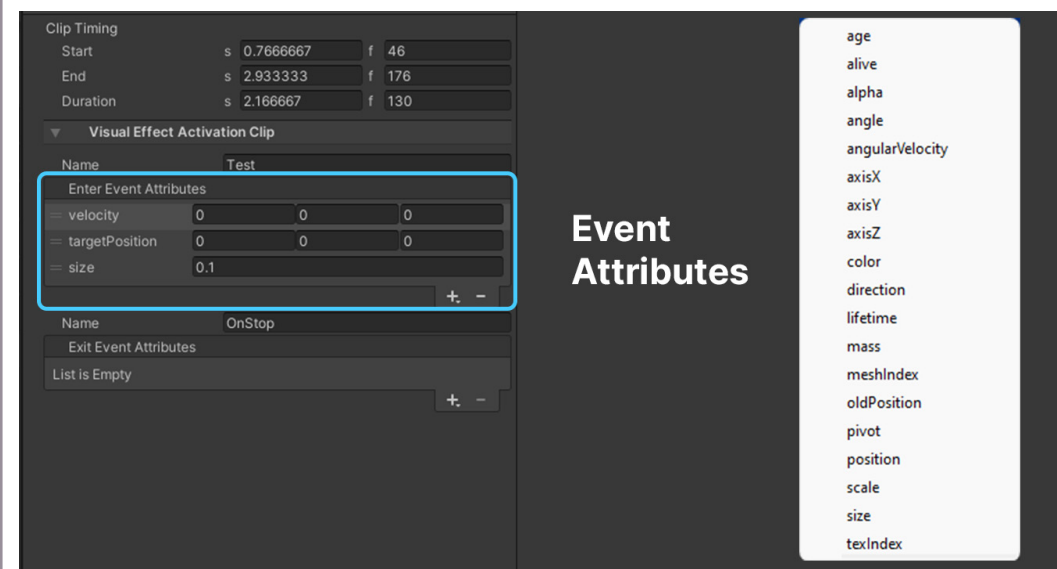
Activation Track をミュートする

また、Timeline を使用して特定の Visual Effect Activation Track をミュートし、イベントの受け取りを一時的に停止することもできます。これはトラブルシューティングを行う時に便利です。

イベント属性

イベントと Timeline Activation Clip の両方で、[イベント属性ペイロード](#)をイベントにアタッチできます。そうすることで、イベントが起動されたときに追加の情報を渡します。

例えば、公開された **Vector3** プロパティを持つイベント属性を作成し、エフェクトをインスタンス化する場所をグラフに通知することができます。



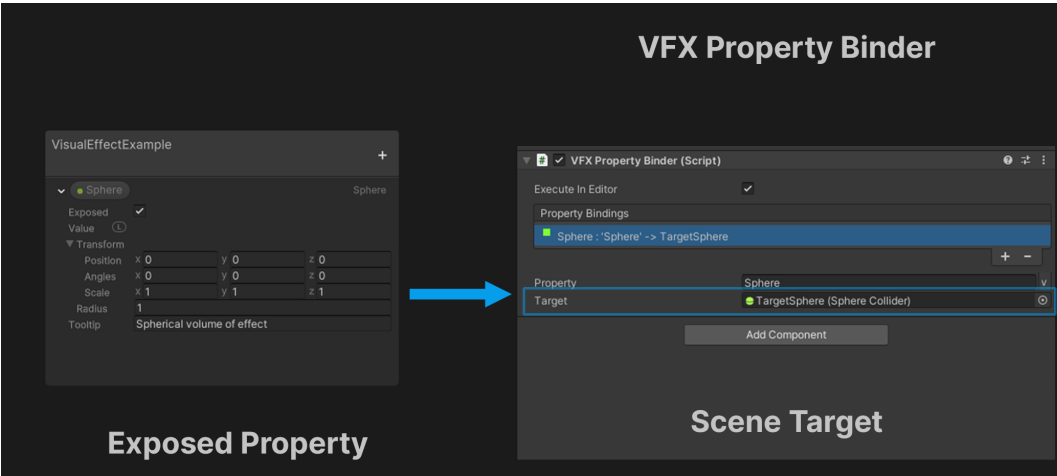
イベント属性を使って、イベントをトリガーする際にデータを渡す。

VFX Graph でこれらの属性を設定するには、**Spawn コンテキスト**の **Set Attribute ブロック**を使用します。C# スクリプトから送信されるイベントにアタッチすることもできます。詳細については、Visual Effect [コンポーネント API](#) を参照してください。

プロパティバインダー

プロパティバインダーは、シーンやゲームプレイの値を VFX Graph の[公開されたプロパティ](#)に接続できるようにする C# の動作です。プロパティバインダーは、**VFX Property Binder** と呼ばれる一般的な MonoBehaviour を通じて追加できます。

例えば、**Sphere Collider Binder** では、シーン内の Sphere Collider の値を使用して **Sphere Exposed Property** の位置と半径を自動的に設定できます。

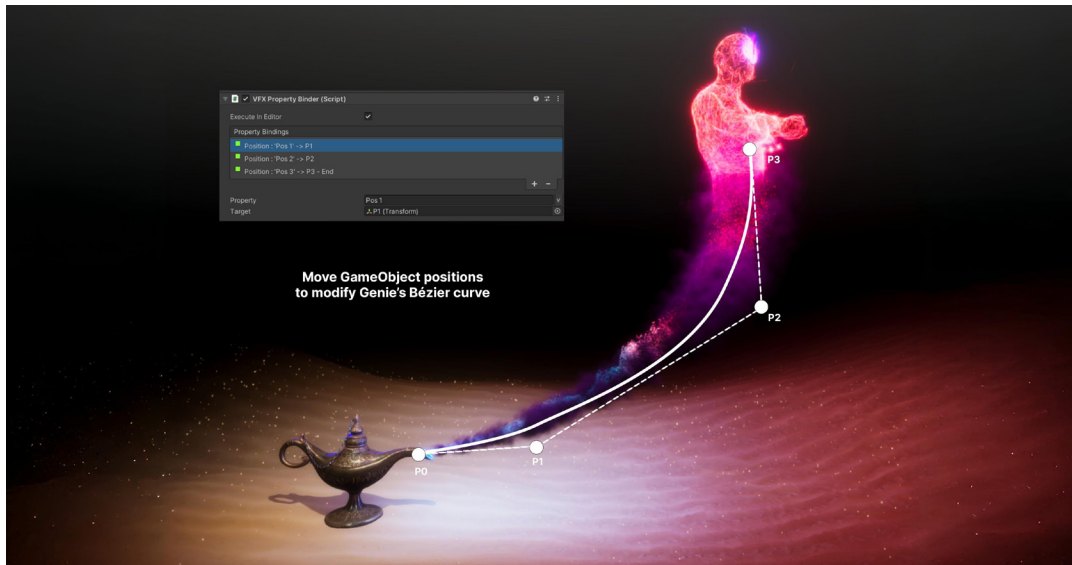


プロパティバインダーは、シーンの値を公開されたプロパティに接続する。

ランタイム時にエフェクトに影響を与えるために、シーン内にライトやカメラが必要ですか？エフェクトは Transform や Vector3 に従いますか？プロパティバインダーは、多くの Exposed Property タイプをシーン内の値と同期させることができます。何が利用できるかは、**Add Component > VFX > Property Binders** を参照してください。

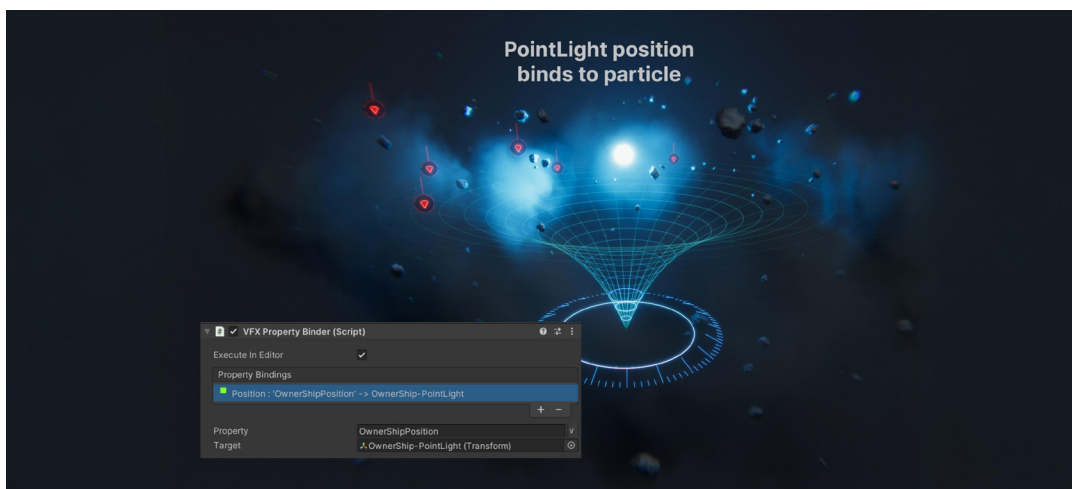
Visual Effect Samples には、プロパティバインダーを使用して Scene Hierarchy をグラフに接続する方法を示すインスタンスがあります。

- Magic Lamp サンプルでは、プロパティバインダーが複数の Scene オブジェクトの位置 (P1、P2、P3) をグラフの Blackboard プロパティ (Pos1、Pos2、Pos3) に関連付けます。これらの **Vector3 ノード** は、魔人の全体的な形状を定義する **Bézier スプライン** を形成します。シーン内で P1、P2、P3 **Transforms** を動かすと、魔人の煙跡がリアルタイムで反応します。



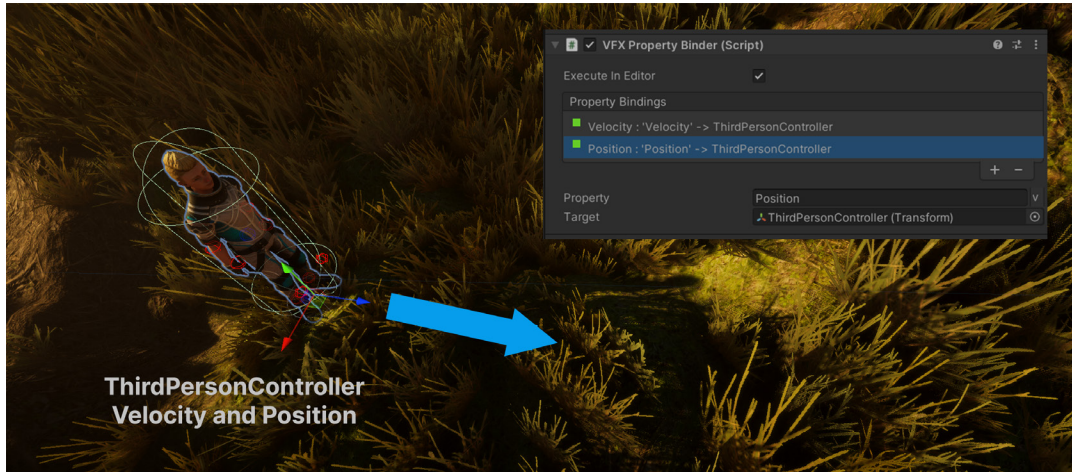
得られた Bézier 曲線が魔人の形を変える

- ARRadar では、**PointLight Transform** によって、プレイヤーの船体が 3D レーダー画面上のどこに表示されるかが決定されます。輝点をリアルタイムライトと同期させます。



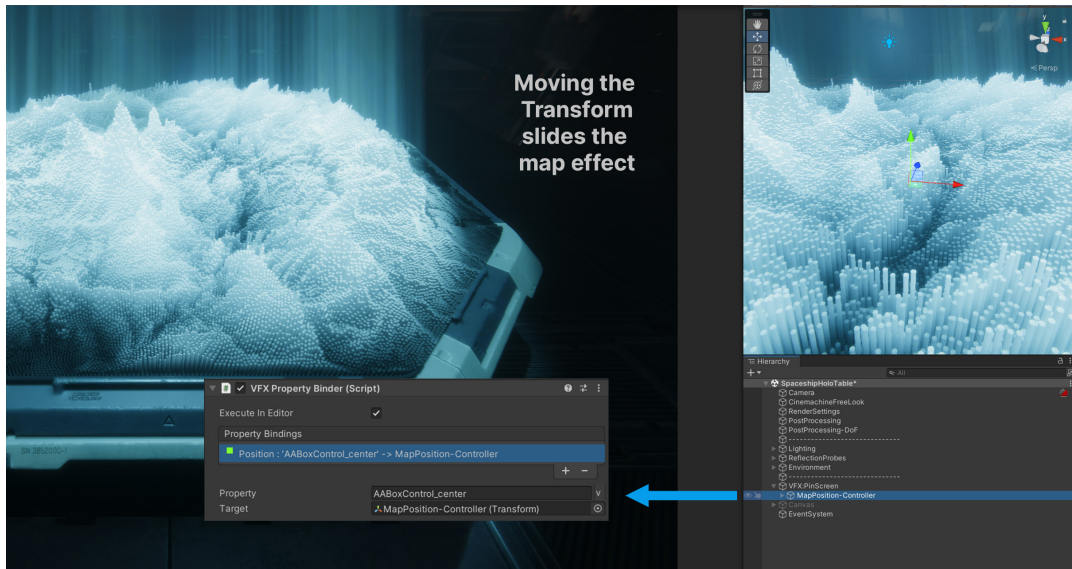
ARRadar が、PointLight を船体アイコンに接続する。

- Grass Wind サンプルでは、プロパティバインダーが **ThirdPersonController** と呼ばれる Transform の位置と速度をキャプチャし、草を動かします。



