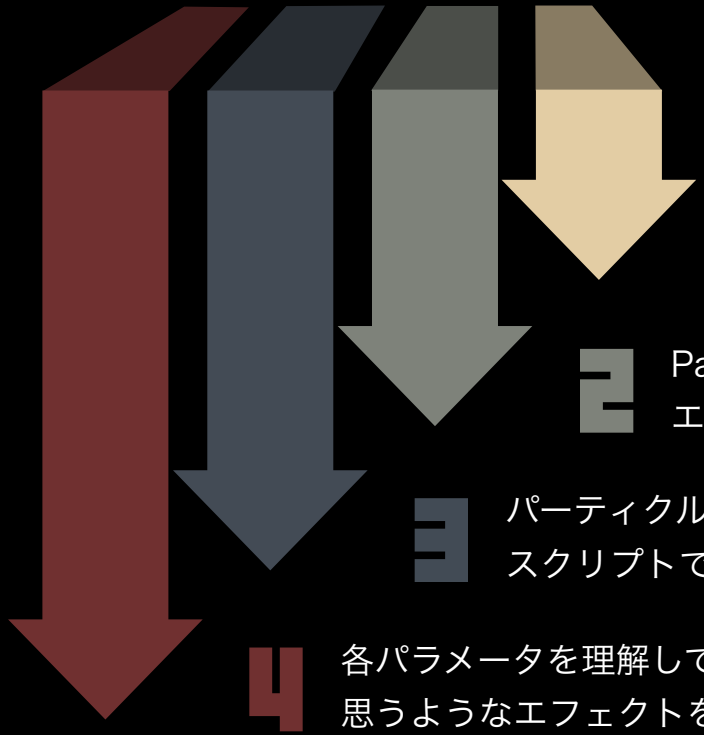
2016.2.21

パーティクル講座

ユニティ・テクノロジーズ・ジャパン
池和田 有輔

今回のゴール ～ マスターまでの道のり



1 Particle Systemの概要を理解している

2 Particle Systemの構造と各パラメータを理解し
エフェクトとして体を成すものを作ることができる

3 パーティクルのために用意されたShaderの特徴を理解し
スクリプトでできることを理解できている

4 各パラメータを理解してパーティクルを自由自在に扱い、
思うようなエフェクトを作ることができる

unity道場

Particle Systemの概要

Particle Systemの概要

そもそもパーティクルとは？

・主に小さな2D画像を大量にアニメーションさせて水、雲、炎などの自然現象をシミュレートするもの。
・ゲームでは魔法やレーザーなどのエフェクトを表現するために使われる機会が多い。

Particle System (Shuriken)の特徴

・Unity3.5が初出。
・Particle Systemの更新はマルチスレッドで行われ、それなりに高速。
・プレイしなくても確認・編集が可能（調整がとても楽）。
・Collisionの処理が扱える。
➡エフェクトのみならずゲームデザインに落とし込むことも可能に。

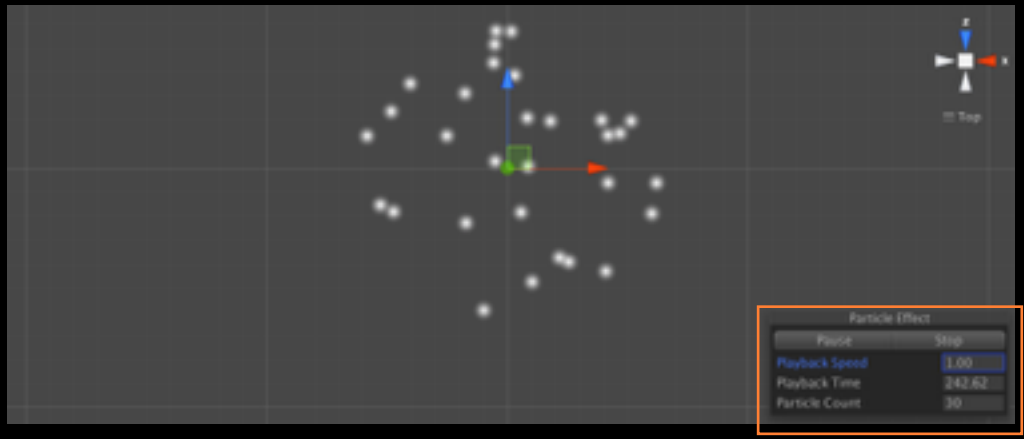
その他

・旧パーティクル系コンポーネントに属するEllipsoid Particle Emitter、Mesh Particle Emitter、Particle Animator、Particle Renderer、World Particle Colliderは特に理由のない限り使わない方向で。

Particle Systemことはじめ

とりあえず使ってみよう

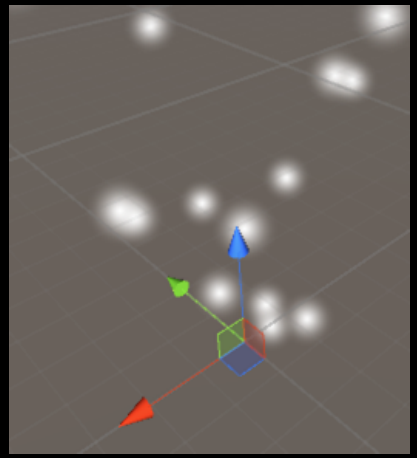
・シーンに配置すると白い玉が連続的に出続ける。これを元に編集していくのが基本。
・プレイしなくてもパーティクルは再生される。
・シーンビューのパネルで再生、一時停止、早送りなどもできるので活用しよう！