Grass Wind サンプル

- SpaceshipHoloTable は、**MapPosition-Controller** と呼ばれる Transform を使用して、卓上の PinScreen エフェクトの位置を制御します。事前に設定されたモーションで **Animator** がこれを動かすか、ランタイム時にドラッグしてホログラムが動き出します。



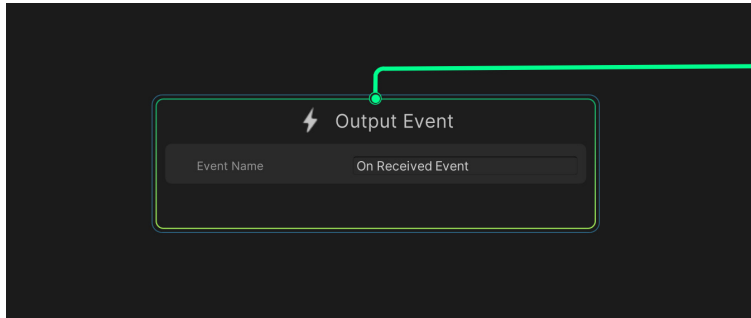
SpaceshipHoloTable

これらは、プロパティバインダーでグラフとシーンの関係性を固めることのできる、ほんの一例です。オーディオ、入力、物理演算、UI などの **ビルトインプロパティバインダー** が見つかります。

まだ存在していないものが必要ですか？ **UnityEngine.VFX.Utility.VFXBinderBase** クラスを使用して、**独自のプロパティバインダーを記述**してください。

Output イベント

イベントを利用して VFX Graph にメッセージを送信するのと同じように、イベントを使用してメッセージを外部に送信することもできます。[Output イベント](#)を使用すると、**Spawner コンテキスト**から新しいパーティクルの属性を取得できます。**Output イベントハンドラー**と一緒に使用して、シーン内の C# スクリプトに通知します。



Output イベント

カメラを揺らす、サウンドを再生する、プレハブを生成するなど、ゲームプレイロジックに指示するエフェクトに反応する動作をいくつでも作成できます。

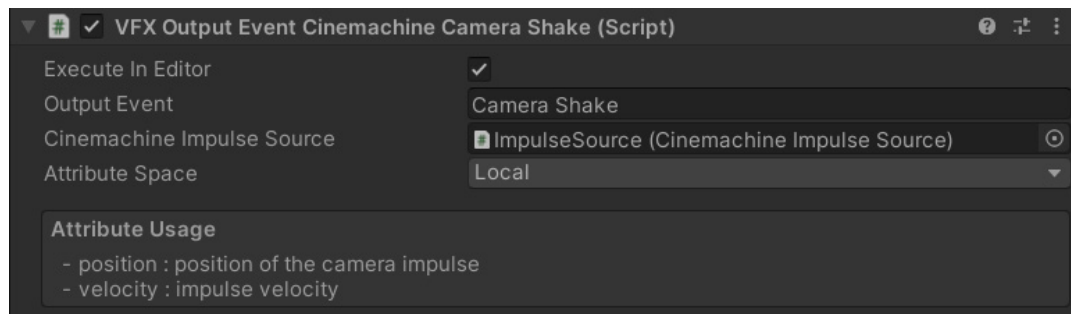
イベントを受け取るには、**UnityEngine.VFX.Utility** 名前空間内の **VFXOutputEventAbstractHandler** から[スクリプト化されたクラスを継承](#)します。次に、このメソッドをオーバーライドします。

```
override void OnVFXOutputEvent(VFXEventAttribute eventAttribute)
```

Unity は、イベントがトリガーされるたびに `OnVFXOutputEvent` を呼び出し、イベント属性をパラメーターとして渡します。VFX Graph に含まれている **Output イベントハンドラー** に実装例があります。Package Manager からインストールして、サンプルスクリプトを確認してください。

また、Meteorite サンプルで、実際のグラフ内でどのように機能するかを確認することもできます。これらの Output イベントハンドラーの一部を **VFX_MeteoriteMain** ゲームオブジェクトにアタッチします。

- **カメラの揺れ** : **VFX Output Event Cinemachine Camera Shake** コンポーネントは **Camera Shake イベント**時にカメラを揺らします。



Camera Shake スクリプト

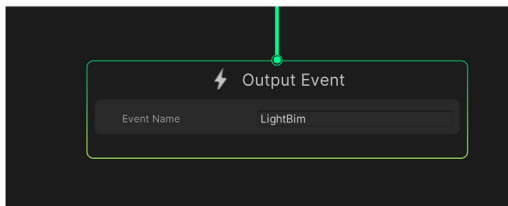
- **二次エフェクト :VFX Output Event Prefab Spawn** コンポーネントは、**Buffer イベント**と **PlankImpulse イベント**を生成します。その結果生じる衝撃波が草を突き抜け、Output イベントハンドラーの働きにより木片が飛びます。



Output イベントハンドラーは二次エフェクトを生成する。

- **ライトアニメーション** : 別の VFX Output Event Prefab Spawn により、隕石の衝突時のライトアニメーションが作成されます。カスタムの **Output Event Handler Light Update** スクリプトは、ボリュームメトリックスケール、明度、色を変更し、衝突をよりドラマチックに演出します。

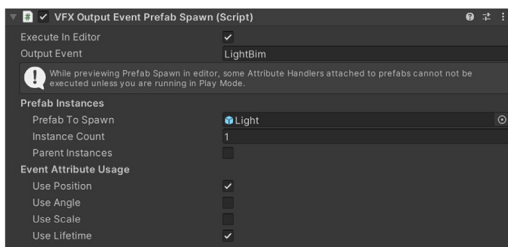
VFX Graph Output Event



GameObject



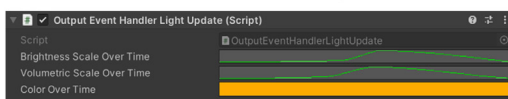
Output Event Handler spawns Prefab



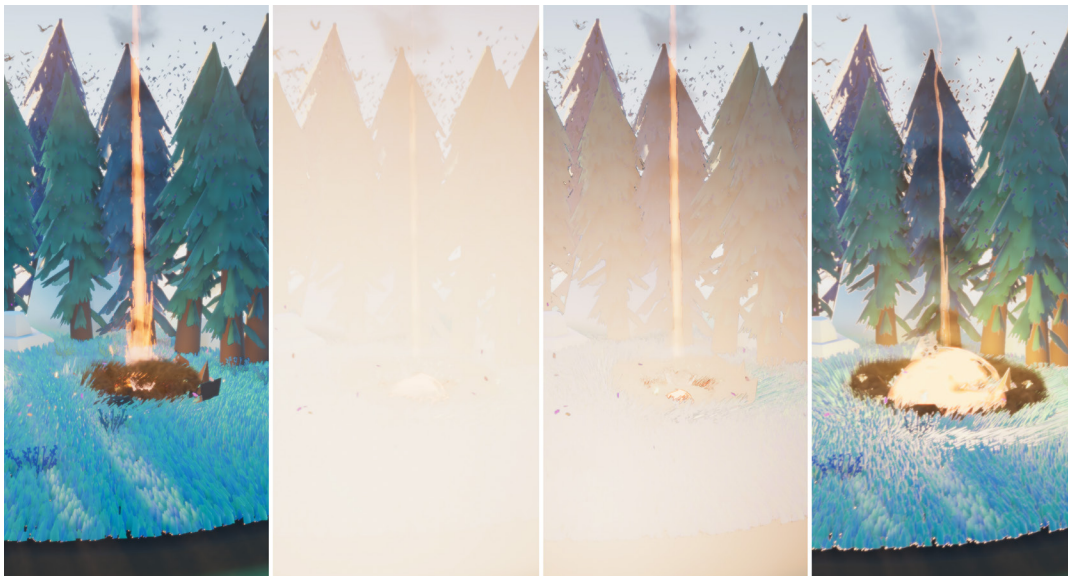
Prefab instance



Light animation from another Output Event Handler



カスタム Light Update Output イベントハンドラー



隕石の衝突エフェクトにライトが反応する。

これらのアイデアを見るところから始めてみましょう。サンプルスクリプトは、コードを書かずに直接使用することも、独自のスクリプトのたたき台として扱うこともできます。少し時間をかければ、アプリケーションで必要な Output イベントハンドラーを自作できるようになります。

パイプラインツール

エフェクトは孤立しているものではありません。多くの場合、意図した外観を実現するためには、外部データを提供する必要があります。

魔法のランプから魔人を出現させたい場合はどうすればよいでしょうか？あるいは、ホログラムを Sci-Fi 宇宙船と連携させたいですか？その多くは数学関数やオペレーターを使用して実現できますが、より複雑な形状やフォームと相互作用させるにはエフェクトが必要になる場合があります。

このため、Unity では、多くのデータタイプを用意しています。

- **Point Cache**: Transform、法線、色、UV など、空間内の点の属性を格納します。
- **符号付き距離フィールド**: ボリュームトリック表現を使用してパーティクルを引き付けて衝突させます。
- **ベクトル場**: パーティクルの位置をサンプリングし、3D 空間でパーティクルを動かします。

Unity では、これらのファイル形式の生成を支援するサポートユーティリティもいくつか用意されています。

Point Cache

Point Cache は、ポイントとその位置、法線、色など、**Particle 属性**データの固定リストを格納するアセットです。

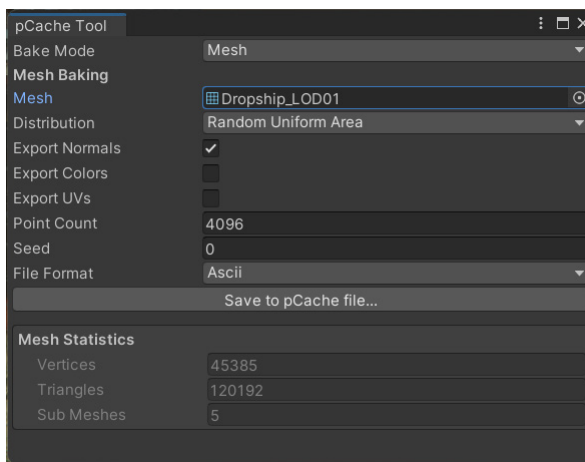
Point Cache アセットは、オープンソースの **Point Cache 仕様**に従っており、ファイル拡張子は **.pCache**を使用します。Point Cache は内部的にはネストされた **ScriptableObject** であり、Particle 属性のマップを表すさまざまなテクスチャが含まれます。符号付き距離フィールドよりも少ないリソースで済みます。

ビジュアルエフェクトで使用する Point Cache を生成するには、次の手順に従います。

- ビルトインの [Point Cache](#) ベイクツールは、**Window > Visual Effects > Utilities > Point Cache Bake Tool** で使用できます。
- [VFXToolbox](#) にバンドルされている **Houdini pCache Exporter** を選択します。
- 独自のカスタムエクスポートを作成します。アセットの形式と仕様の詳細については、[pCache README](#) を参照してください。

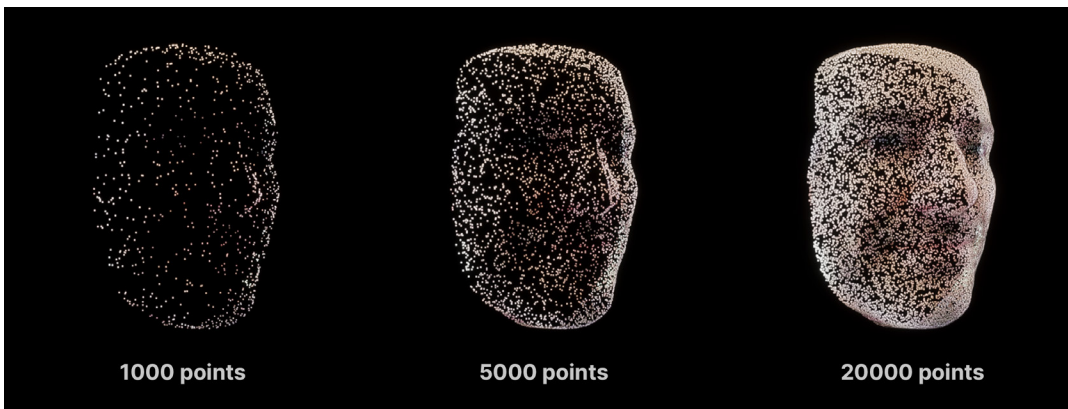
Point Cache ベイクツール

Point Cache ベイクツールを使用して、入力メッシュまたは 2D テクスチャから Point Cache を作成します。



Point Cache ベイクツール

Point Cache には、3D メッシュや 2D テクスチャから生成されたポイントのリストが格納されますが、実際のジオメトリは格納されません。ベイクング中、ポイントをより均一に分離し、オーバーラップの数を減らすために、追加のフィルター処理を行います。**Mesh** または **Texture** を選択し、適切な **Point Count** を設定して、**Save to pCache file** を選択します。