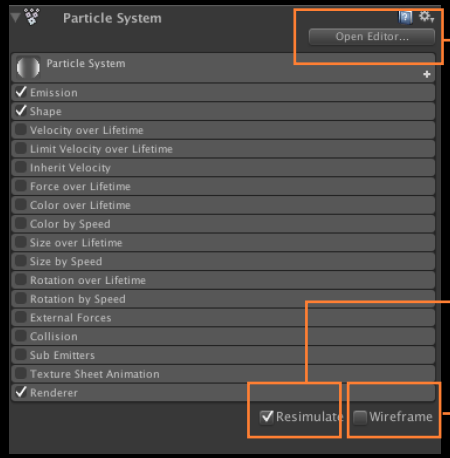


Particle Systemことはじめ

各セクションは「モジュール」と呼ばれている

・コンポーネントは多数の折りたたみ可能なモジュールによって成り立っている。





パーティクルを一括編集できるパネルを開く
複雑な構造のパーティクルを編集したい場合は
重宝するが、画面が狭い場合は得策ではない

リドローするものにのみ設定を反映

ワイヤーフレーム表示

各モジュールの詳細はマニュアルを参照

・英語：http://docs.unity3d.com/ja/current/Manual/PartSysMainModule.html
・日本語：http://docs.unity3d.com/Manual/PartSysMainModule.html

モジュールの中身を理解しよう

v5.3現在、Particle Systemは18のモジュールよって成り立っている

Main	Color Over Lifetime	Collision
Emission	Color By Speed	Sub Emitters
Shape	Size Over Lifetime	Texture Sheet Animation
Velocity Over Lifetime	Size By Speed	Renderer
Limit Velocity Over Lifetime	Rotation Over Lifetime	
Inherit Velocity	Rotation By Speed	
Force Over Lifetime	External Forces	

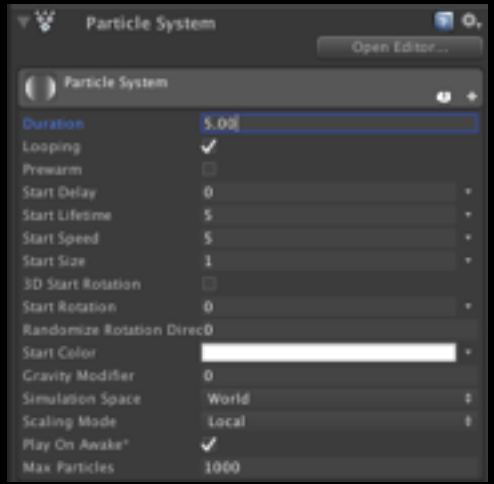
■ 基本　■ 移動　■ カラー　■ サイズ　■ 回転　■ イベント　■ アニメーション・描画

・上記のようにカテゴリー分けすることにより、必要なものにアクセスしやすくなるはずだ
・不要なモジュールはアクティブを切っておこう

モジュールの中身を理解しよう

Mainモジュール

- パーティクルの初期速度や大きさ、色など基本となる設定はここで。
- 調整時にはLoopingにチェックを入れておこう。
- Random Between Two ConstantsやCurveを使いこなそう。
  - ➡カーブデータはプリセットライブラリとして保存したり読み込んだりできるのでガンガン活用しよう！
    - ➡ただし素のAnimationCurveとは互換がない（拡張子からして違う）ので注意すべし。
- スピードはマイナスの値を入れることができる。
- 基本的にパーティクルの寿命は時間で決まる。移動距離などで消すにはCollisionを使うなどで解決しよう。
  - ➡Lifetime Loss を1にするとコリジョン判定のタイミングで即死する。



モジュールの中身を理解しよう

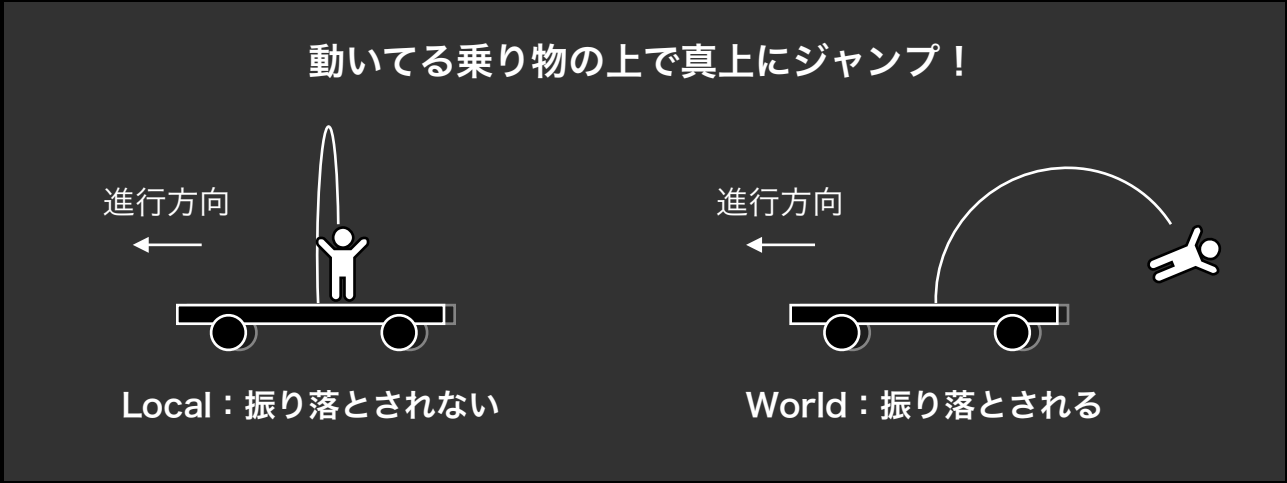
Mainモジュール（続き）

- 5.3での変更点として、3D Start Rotationにチェックを入れることにより、Render Modeがデフォルトのbillboard時、パーティクルの角度の変更が可能になった。これに伴いRotation over Lifetimeなど関連するパネルも変更された。
- さらにRandomize Rotation Directionに0-1の数値を入れることによって、確率による逆回転も可能に。