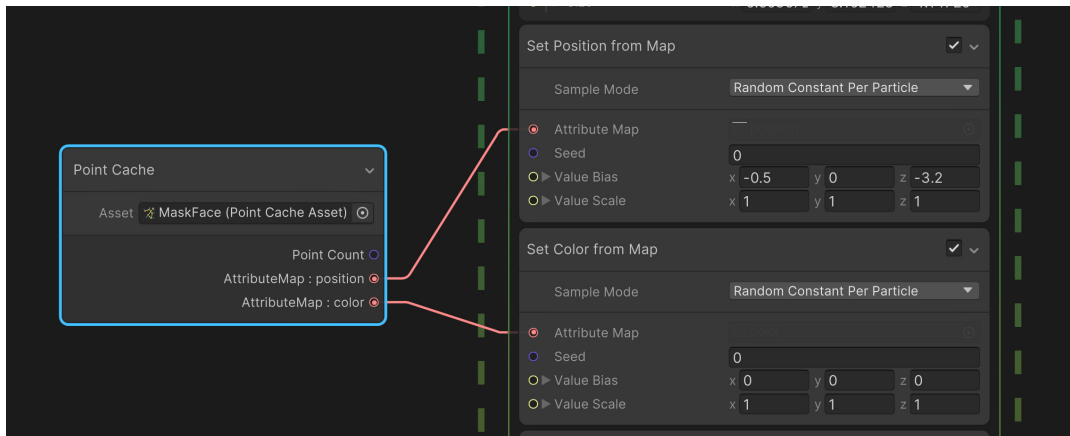
さまざまなサンプルを使用した Point Cache の視覚化

Point Cache は、[Stanford PLY](#) ファイルと似ていますが、.pCache ファイル形式ではポリゴンが削除され、ベクトルに対するサポートが追加されています。そのため、Python や C# ではより読みやすく、書きやすいものになっています。

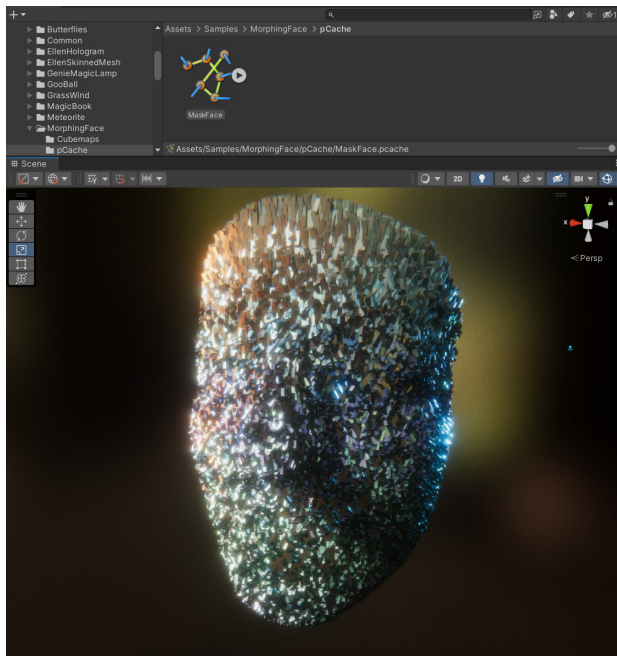
Point Cache の使用

VFX Graph では、[Point Cache オペレーター](#) が Point Cache アセットからパーティクルの数とその属性を抽出します。その後、それらをオペレーターの出力ポートとして公開します。

Morphing Face サンプルを見ると、オペレーターは Point Count 用の出力スロット 1 つと、Attribute Maps 用のテクスチャスロット 1 つを作成しています。出力は、[Set Attribute from Map ブロック](#)など、ほかのノードに接続できます。



VFX Graph で Point Cache を使用する



Point Cache アセットに適用された Morphing Face エフェクト

詳細については、[Point Cache のドキュメント](#)を参照してください。

MaskFace Point Cache は、エフェクトの基本構造を制御します。

符号付き距離フィールド

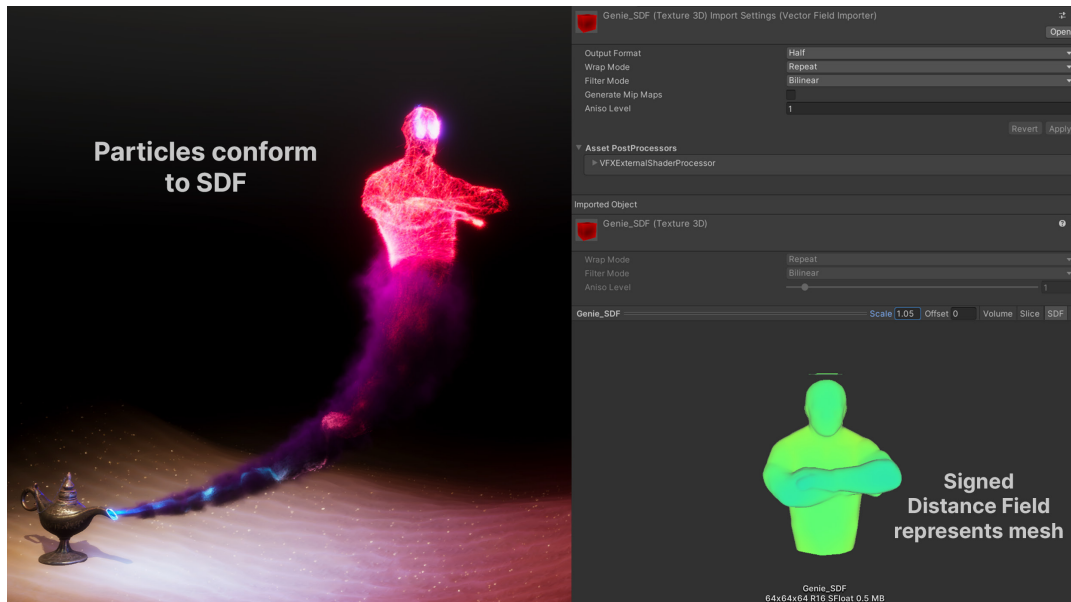
符号付き距離フィールド (SDF) は、メッシュジオメトリの 3D テクスチャ表現です。各テクセルには、メッシュの表面に最も近い距離値が格納されます。

決めごととして、この距離はメッシュの内側ではマイナス、外側ではプラスです。これにより、サーフェスの任意の点、ジオメトリの境界の内側、またはそこからの任意の距離にパーティクルを配置できます。

SDF の計算は Point Cache よりもリソース消費が多くなりますが、追加機能を提供することもできます。メッシュが非常に細かい場合はテクスチャの解像度を高くする必要があり、一般的にメモリをより多く消費します。

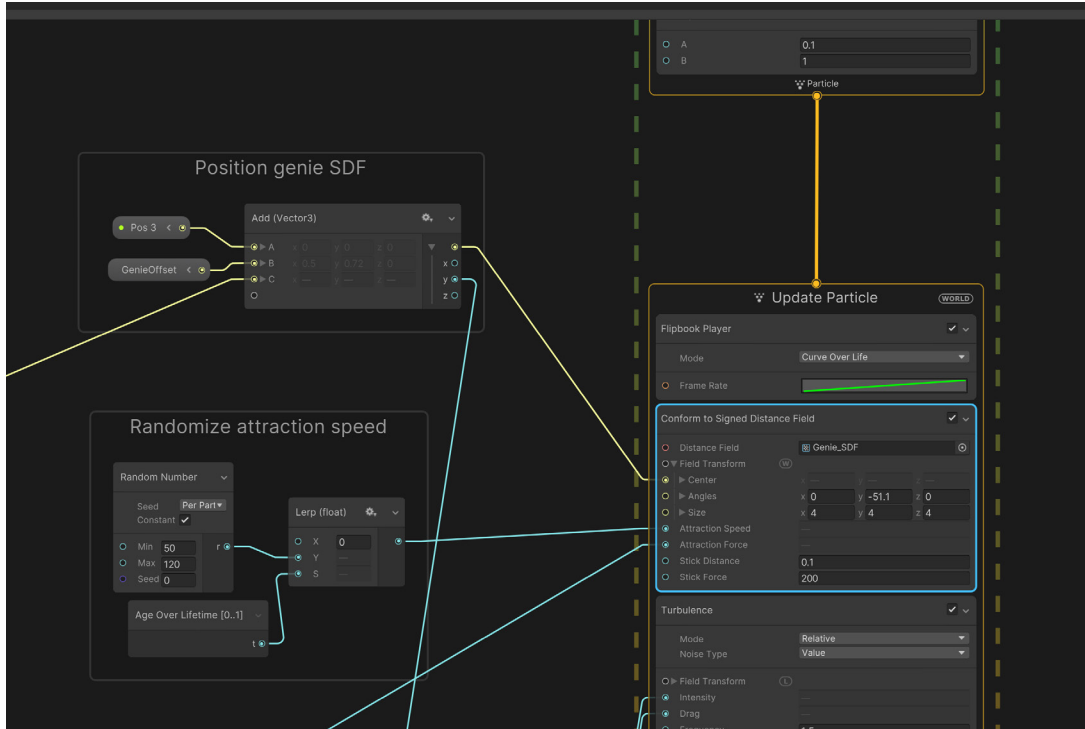
SDF の使用

ビジュアルエフェクトでは、SDF を使用してパーティクルを配置したり、パーティクルを形状に合わせたり、パーティクルと衝突させたりすることができます。Magic Lamp のサンプルでは、魔人本体に SDF が使用されています。SDF アセットのプレビューは次のようになります。

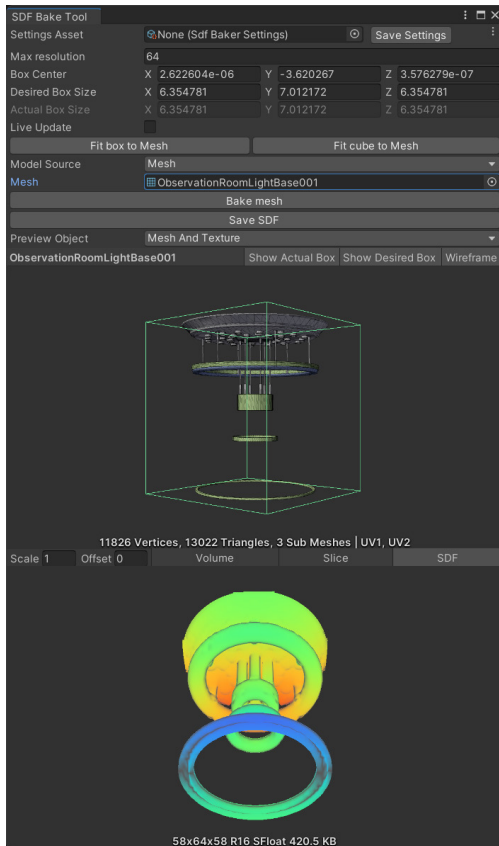


符号付き距離フィールドの視覚表現

SDF アセットを 3D テクスチャ **Volume File (.vf)** としてインポートします。互換性のある VFX Graph ブロックとオペレーターを使用して、パーティクルシステムがサンプリングされた点と相互作用するようになります。この例では、**Conform to Signed Distance Field ブロック**が、魔人の胴体が現れる領域にパーティクルを引き付けます。



VFX Graph で SDF を使用する



SDF バイクツール

SDF バイクツール

SideFX Houdini などの外部 DCC ツールを使用して SDF を作成できます。[VFXToolbox](#) ユーティリティは、**Houdini Volume Exporter** から Volume File をバイクできます。

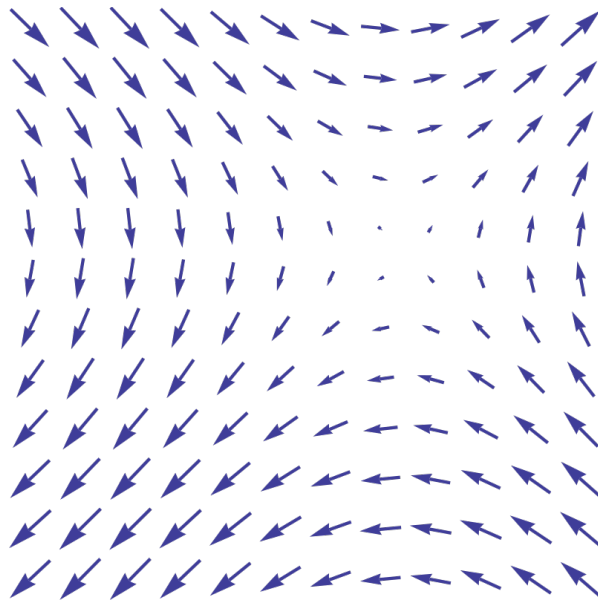
Unity にはこのプロセスを簡略化するユーティリティが含まれています。SDF バイクツール (**Window > Visual Effects > Utilities > SDF Bake Tool**) は、入力メッシュ (または1つのプレハブ内の複数のメッシュ) を取得し、その 3D テキスチャ表現を生成します。

ランタイム時や Unity エディターで [SDF Bake Tool API](#) を使用して、SDF をベイクすることもできます。ランタイムベイクは多くのリソースを消費することに注意してください。ベストプラクティスは、低解像度の SDF を使用し、n フレームごとにのみ処理することです。

SDF を生成して使用方法の詳細については、[Signed Distance Fields in the VFX Graph](#) (VFX Graph の符号付き距離フィールド) を参照してください。また、[このリポジトリ](#)で追加のサンプルを見つけることもできます。

ベクトル場

ベクトル場は、パーティクルの速度や加速度を制御するベクトルの均一なグリッドです。矢印は各ベクトルを表します。ベクトルのサイズが大きいほど、パーティクルの移動速度が速くなります。



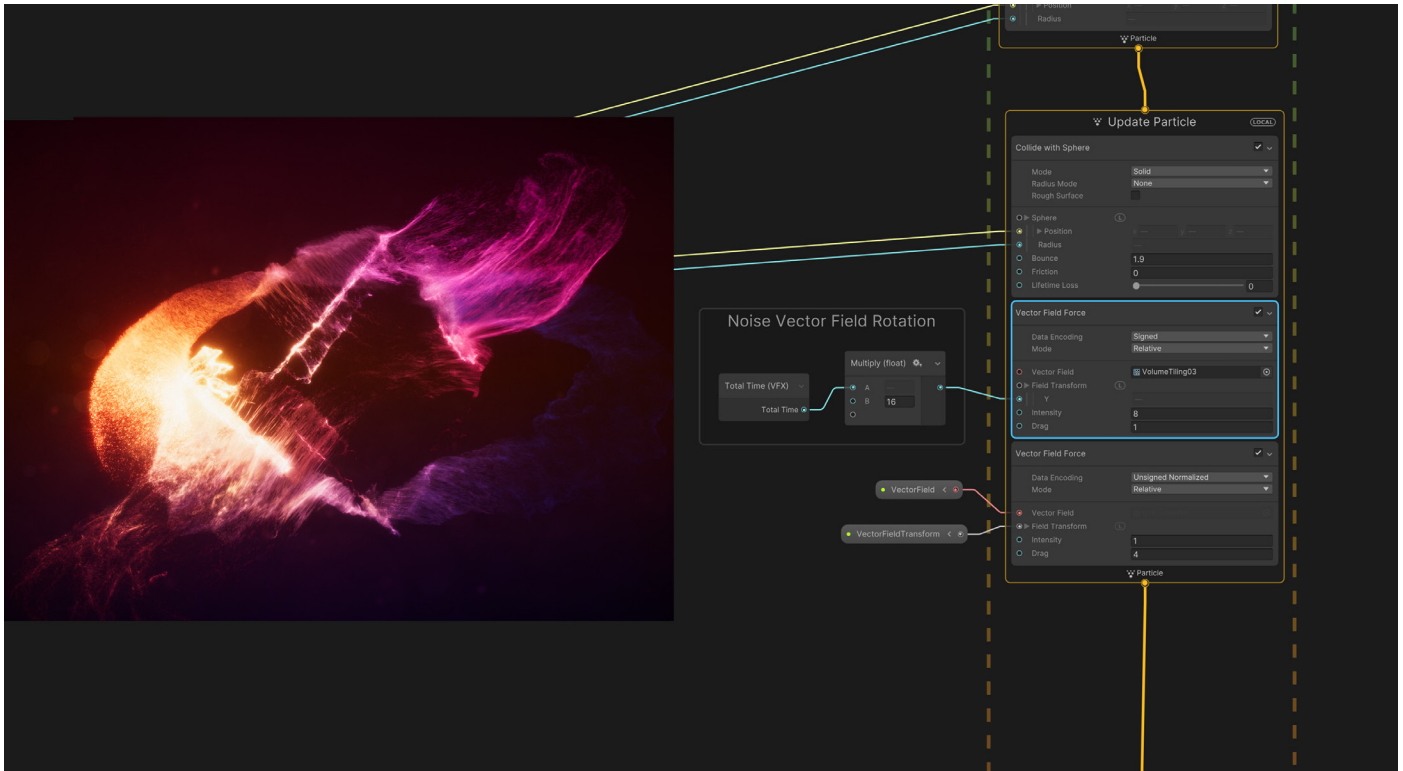
2D ベクトル場 (出典 :[Wikipedia](#))

SDF と同様に、オープンソースの [Volume File \(.vf\)](#) 形式を使用してベクトル場を表すことも、[VFXToolbox](#) にバンドルされている Houdini VF Exporter からベクトル場を生成することもできます。Volume File 仕様に従った独自の VF File Exporter を作成することもできます。

UnityLogo シーンでは、パーティクルは目に見えないベクトル場の流れに押されるように流れます。

TFlow (Asset Store)

Unity Asset Store で入手できる [TFlow](#) は、“あなたのフリップブックの実用性と品質を高めるモーションベクターとモーションブラージェネレーター”です。[Built-In -Particle System](#) と [VFX Graph](#) の両方で使用できます。



ベクトル場が UnityLogo のエフェクトを生み出す。

VFXToolbox

VFXToolbox には、Unity ビジュアルエフェクトアーティスト向けの追加ツールが備わっています。SideFX の **Houdini Point Cache Exporter** および **Volume Exporter** から .pCache および .vf ファイルをエクスポートできます。

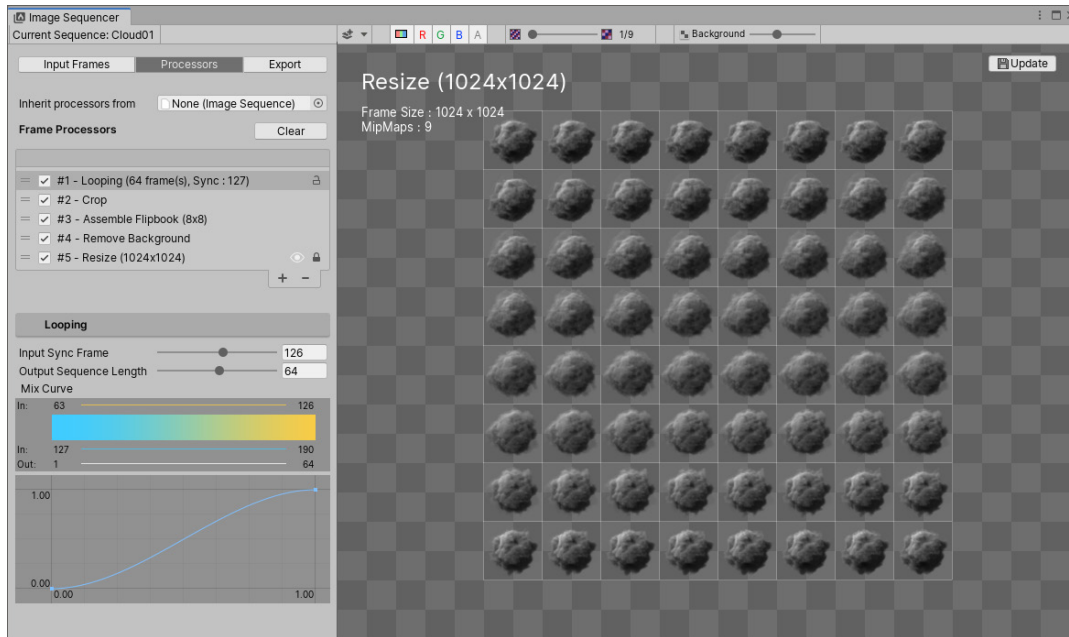
[GitHub](#) のリポジトリからダウンロードし、Package Manager を使用してインストールしてください。

Image Sequencer

フリップブックテクスチャシートを使用すると、アニメーション化されたエフェクトをスプライトにバイクできます。煙、炎、爆発などのエフェクトをシミュレートするフレームバジェットがない場合は、画像をフリップブックテクスチャシートとして保存することで、コストをかけることなく同等の“バイク”エフェクトを生成できます。

まず、Unity または別の DCC パッケージを使用して、エフェクトのイメージシーケンスをプロジェクトフォルダーにレンダリングします。次に、Image Sequencer を使用して個々の画像を 1 つのテクスチャシートに変換します。好みに合わせて時間を調整してループさせ、[Flipbook Player ブロック](#)で画像を再生します。

このツールで作成された[サンプルフリップブック](#)をいくつかご確認ください。



フリップブックテクスチャの例

デジタルコンテンツ制作ツール

Unity はゲーム開発のための中心的なツールですが、唯一のツールではありません。リアルタイムのビジュアルエフェクトの作成は複雑な場合があるため、Unity 以外のデジタルコンテンツ制作 (DCC) ソフトウェアの専門的なサポートを検討することもあるでしょう。Unity と VFX Graph を補完するために多くのアーティストが使用しているツールをいくつか紹介します。

SideFX Houdini

Houdini は長い間、シミュレーションとビジュアルエフェクトのための業界標準のツールです。プロシージャルワークフローとノードベースのインターフェースにより、比較的少ない手順でテクスチャ、シェーダー、パーティクルを生成できます。オペレーター中心の構造により、ノンリニア開発が促進され、3D 制作の主な領域がすべて網羅されています。

Autodesk Maya

Maya は多くのゲーム開発スタジオの基盤を強化しています。比較的新しい Bifrost システムにより、ビジュアルプログラミング環境で物理的に正確かつ驚くほど詳細なシミュレーションを作成できます。



Blender

Blender は無料のオープンソースの 3D 制作スイートです。その機能は、モデリングやリギングからアニメーション、シミュレーション、レンダリングまで、3D 制作のあらゆる側面をカバーします。Blender はクロスプラットフォームであり、Linux、Windows、macOS で同じ動作をするので、コミュニティからの幅広い支持を受け続けています。

Adobe Photoshop

ここで紹介した他の 3D ツールとともに、Adobe Photoshop などの画像編集ソフトウェアを活用すると便利です。Photoshop を使用して複数のレイヤーでラスター画像を編集および作成し、[マスク](#)、[アルファコンポジット](#)、いくつかの[色 モデル](#)をサポートします。Photoshop では、これらの機能に対応するために独自の PSD および PSB ファイルフォーマットを使用しています。

もちろん、これらは利用できる DCC ツールの一部でしかありません。グラフの作成を開始すると、フリップブックテキストチャやメッシュなど、ビジョンを実現するためのあらゆるものを揃えるのに、これらが役立ちます。

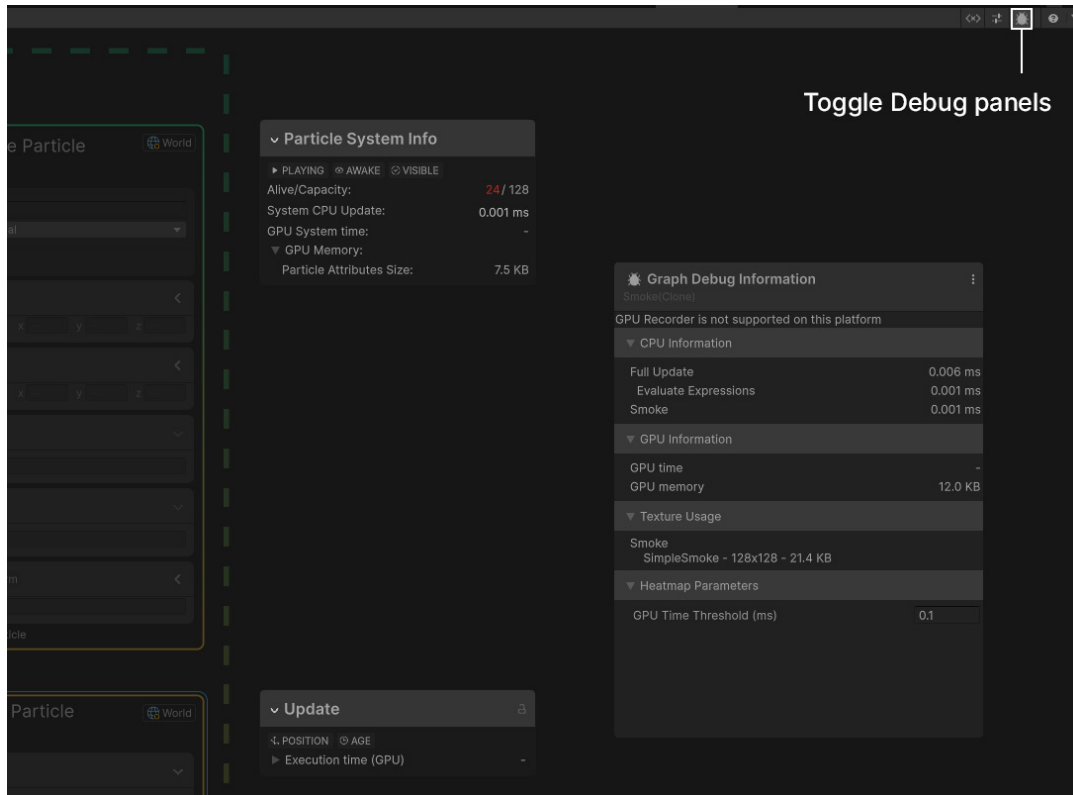
最適化

VFX Graph でじっくり作業した後、プログラマーがコードをプロファイリングしてパフォーマンスをチェックするのと同じように、それらを再整理して最適化したいと思うでしょう。エフェクトが正しく表示されたら、最終的なゲームやアプリケーションにデプロイする前に、余分なリソースを使用していないことを確認してください。