モジュールの中身を理解しよう

Mainモジュール（続き）

・Simulation Spaceは非常に重要な概念で、Localを選択するとパーティクルがオブジェクトを基点とした空間内を動きまわり、Worldを選択するとオブジェクトは無関係に動くようになる。
　・ポジションの変更は再生中でなくとも正しく反映されるが、ローテーションの変更は再生中のみ反映されることに注意。

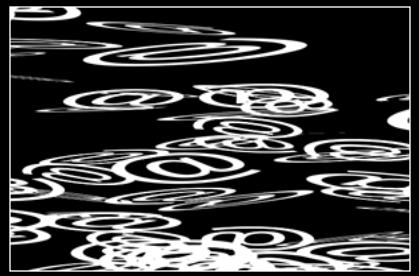


モジュールの中身を理解しよう

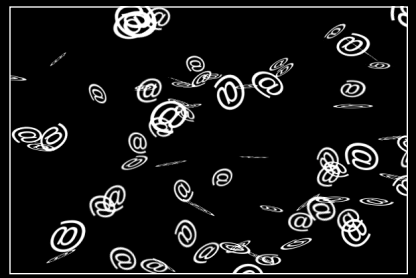
Mainモジュール（続き）

・5.3から待望のスケーリングオプション（Scaling Mode）が追加され、拡大縮小に対して次のオプションが選択できるようになった。
  ・Hierarchy: 絶対的なスケールに依存して全体を拡大縮小する。
  ・Local: 親のスケールに依存せずに全体を拡大縮小する。
  ・Shape: エミッションシェイプのみ拡大縮小し、個々のパーティクルの大きさは変化しない。
  ➡Start時のみ有効であり、動的に変えることはできない。

オリジナル



LocalでXを引き延ばした例



SpaceでXを引き延ばした例

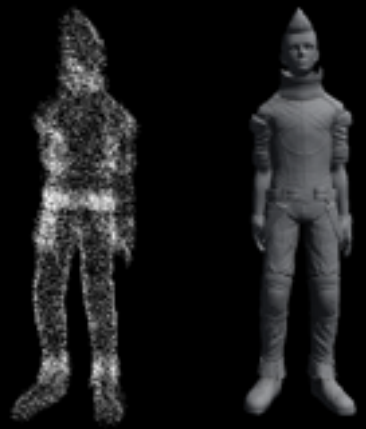
モジュールの中身を理解しよう

Emissionモジュール

・放出の頻度及びタイミングを決定する役割を担う。ほぼ有効化が必須。
・時間に依存して放出する「Time」と距離に依存して放出する「Distance」の2種類のオプションがある。
・「Distanceにしたのに出ない！」というときはSimulation SpaceをWorldにしてあるかチェックしよう。
・5.3からBurstsの放出量にもランダム性を持たせることができるようになった。

Shapeモジュール

・エミッターの形状や指定した形状のどこから放出するのかを決定する。
・Shapeの種類が増え、Skinned Mesh Rendererが使えるようになった。
➡キャラクターからオーラを出したりゴースト作ったりするのがとても簡単に。
・Sphereを指定してStart Speedをマイナスにすると、球の中心へと収束するエフェクトができる。



モジュールの中身を理解しよう

Velocity Over Lifetimeモジュール

・パーティクル個々に常に一定の速度を与える（等速運動）。
・進行方向にはMainモジュールのStart Speedを使うが、それ以外の方向に動かしたい場合などに用いる。

Limit Velocity Over Lifetimeモジュール

・減速させたい場合に使う。
・ここで入力するSpeedは、多くの場合空気抵抗と釣り合いが取れた速度、いわば終端速度（ターミナルベロシティ）となる。
・Dampenは減速率となる係数。

Inherit Velocityモジュール

・オブジェクト自体の速度をパーティクルに反映させる時に使う。
・MainモジュールのSimulation SpaceがWorldの時のみのオプションなので、反映されない時はチェックしてみるべし。