VFX Graph の Profiling パネルと Debug パネル

Unity 6 には、実行中のビジュアルエフェクトに関する重要な情報を提供する Profiling パネルと Debug パネルがあります。これらのツールは、CPU と GPU のタイミング、メモリの使用状況、テクスチャの使用状況、さまざまな状態などの情報を提供します。VFX Graph のパフォーマンスを監視し最適化するために使用できます。

Visual Effect Graph ウィンドウの右上にあるデバッグアイコンをクリックします。これらのパネルは、VFX Graph ウィンドウが Visual Effect コンポーネントを持つゲームオブジェクトにアタッチされている場合にのみ機能することに注意してください。



Profiling パネルと Debug パネルを使用して VFX Graph を最適化する。

Graph Debug Information パネル

Graph Debug Information パネルには、以下の情報が表示されます。

- **CPU Information:** CPU Information パネルには、グラフ全体の完全な更新時間 (ミリ秒単位)、CPU で計算されたグラフパラメーター評価の所要時間、グラフ内の特定のシステムの更新時間 (ミリ秒単位) など、重要なパフォーマンス指標が表示されます。
- **GPU Information:** GPU Information パネルには、GPU での VFX Graph の実行時間 (ミリ秒単位) と GPU メモリの使用状況が表示されます。

- **Texture Usage:** 使用されているテクスチャ、その次元、メモリサイズが一覧表示されます。
- **Heatmap Parameters:** GPU 時間しきい値を調整して、グラフの高価な部分をハイライトします。

右上の省略符号 (:) メニューから **Rendering Debugger** (URP/HDRP) と **Unity Profiler** にアクセスすることもできます。

Particle System Info パネル

Particle System Info パネルは、各システムの Initialize コンテキストにアタッチされています。このパネルには、特定のシステムに関連する情報が表示されます。

- **Particle state:** これは、システムが再生中 / 一時停止中、起動中 / スリープ中、表示中 / カリング中であることを示します。
- **Alive/capacity:** これは、生存しているパーティクルの数と、Initialize コンテキストでユーザーによって設定されたキャパシティを示します。このキャパシティを、動作中のパーティクルの最大数に合わせて最適化すると、メモリ割り当てスペースを節約できます。

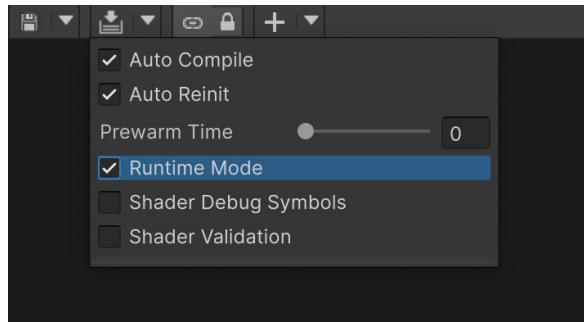
Context Debug パネル

これらのパネルは、コンテキストを選択すると表示されます。右上にある錠前のマークを使用して画面上にピン止めします。これらのパネルには、コンテキストごとに異なる情報を表示できます。

- **Spawn コンテキスト :** これは Spawner が再生中か停止中かを示します。
- **Initialize、Update、Output コンテキスト :** これらのパネルは、Update コンテキストに暗黙的に更新された属性を表示し、GPU の実行時間をタスク別に分け、テクスチャの使用状況をそれらの次元やメモリサイズとともに一覧表示します。

このパネルのプロファイリングのタイミングは比較用であり、エディターに記録されたものであるため、正確な測定値ではないことに注意してください。ボトルネックを特定し、パフォーマンスを最適化するために使用できます。

さらに、Toolbar の Runtime Mode を有効にすることで、ランタイムコンパイルを強制することもできます。これにより、エフェクトで、オーサリングバージョンではなく、最適化されたランタイムバージョンが使用されるようになります。エフェクトを完全に再編集可能にしたい場合は、このオプションをオフにします。



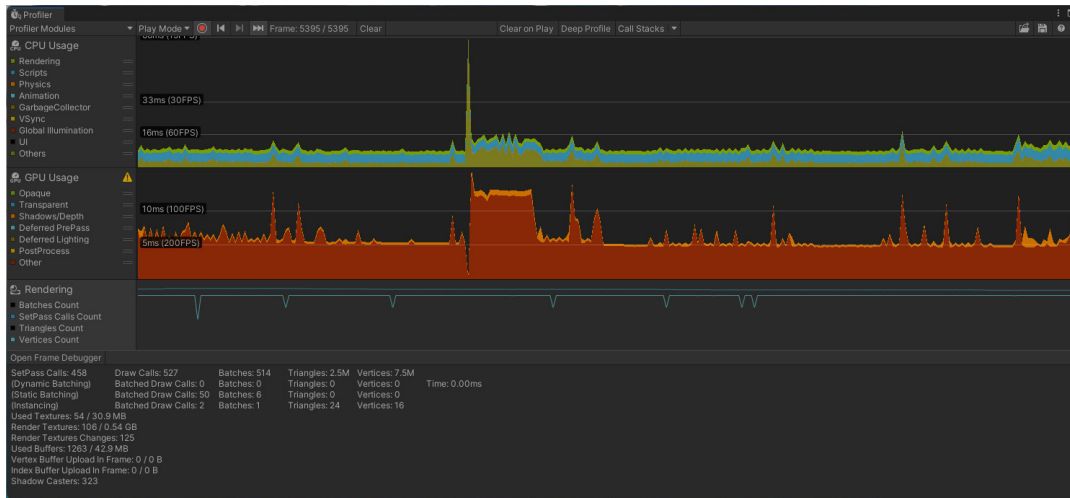
Toolbar の Runtime Mode を有効にする。

注 : Apple Silicon では GPU 実行タイミングを利用できず、プロファイリングパネルではアタッチされたビジュアルエフェクトのインスタンスングが無効になります。パフォーマンスを正確に評価するために、Unity プロファイラーを使用して常にターゲットデバイス上でプロファイリングを行ってください。

Unity Profiler と Frame Debugger

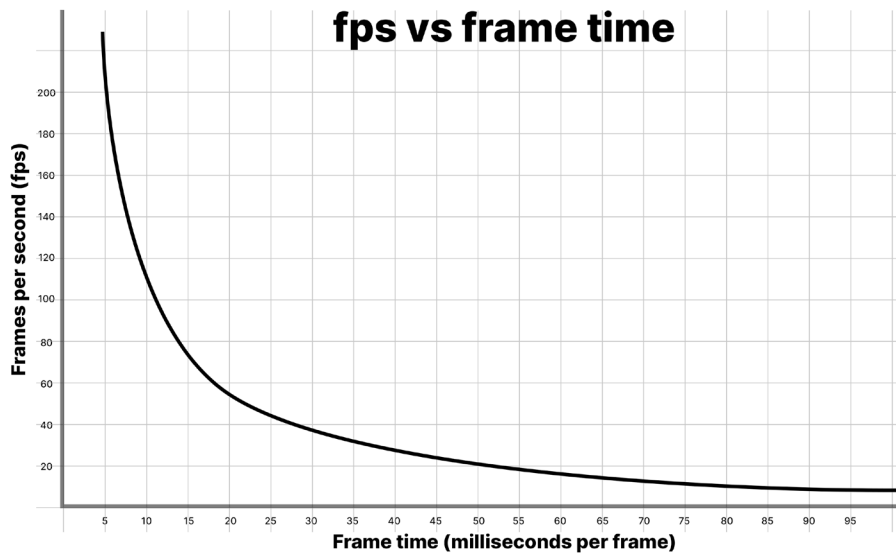
Unity Profiler (Window > Analysis > Profiler) と **Frame Debugger** (Window > Analysis > Frame Debugger) を使用して、グラフを最適化し、パフォーマンスを強化できます。ただし、Unity エディターはプロファイリング情報に影響を与え、結果が不正確になる可能性があります。

現実世界でのパフォーマンスを測定する必要がある場合は、**Profiler Standalone Process** オプションを使用するか、別のビルドを作成します。高いフレームレートを維持し、可能な限り最高の体験をプレイヤーに届けるために、[グラフィックスパフォーマンスの基礎](#) を考慮しましょう。



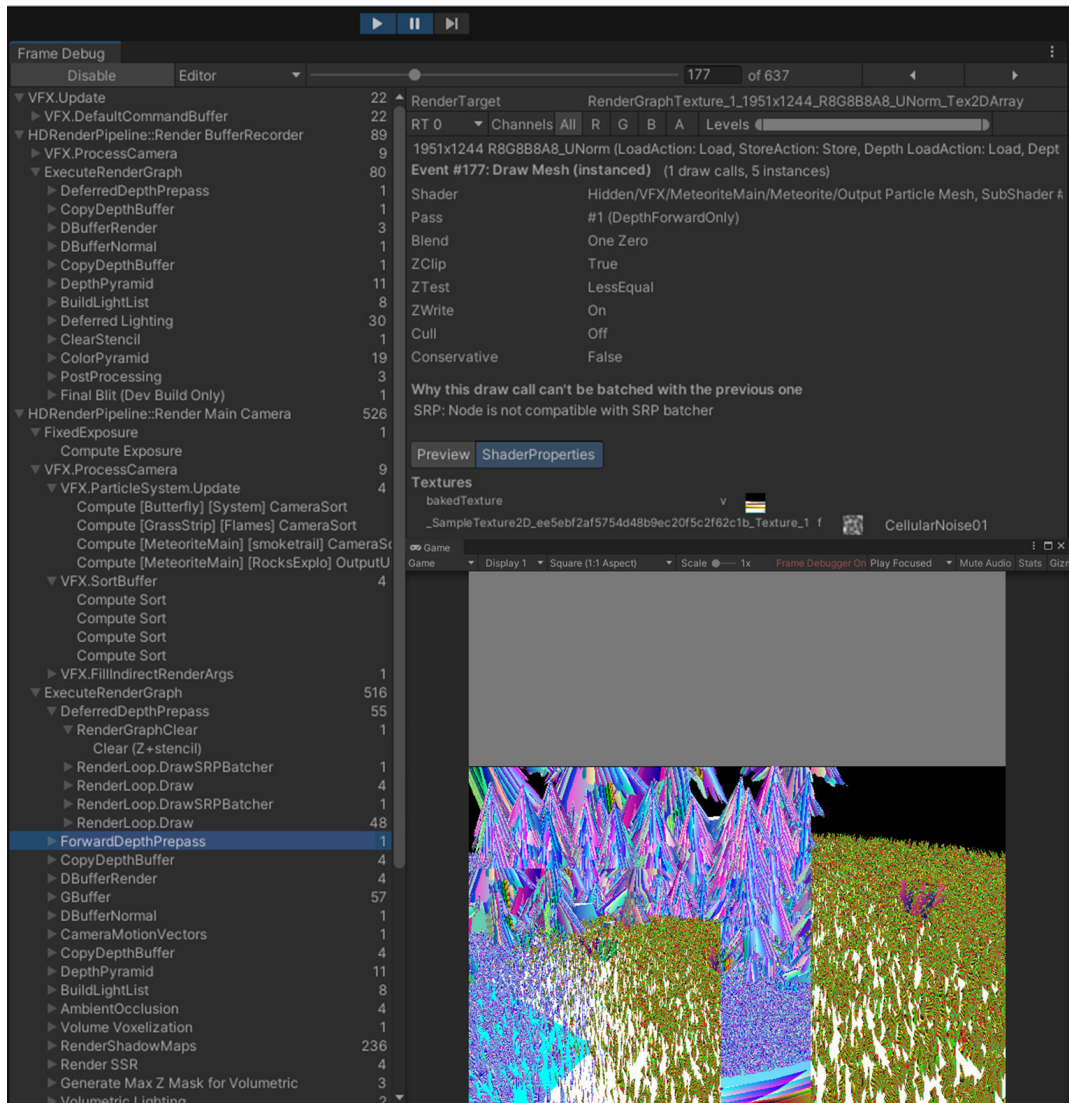
Unity Profiler

[レンダリング統計](#)を調べるときは、1 秒あたりのフレーム数ではなく、フレームあたりの時間コストに注目してください。fps は非線形であるため、ベンチマークとしては間違っって印象を与える可能性があります (下のグラフを参照)。60 fps を目指す場合は、フレームバジェットとして 1 フレームあたり 16 ミリ秒を使用してください (30 fps の場合は 1 フレームあたり 33 ミリ秒)。



フレーム時間を最適化の目安にする。

Frame Debugger にはドローコールの情報が表示されるので、フレームの構築方法を制御できます。



Frame Debugger

上の画像では、左側のパネルにドローコールのシーケンスとその他のレンダリングイベントが階層的に配置されています。一方、右側のパネルには、シェーダーパスやテクスチャなど、選択したドローコールの詳細が表示されます。これは、“フレーム探偵”として、Unity がリソースを消費している場所を見つけるのに役立ちます。