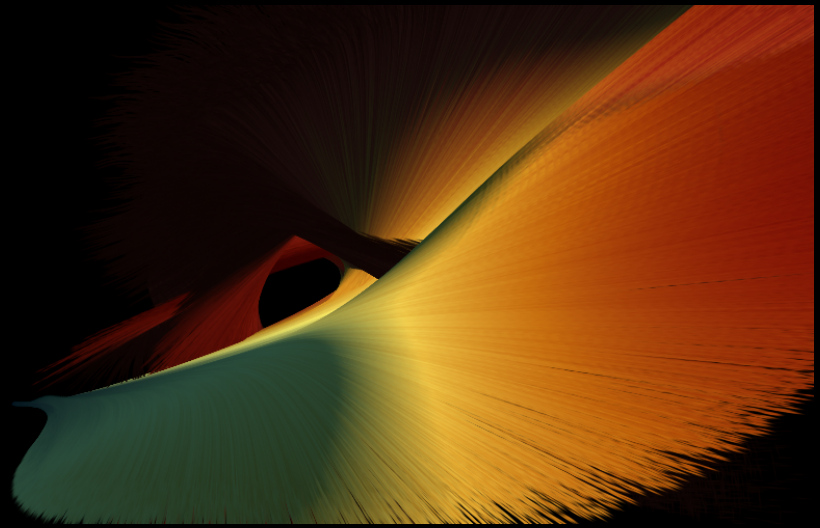
モジュールの中身を理解しよう

Force Over Lifetimeモジュール

・徐々に早くなるミサイルなど、パーティクル個々に加速度を与えたい場合に用いる。
・重力の表現としても使えるが、特に理由のない限り重力はMainモジュールのGravity Modifireを使おう。

External Forceモジュール

・Wind Zoneを使った場合に用いるもの。
・細かなコントロールには不向きだが、うまく
使えば様々な表現が可能。



モジュールの中身を理解しよう

Color Over Lifetimeモジュール

・時間によってに色を変える場合に用いる。
・上段でアルファを指定をし、下段でカラーの指定をする。
・補完のためのキーはそれぞれ最大8つまで登録可能。

Color By Speedモジュール

・早さによって色を変える場合に用いる。

Size Over Lifetimeモジュール / Size By Speedモジュール

・上記のスケール版

Rotation Over Lifetimeモジュール / Rotation By Speedモジュール

・上記の回転版

モジュールの中身を理解しよう

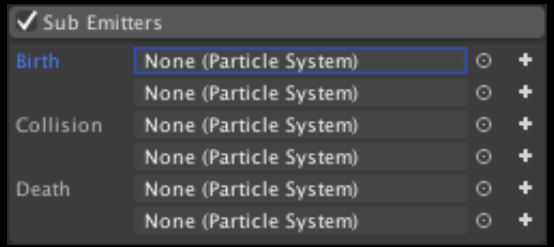
Collisionモジュール

- パーティクルと他のオブジェクトとの衝突を管理する。
- オプションのPlaneは負荷が軽くて対象物（空オブジェクトで十分）のローカルY軸に対してのみ衝突が発生する。それに対してWorldは一般的なコライダとの衝突を行う。
- Unity5.3から2D系のコライダに対してもヒットするようになった。
- Visible Bounceフラグはパーティクル自体のコライダをギズモによって視覚化したものだが、精度は割といい加減。
  ➡これにチェックを入れると、コリジョンモジュール自体を切ってもギズモが表示されるので注意しよう。
- スクリプトからコリジョンを検出したい場合、Send Collision Messagesフラグにチェックを入れるとMonoBehaviour.OnParticleCollision()で拾える。
  ➡衝突する側とされる側、どちらに書いても行える。
- スクリプトと後述するSubEmittersを併用すれば、雨のパーティクルが水面に当たると波紋になり、それ以外のオブジェクトに当たると消えるなどの処理も可能。

モジュールの中身を理解しよう

Sub Emitterモジュール

- パーティクル個々のポジションから派生したパーティクルを発生させる場合、どのタイミングで開始するのかを決定するためのモジュール。
- Birth（誕生時）、Collision（衝突時）、Death（消失時）の3つに登録できる。
- ＋をクリックすると自動でサブパーティクルが生成されるのからも分かる通り、子オブジェクトを登録するのがほぼ前提となっている。

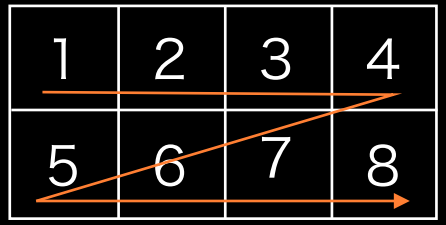


モジュールの中身を理解しよう

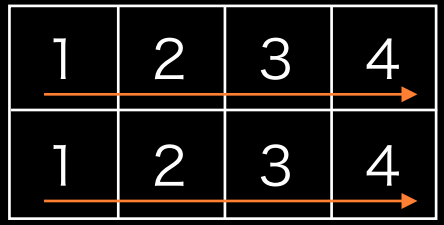
Texture Sheet Animation モジュール

・テクスチャ上でスプライトアニメーションを行うためのモジュール。
・マス目乗に区切ったセルに対してコマ送りアニメーションを行うイメージであり、縦横のマス目数は2のべき乗（2、4、8・・）にしなくてはならない。
・Animationオプションで「Whole Sheet」を選択すると全てのコマを使ったアニメーションを行い、「SingleRow」を選択すると列ごとのみのシーケンスで行われる（どの列を行うかはランダムにできる）。

・Whole Sheet



・Whole Sheet



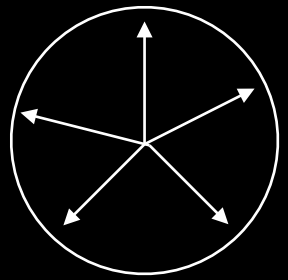
・このランダム性を利用して、右図のように異なった形状や色のパーティクルを単一のオブジェクトからエミットさせることもできる。

モジュールの中身を理解しよう

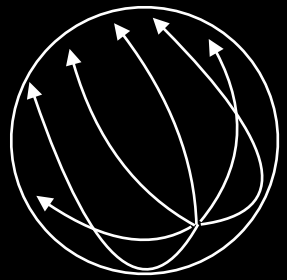
Renderer モジュール

- スクリーンに対してパーティクルをどう描画するかを決める、重要な役割を担う。
- Render ModeをStretched BillboardにしてSpeed Scaleを正に設定すると、速度によって進行方向にパーティクルが伸びるため、ダイナミックな表現が行える。また、Meshを選択すれば3Dオブジェクトを飛ばすこともできる。もちろんその場合はパーティクル用シェーダ以外を使うべし。
- Cast Shadowsにチェックを入れてもプリセットのパーティクル用シェーダは影を落とすことができない。使うにはShadowCasterに対応したシェーダを使う必要がある。
- 5.2で追加されたPivotはかなり面白い機能。スケールに比例して、パーティクルの拡散先がPivotで指定したベクトルの方向に変化する。

- Pivot off



- Pivot on

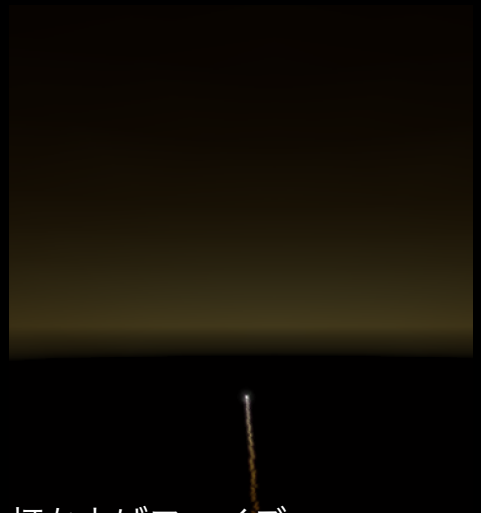


unity道場

打ち上げ花火を作ろう

ワークショップ：打ち上げ花火を作ってみよう

打ち上げ花火を3つの行程に分けて考えてみよう。

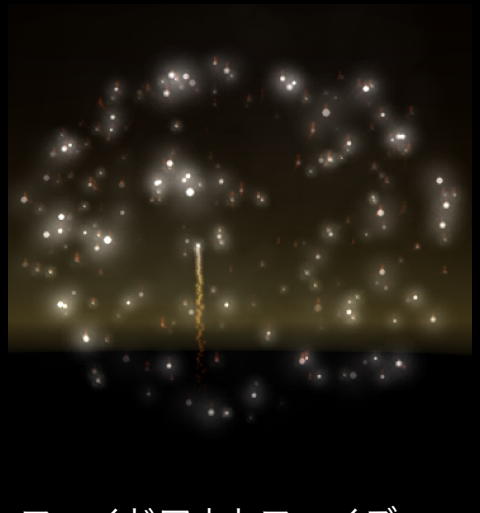


打ち上げフェイズ

・花火玉（本体）
・追跡する炎

爆発フェイズ

・拡散する火花
・炎を追跡する線
・煙



フェイドアウトフェイズ

・煌めく光

ワークショップ：打ち上げ花火を作ってみよう

まずは舞台設定

- 新規のシーンに置かれたDirectional LightをLightingパネル上のSunに設定し、適当に傾けてスカイボックスを夜にする。
- Main Cameraのポジションを（0, 25, -30)、ローテーションを(0, 0, 0)に設定。

ワークショップ：打ち上げ花火を作ってみよう

打ち上げの準備

- ParticleSystemをクリエイトし、ポジションを(0, 0, 0)、ローテーションを(-90, 0, 0)に設定。
- 上方向に白い玉がエミットされるのを確認してパーティクルインスペクタを下記のように変更。
  - Mainパネル / Start Lifetime: 3、Start Speed: 20、Gravity: 0.5
  - Emissionパネル / Rate: 0.5
  - Shapeパネル / Angle: 5
  - RendererパネルMaterialをParticle Fireworkに変更

これで2秒に1回、花火が打ち上げられるようになりました。
とりあえず名前をFireworksとしておこう。

ワークショップ：打ち上げ花火を作ってみよう

玉を追跡する炎を作る

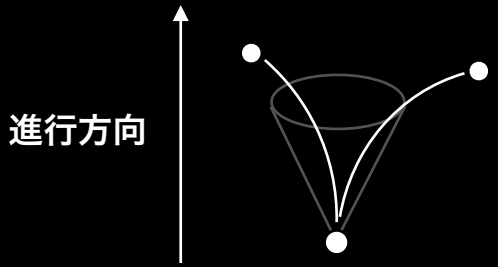
- 前項の玉のSub EmittersのBirthの「＋」からSubEmitterを生成。これは実質的にクリエイトして子オブジェクトに設定するのと同じ操作。これを編集して追跡する炎を作ろう。
- 炎は玉を追っかけるものなので、位置を玉自体に追従させ、かつインスタンス化のタイミングを打ち上げ時に設定する必要がある。
  ➡FireworksのSub EmittersパネルのBirthにこのオブジェクトをアサインすることで解決。
- また、エミット（放出）方法を時間単位でなく、移動量単位に設定するためにEmissionパネルのプルダウンをDistanceに変更し、Rateを30に変更。これにより、Rateが1秒単位でなく1unit（≒メートル）ごとの放出量に変化する。
- それからRendererモジュールのMaterialをParticle Fireworkに変更。