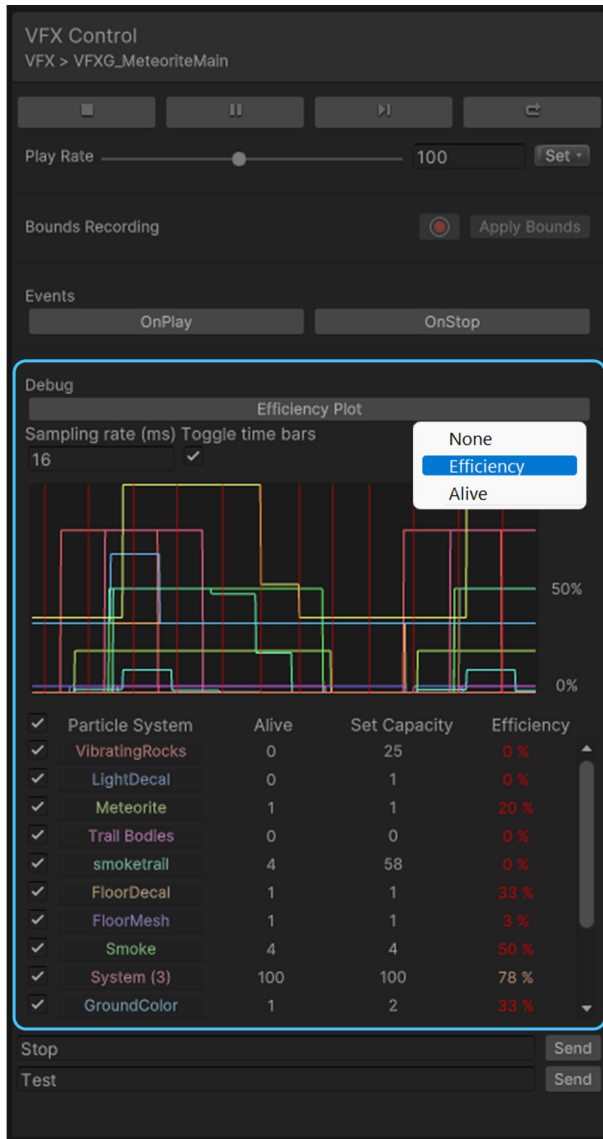
エフェクトを最適化するには、以下のよくある疑問点を調べるようにしてください。

- **テクスチャサイズ** : アセットがカメラに近づく場合は、その解像度を下げます。
- **キャパシティ** : パーティクルが少ないほど、使用リソースも少なく済みます。Initialize ブロックでキャパシティを設定し、システムのパーティクル最大数を制限します。
- **可視性と生存期間** : 通常、画面に何も表示されない場合はオフにします。

VFX Control の Debug モード

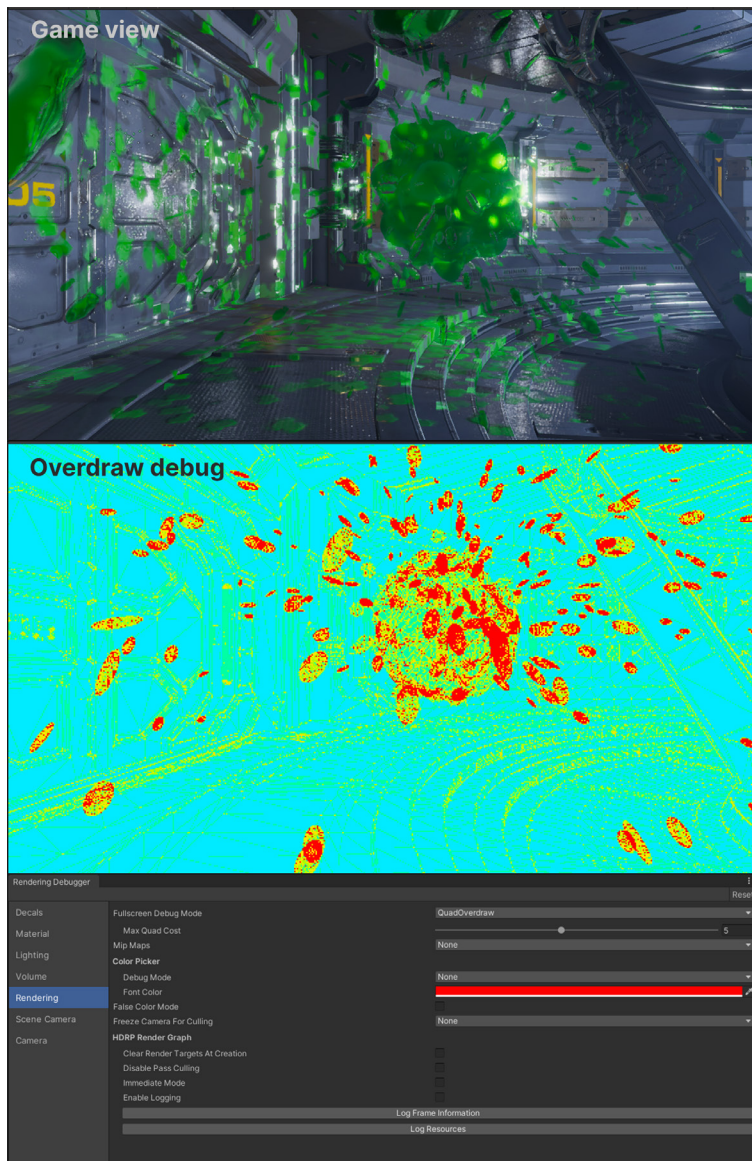
VFX Control パネルには、Debug モードがあり、パーティクルの生存期間とキャパシティを決定するために使用できます。これらはパフォーマンスとメモリ使用量に同様に影響を与える可能性があります。Inspector から Visual Effect インスタンスを編集し、Debug オプションを **Alive** または **Efficiency** に設定します。

結果として表示されるプロットは、アクティブなパーティクルの数や、その数とシステムで設定したキャパシティとの比較を示します。数とキャパシティの設定を調整して、VFX Graph の効率を改善します。



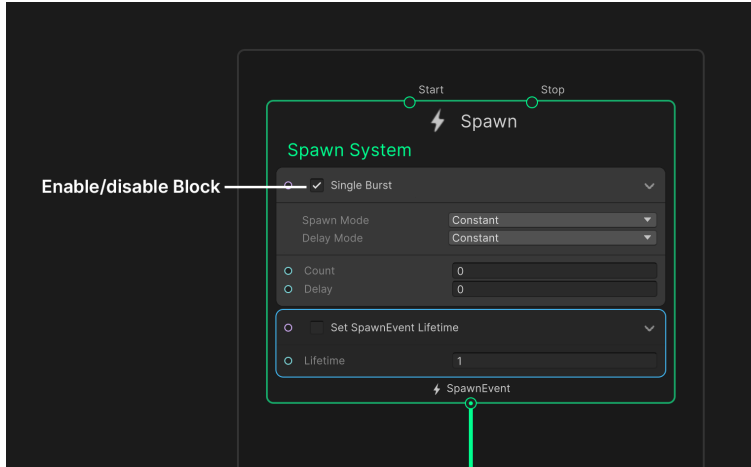
VFX Control の Debug モード

- **オペレーターとメモリ** : 不要なオペレーターを簡素化します。明らかな違いがなければ、イテレーションの使用回数を減らします。
- **フリップブック** : すべてのパーティクルをシミュレートする代わりに、特定のエフェクトをテクスチャフリップブックに事前にレンダリングすることを検討してください。その後、完全なシミュレーションが不要な場合であれば、アニメーション化されたテクスチャを再生します。
- **メッシュのサイズ** : パーティクルが Output メッシュの場合は、ポリゴン数を減らすようにしてください。
- **過剰なオーバードロー** : 透明なサーフェスが多数あると、レンダリングリソースを消費します。Rendering Debugger (**Window > Analysis > Rendering Debugger**) を使用して過剰なオーバードローをチェックし、それに応じてグラフを調整します。また、可能な場合は八角形のパーティクルに切り替えてください。



オーバードロー（赤）を最小限に抑えてパフォーマンスを改善

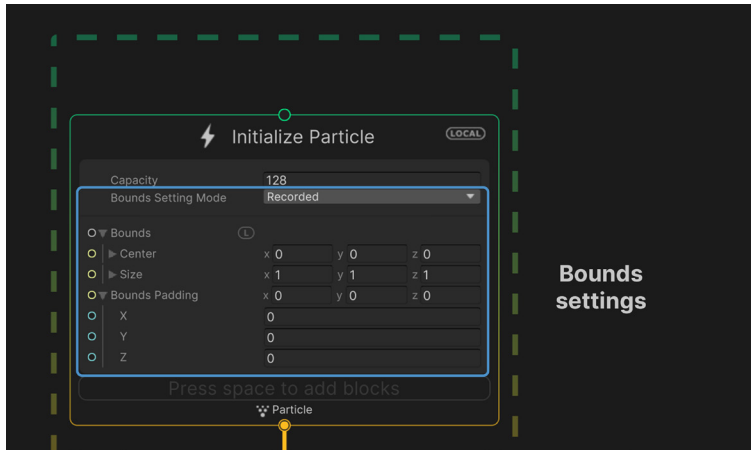
パフォーマンスのトラブルシューティングを行うときは、右上隅にあるチェックボックスで各ブロックを有効または無効にします。これにより、迅速な A/B テストを実施してパフォーマンス（適用前と適用後）を測定することで、グラフの一部を分離できます。完了したら、ブロックを Active 状態に復元することを忘れないでください。



テスト用ブロックを無効または有効にする。

バウンド

ビジュアルエフェクトのバウンドは、可視性に基づくビルトインの最適化で決まります。Initialize コンテキストで設定が表示されることに気付いているかもしれません。



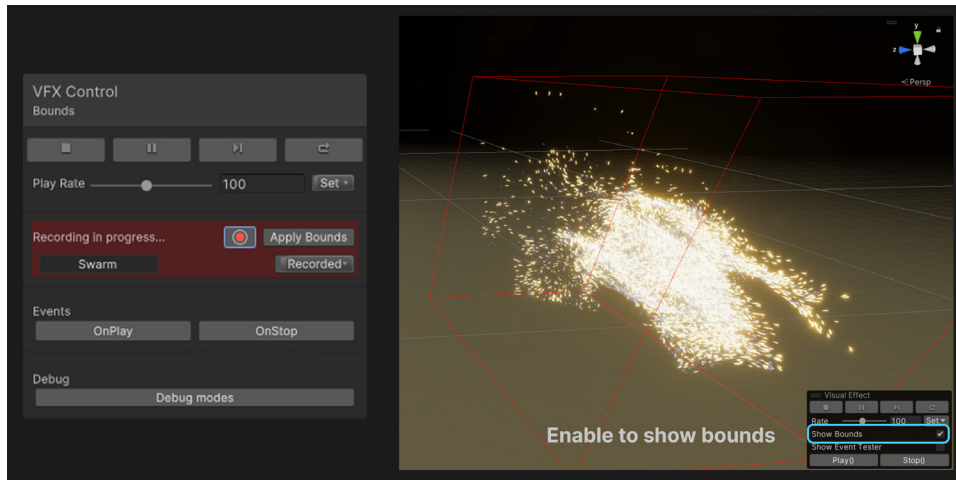
バウンド設定を使用して、エフェクトをレンダリングする場所を定義する。

カメラがバウンドを認識できない場合、Unity はエフェクトをカリングするため、レンダリングされません。以下のガイドラインに従って、各システムのバウンドを設定します。

- バウンドが大きすぎると、個々のパーティクルが画面外に出てもカメラはビジュアルエフェクトを処理します。これはリソースを浪費します。
- バウンドが小さすぎると、一部のパーティクルが画面に残っている場合でも、ビジュアルエフェクトが Unity によってカリングされる可能性があります。ポッピングが見えることがあります。

デフォルトでは、Unity は各システムのバウンドを自動的に計算しますが、Bounds Setting Mode を次のように変更できます。

- **Automatic (自動)**: Unity は、エフェクトが見えるようバウンドを調整します。このオプションが最も効率的ではない場合は、他のオプションのいずれかを使用してバウンドを最適化してください。
- **Manual (手動)**: バウンドと **Bounds Padding** を使用して、Initialize コンテキストでボリュームを定義します。これはシンプルですが、すべてのシステムに対して設定するには時間がかかります。
- **Recorded (記録)**: このオプションを使用すると、VFX Control パネルからバウンドを記録できます。記録時に赤で表示されるバウンドは、エフェクトを再生すると拡大します。**Apply Bounds** で境界寸法を保存します。