これでサブパーティクルは無事玉を追従するようになった。
次のステップで、この玉の塊を追跡する炎に近づけていく。

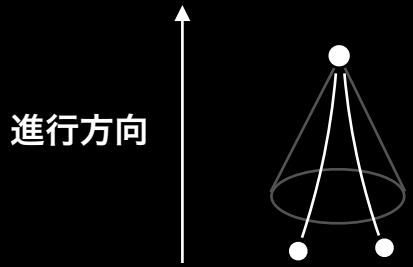
ワークショップ：打ち上げ花火を作ってみよう

追跡する炎のカスタマイズ

- 玉の後方から炎が噴射していくイメージなので、ShapeをConeに変更。
- また、初期の寿命、スピード、サイズを変更する必要があるのでMainパネルのStart Lifetime、Start Speed、Start Sizeを調整。ここではそれぞれ1、-1、0.4に設定。
- スピードをマイナスにしたのは、デフォルトでは進行方向（上方向）にコーンが広がっていくため。実際は後方に炎を広げる必要があるので、逆向きのマイナスを指定。また、拡散するのではなく収束するようなパーティクルの実装もスピードにマイナスを代入することで対応可能。



☒ Start Speedがプラスの場合

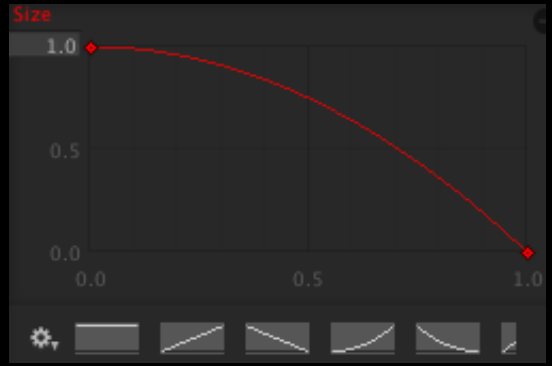


☑ Start Speedがマイナスの場合

ワークショップ：打ち上げ花火を作ってみよう

追跡する炎のカスタマイズ

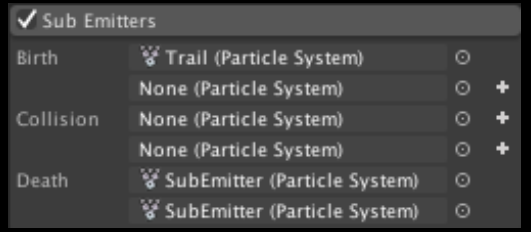
- 次いで色を変更を行う。Color Over Lifetimeから白→黄→オレンジと変化するように設定。
- また、Size Over Lifetimeでは最終的にスケールがゼロになるように設定。イージングはお好みだけど、なだらかなカーブが良さげ。
- これでトレイルの演出は完了。



ワークショップ：打ち上げ花火を作ってみよう

爆発を起こそう

- 爆発フェイズに必要なのは拡散する火花と煙の2つ。
- まずは追跡する炎と同様、SubEmittersのDeathに2つオブジェクトを追加

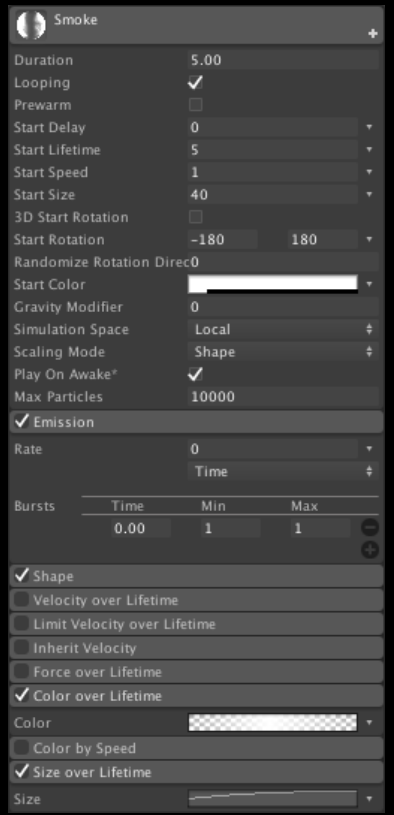


- ExplosionにリネームしてStart Lifetimeを右のように0.8-1.0に、Start Speedを40-60に、Gravity Modifierを3に設定。また、EmissionのMinとMaxを300にてエミット量を増やしておく。これで大きく弾け飛ぶような動きに。
- 火花は初速が早く、空気抵抗で急激に減速するのでLimit Velocity Over Lifetimeを有効にしてSpeedのカーブを減速するものに変更し、Dampenを0.1に設定。これで爆発直後の速度が緩慢に。

ワークショップ：打ち上げ花火を作ってみよう

煙を形作る

- 爆発はまだ未完成だが、続いて煙の作成を行う。
- 先ほど作ったサブパーティクルをSmokeとリネームし、Rendererモジュールのマテリアルを Particle Smoke Whiteに変更しておく。
- 調整のため、MainモジュールのStart Sizeを40に、均一化を避けるためにStart Rotationで -180と180の間でZ軸をランダムに回転させておく。Start Colorもアルファを30程度に下げておこう
- また、煙はパーティクル1つで表現するため、EmissionモジュールでButsts数を1にしておく。
- 徐々にフェイドイン・フェイドアウトするためにColor Over Lifetimeのアルファを0 → 255 → 0と変化するよう設定。
- 大きさも右図のようにSize Over Lifetimeで50パーセントからスタートさせてみるなどの工夫をするといい感じに



ワークショップ：打ち上げ花火を作ってみよう

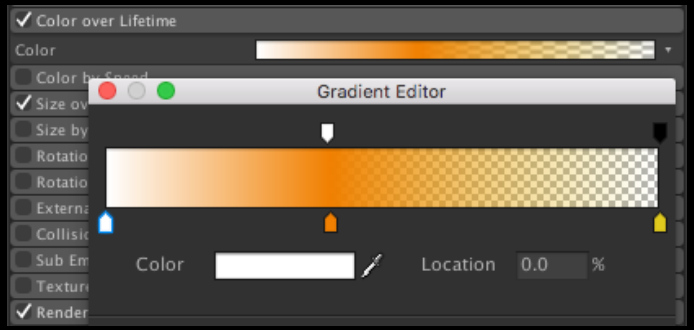
炎を追跡する線を作り、爆発を華やかに

- 続いては爆発を追跡する線を作る
- ExplosionのSub EmittersのBirthにサブパーティクルを追加し、Trailとリネームしておこう
- 寿命に幅をもたせるためStart Livetimeを0.6-1.2に設定し、Start Sizeを0.5とする。また、少し明るすぎるのでStart Colorのアルファを100程度にしておく。Gravity Modifierも1にしておけば花火本体とは別にこちらも下降するようになる。
- 続いてはEmissionモジュールの設定だ。先に作った追跡する炎と同様、Emissionの設定をDistanceに設定し、移動ごとにパーティクルが生まれるようにしよう。Rateを6に設定し、MainモジュールのMax Particleを100000に設定しておく。
- Shapeモジュールは使わないので切っておこう。

ワークショップ：打ち上げ花火を作ってみよう

炎を追跡する線を作り、爆発を華やかに

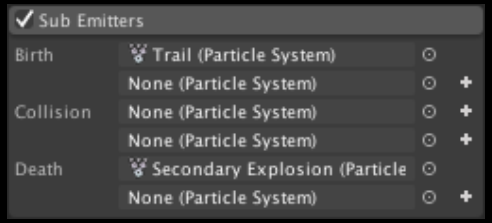
- 色をつけることによって華やかになる。
- Color Over Lifetimeのグラデーションを設定し、色を反映させよう。色は好みで決めれば良いが、白から任意の色に変化させ、またアルファを最終的に0にしたほうが見栄えが良いかもしれない。
- スケールもSize Over Lifetimeで変化させておこう。
- 爆発フェイズは以上。



ワークショップ：打ち上げ花火を作ってみよう

煌めく光を作る

- 最後に作る煌めく光は爆発後に起きるので、ExplosionのSub Emitters/DeathにSecondary Explosionと命名して追加しておく。

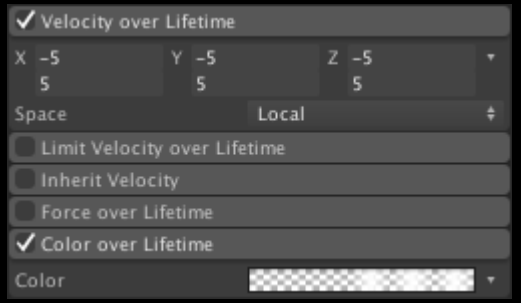


- Secondary ExplosionのStart Lifetimeを0.6-1に、StartSizeを1-3に設定することで消えるタイミングとサイズに幅をもたせておく。
- EmissionのBursts数を1に設定してエミットされる数を調整。
- RendererのMaterialをParticle Fireworkに変更。

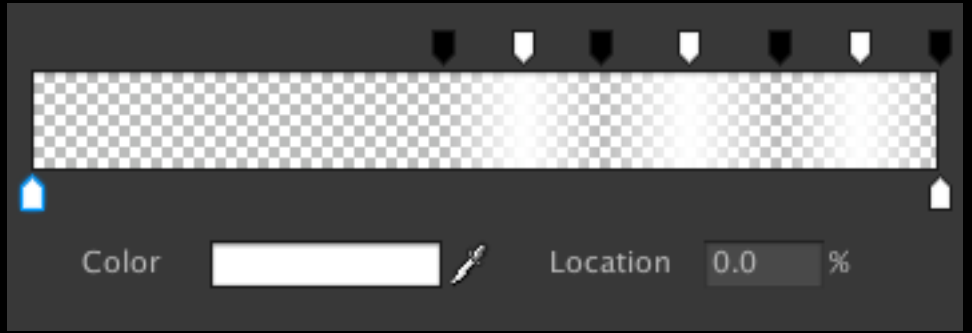
ワークショップ：打ち上げ花火を作ってみよう

煌めく光を作る

- 動きを少々まばらに、風に流される様を演出するためにVelocity Over Lifetimeを右図のように(-5, -5, -5)、(5, 5, 5)の間でランダムに動作するよう設定します