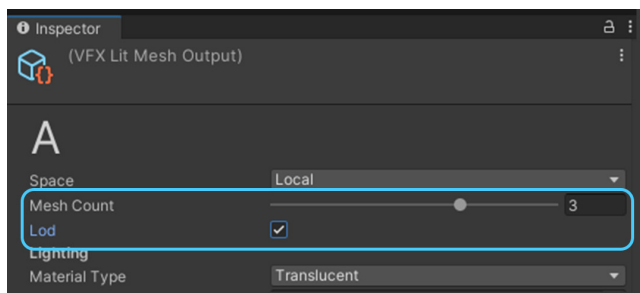
ランタイム時にバウンドを記録

ランタイム時にオペレーターを使用すると、**Manual** モードまたは **Recorded** モードで各システムのバウンドを計算できます。Initialize コンテキストには Bounds Padding 入力が含まれます。この Vector3 を使用してバウンドの値を大きくします。

メッシュ LOD

パーティクルがメッシュを出力する場合は、**Level of Detail** (LOD: 詳細レベル) を活用しましょう。ここでは、遠くにあるパーティクルのシンプルなメッシュを手動で指定できます。

Particle Mesh Outputs には、Inspector に表示される **Mesh Count** パラメーターがあります。このパラメーターでは、出力ごとに最大 4 つのメッシュを指定できます。これを LOD チェックボックスと組み合わせると、画面上に表示されるサイズに基づいてメッシュを自動的に切り替えることができます。



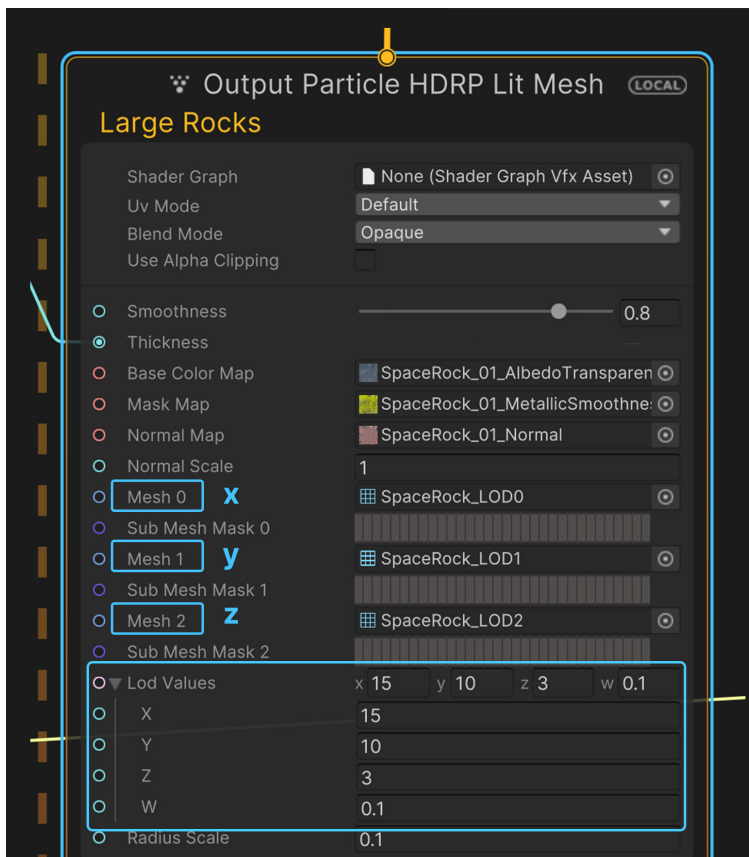
Inspector での Mesh Count と LOD の設定

Output コンテキストの LOD 値におけるスクリーンスペースの割合に応じて、解像度の高いモデルは解像度の低いモデルに引き継がれる可能性があります。



LOD の解像度

この例では、メッシュが画面の 15% 未満しか占めない場合、SpaceRock_LOD0 モデルが小さな SpaceRock_LOD1 モデルに切り替わります。



LOD 値



大量のメッシュパーティクルを作成する場合、何百万ものポリゴンをレンダリングする必要はありません。これにより、フレーム時間が大幅に短縮されます。

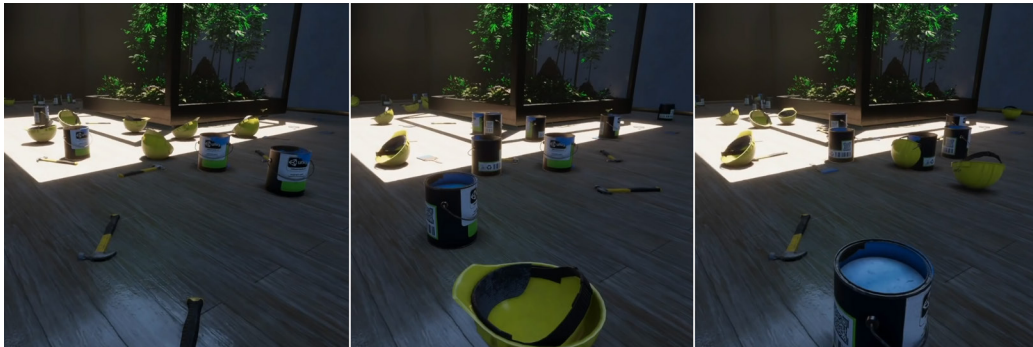


Mesh LOD はレンダリングリソースを節約する。

Mesh LOD を直接確認するには、[このプロジェクト](#)の PlanetaryRing の例をご覧ください。

Mesh Count

同様に、LOD なしで Mesh Count を活用できます。この場合、複数のメッシュを使用してランダム性を追加します。Output Particle Lit Mesh の 4 つの異なるメッシュは、床に散らばっているさまざまな小道具を作成します。

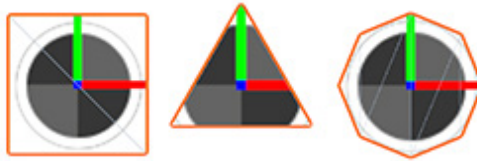


Mesh Count は、メッシュパーティクルをランダムにする。

パーティクルレンダリング

目標とするフレームレートとフレームバジェットを達成するには、パーティクルやメッシュをレンダリングするときに以下の最適化のヒントを考慮してください。

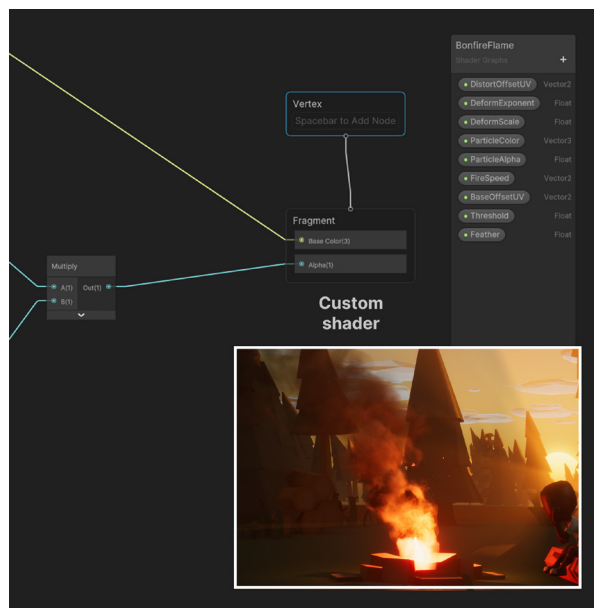
- **三角形のパーティクル** : クアッドパーティクルの半分のジオメトリなので、高速で動くエフェクトや大量のパーティクルのレンダリングに効果的です。
- **シンプルなライティング** : 完全な [Lit HDRP](#) シェーダーが必要ない場合は、リソース消費の少ないシェーダーに切り替えてください。Shader Graph の出力をカスタマイズして、特定のエフェクトに不要な機能を外すことができます。例えば、Bonfire のサンプルシーンでは、定型化された Shader Graph が使用されており、出力が大幅に簡略化されています。



Output Particle Primitive (クアッド、三角形、八角形)

以下のヒントは HDRP にのみ適用されます。

- **低解像度の透明度** : [HDRP Rendering](#) プロパティで Low Res Transparency を有効にすると、透明パーティクルの解像度を下げてレンダリングできます。これにより、多少のぼやけはあるものの、パフォーマンスが 4 倍に向上します。適切に使用すると、フル解像度でのレンダリングとほとんど見分けがつきません。
- **八角形パーティクル** : 八角形パーティクルは、クアッド (四角形) パーティクルの角を切り取った形です。パーティクルテクスチャの角が透明であれば、この手法でオーバードローを減らしたり防止したりできます。透明な領域を重ねる場合にも計算が必要なため、八角形を使用することで、クアッドの角が交差する部分の不要な演算を節約できます。



Visual Effect Samples の Bonfire のシーン

高度な機能

Graphics Buffer のサポート

グラフィックスバッファとコンピュートバッファにより、C# から VFX Graph への大量のデータの処理と転送が容易になります。

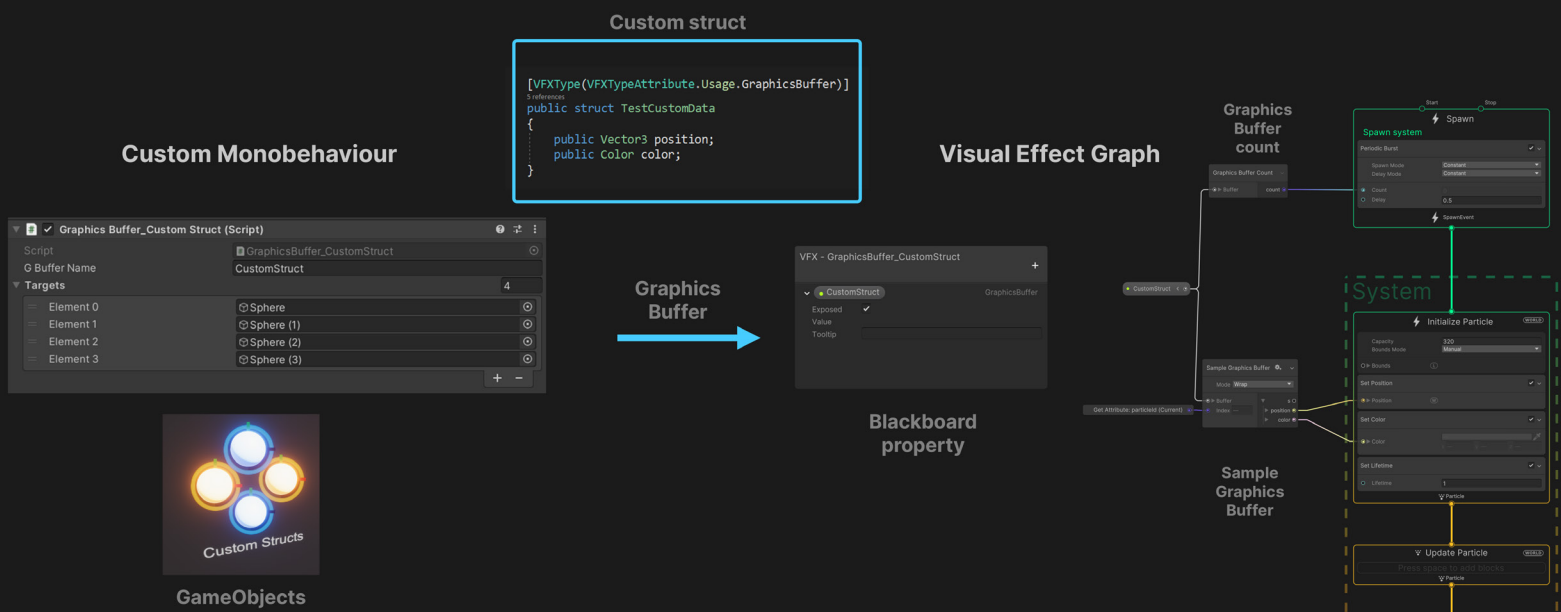
グラフ内の複数のゲームオブジェクトの位置を追跡している場合は、スクリプティングを通じてその情報をシミュレーションで利用できるようにすることができます。ここまでのサンプルのいくつかで見たように、Graphics Buffers がテクスチャ内のデータの格納を同じように置き換えることができます。

この例では、ゲームオブジェクトの位置と色のデータを、ビルトインの型、カスタム構造体、コンピュートシェーダーから VFX Graph に渡します。



Graphics Buffer は、ゲームオブジェクトのデータを VFX Graph に送信する。

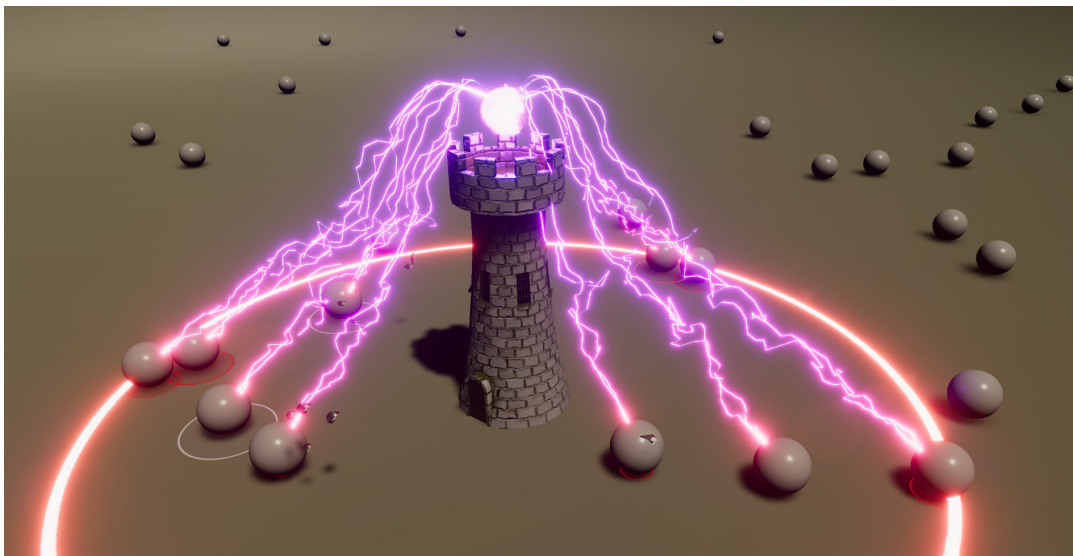
スクリプトを使用して Graphics Buffer を定義し、ゲームオブジェクトのデータで満たします。その後、Blackboard プロパティを使用して VFX Graph に渡します。



カスタム構造体を介してデータを渡す

この複雑な例では、Graphics Buffer を介して、帯電しているタワーから近付いているスフィアの位置にアクセスできます。多くのスフィアがある場合、すべてのゲームオブジェクトのプロパティを公開することは現実的ではありません。

Blackboard には、Graphics Buffer 内で使用するカスタム構造体が 1 つだけあります。このタワーは数百のターゲットに対応する可能性があり、少数のオペレーターだけでそのデータにアクセスできます。



Graphics Buffer の活用事例



これは、VFX Graph がシーン内でどのように相互作用するかを示すものです。ボイド、流体、髪の毛のシミュレーション、群衆などの複雑なシミュレーションを思い浮かべてください。Graphics Buffer を使用するには C# API の知識が必要ですが、ゲームオブジェクトとのデータのやりとりをこれまで以上に便利にします。

Unity で VFX Graph を使用して Graphics Buffer を使用するその他の例については、[このプロジェクト](#)をご覧ください。

カスタム HLSL

経験豊富な VFX アーティストや開発者が、[カスタム HLSL ブロック](#)を利用できるようになりました。この機能を使用すると、Unity ではまだネイティブサポートされていない独自のエフェクトを作成できます。カスタム HLSL を使用すると、高度な物理演算シミュレーション、群れ動作、リアルタイムデータビジュアライゼーションを作成できます。

カスタム HLSL ノードを使用すると、パーティクルシミュレーション中にカスタム HLSL コードを実行できます。コンテキスト内では、水平フローにオペレーター、垂直フローにブロックを使用できます。

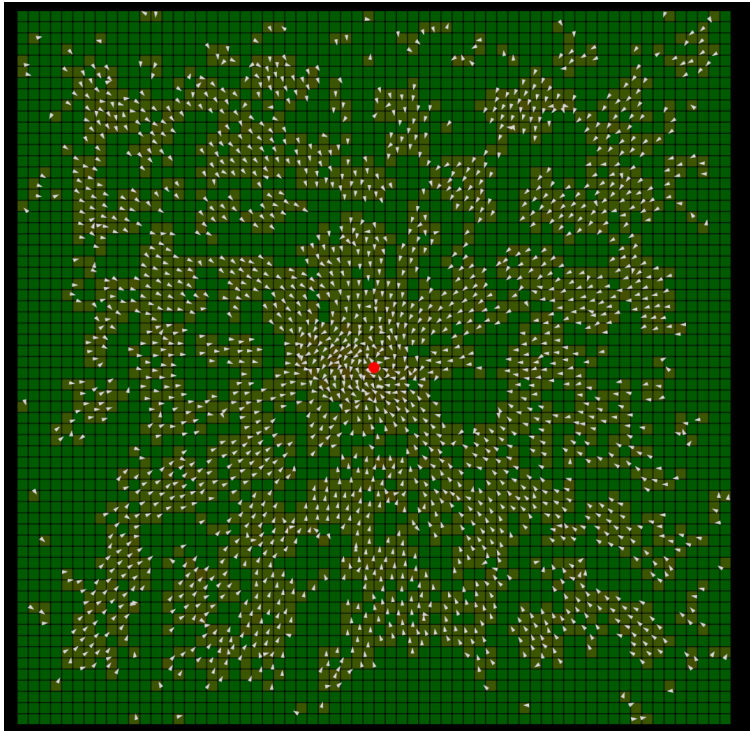
VFX Graph と互換性を持たせるには、カスタム HLSL ブロックが次の要件を満たす必要があります。

- 戻り値の型が `void` であること。
- アクセス修飾子が `inout` の `VFXAttributes` 型のパラメーターが 1 つ存在すること。

カスタム HLSL ノードを作成し、HLSL コードを埋め込むか、HLSL テキストファイルをソースとして使用します。要件の詳細については、この[ドキュメントページ](#)を参照してください。

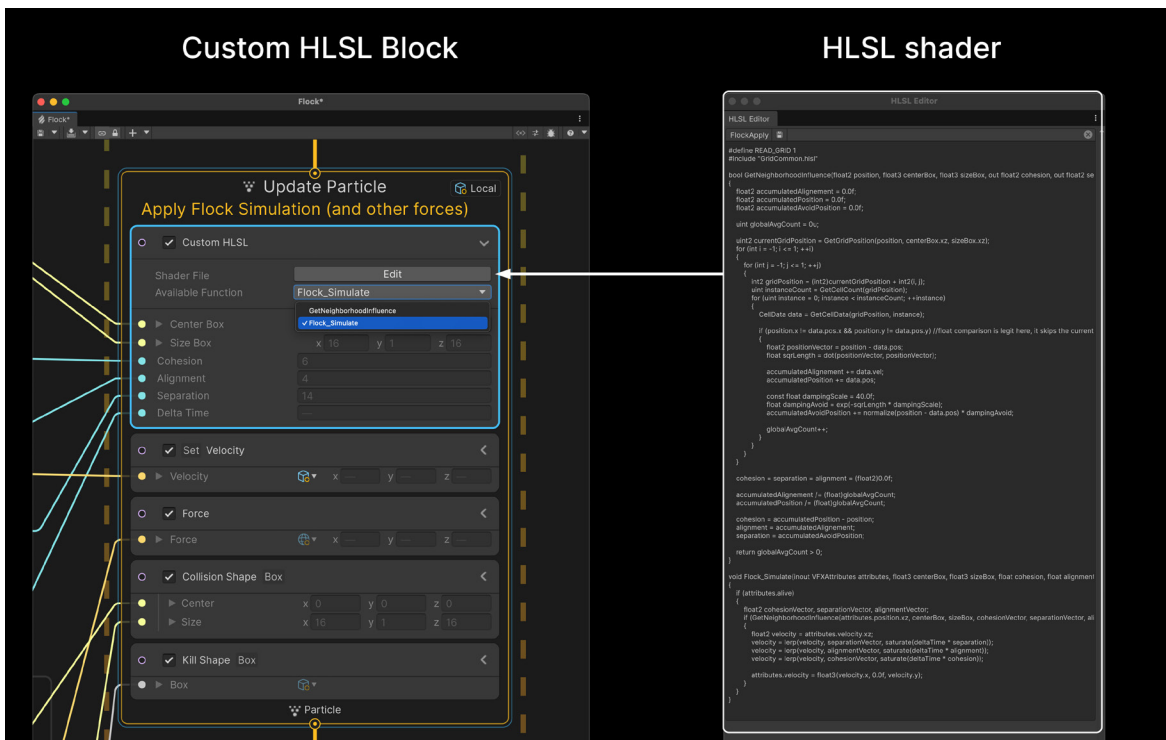
VFX Graph ノードの複雑なネットワークを使用するよりも、低レベルの HLSL コードを記述する方が効率的です。HLSL で特定の計算を直接実装することで、特に計算負荷の高いエフェクトのパフォーマンスを改善できる可能性があります。

カスタム HLSL は、数学関数、ノイズアルゴリズム、その他の手続き型手法に基づいてパーティクルの属性や動作を手続き型で生成するために使用できます。これにより、より有機的で多様なエフェクトを生み出すことができます。



カスタム HLSL はプロシージャル動作を作成できる。

このプロジェクトでは、カスタム HLSL ブロックと Graphics Buffers (GridManager スクリプトを参照) を使用して、近傍探索により、Visual Effect Graph で 2D のフロッキングをシミュレートする方法を示しています。

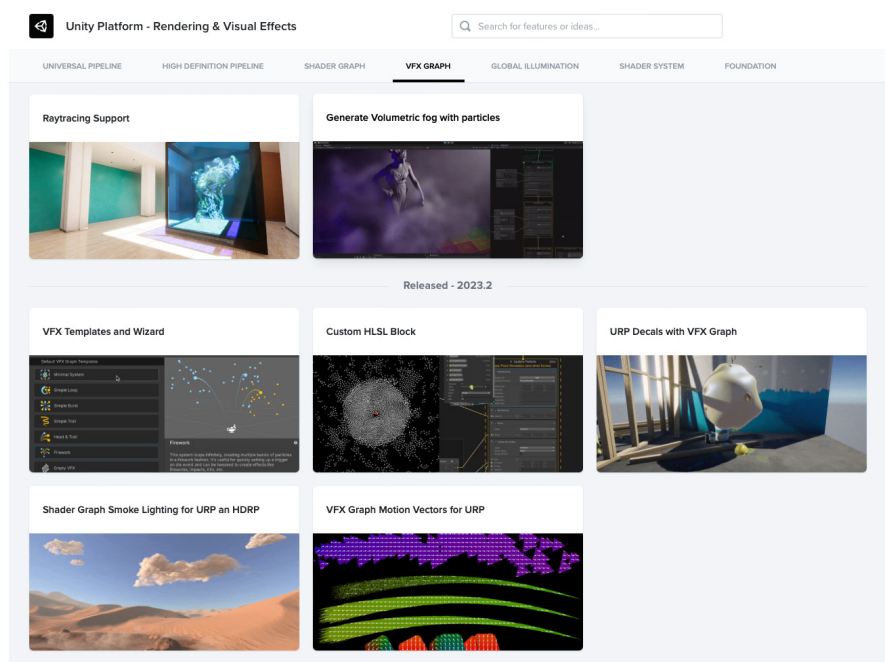


このカスタム HLSL は、近傍探索を実装してフロックを作成します。

追加リソース

このガイドが、VFX Graph と Unity のリアルタイムビジュアルエフェクトツールセットについて理解を深めるきっかけとなれば幸いです。なんと言っても、私たちのミッションは、すべてのクリエイターが芸術的ビジョンを実現できるよう支援することです。

VFX Graph なら、非常にリアルなシミュレーションと見事なグラフィックスでプレイヤーを魅了できます。皆さんの作品を楽しみにしています。



グラフィックス製品のロードマップ



VFX Graph を活用するための追加の学習用リソースを紹介します。

動画チュートリアル

- [Create amazing VFX with the VFX Graph \(VFX Graph で魅力的な VFX を作成 \)](#): ここでは、独自の VFX Graph を設定するための基礎の多くについて説明しています。
- [The power for artists to create \(アーティストが制作するパワー \)](#): このビデオでは、Mesh LOD、Graphics Buffer、Shader Graph の統合など、VFX Graph に最近追加された機能について説明しています。
- [VFX Graph:Building visual elements in the Spaceship Demo \(VFX Graph: 宇宙船デモにおけるビジュアル要素の構築 \)](#): このセッションでは、Spaceship Demo で使用されている手法をいくつか紹介しています。
- [Build a portal effect with VFX Graph \(VFX Graph によるポータルエフェクトの構築 \)](#):VFX Graph を使用してパーティクルのリングをより動的なエフェクトに変換することで、ポータルエフェクトを生成します。



GitHub 上の VFX プロジェクト

オーディオ反応エフェクト、データ可視化、ポイントクラウド、ボリュメトリックデータの再生など、VFX Graph の可能性を広げます。Unity のシニアクリエイターアドボケイトである高橋啓治郎が制作した以下のプロジェクトをご覧ください。

- カメラトラッキング + Lidar + VFX Graph: [こちら](#)と[こちら](#)を参照
- [Point Cloud \(pcx\) + VFX Graph](#)
- [Azure Kinect + VFX Graph](#)
- [Intel RealSense camera + VFX Graph](#)
- [DepthKit + VFX Graph](#)
- [DepthKit Volumetric video + VFX Graph](#)
- [4DViews Volumetric video + Alembic + VFX Graph](#)
- [Alembic + HAP video + VFX Graph](#)
- [Audio Reactive + VFX Graph](#)
- [VFX Graph + Midi controllers](#)
- [Skinned Mesh sampling](#)
- [SDF + VFX Graph](#)
- [Vertex Animation Texture \(VAT\) + VFX Graph](#)
- [Graphics Buffer test](#)
- [Render Geo data using Graphics Buffer + VFX Graph](#)
- [Depth of Field particle samples](#)
- [Procedural modeling using VFX Graph](#)
- [VFX Graph + Procedural ShaderGraph sprite generation shader](#)
- [VFX Graph test scenes](#)
- [VFX Graph interactive fireworks](#)
- [Sushi + VFX Graph](#)



unity.com