- 最後にColor Over Lifetimeを点滅するように設定すれば完了です。



unity 道場

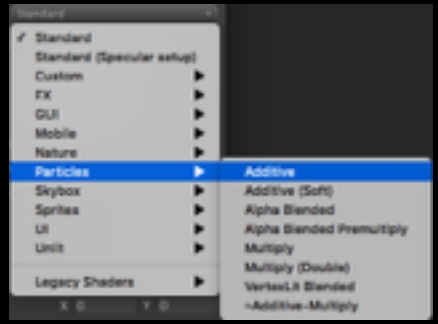
シェーダー、スクリプト
その他のテクニック

シェーダーについて

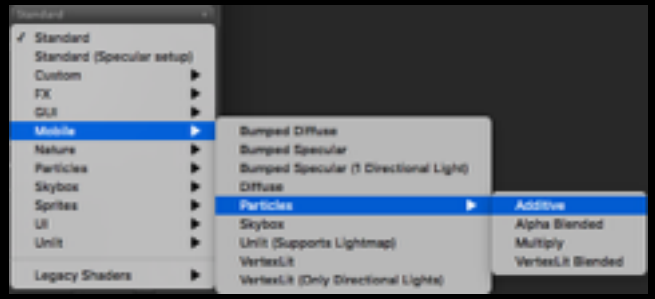
ビルトインシェーダ

- 基本的にパーティクルで使うべきシェーダは
  - /Particles/ディレクトリ以下
  - /Mobile/Particles/ ディレクトリ以下

- 基本的にモバイルプラットフォームではMobileを使うべき



- 大きく次のようなカテゴリーに分けられる
  - Additive
  - Alpha Blended
  - Multiply



シェーダーについて

Alpha Blended

- アルファ値が100%未満の場合、同色を重ねるごとに、その色自体に近づく。
- 紙吹雪や色ガラスなど、発光しないもの全般に使う

Additive(加算)

- 重なり合うごとに明るくなる。
- 基本的に炎やイナズマなど、光を伴う現象に用いる
  ➡元が明るいところでは使えない

Multiply(乗算)

- 絵の具のように混ざり合う色同士が暗くなる。
- 使い所は限られている（闇魔法とか？）
  ➡元が暗いところでは使えない

アルファ100%

Alpha Blended　Additive　Multiply

アルファ50%

Alpha Blended　Additive　Multiply

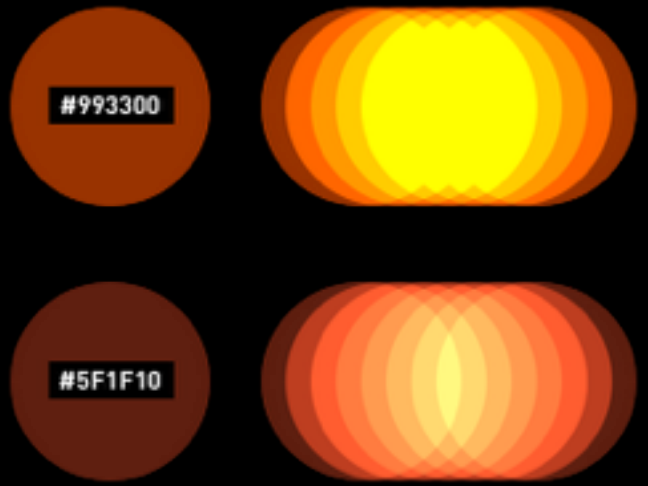
シェーダーについて

AdditiveとMultiplyの注意点

- Additiveは文字通りRGBごとの値を加算していくので、元がゼロではいくら塗りを重ねても白くなっていくことはないことに注意しよう。例えば右図の上部のサークルはR99、G33、B00なので、重ねるごとに#FFFF00に近づくだけ、つまり黄色を目指すだけだ。

- 対して下のサークルは明らかに上部のサークルよりも濃い色ではあるが、Bが10であるため重ねるごとに#FFFFFF、つまり白に近づいていく。多くのケースで必要とされているのは、このようなRGBいずれも0以上の数値であることを心に留めておこう。

- Multiply（乗算）シェーダを用いる場合は逆にFFを含む値があると、いくら重ねても黒には近づかない



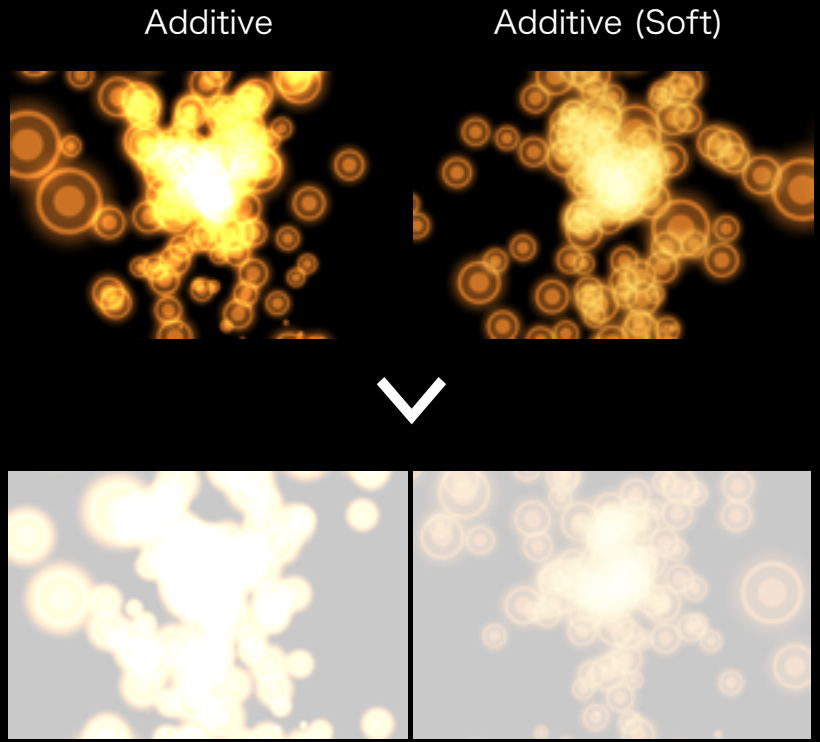
シェーダーについて

AdditiveとAdditive(Soft)

- Additive(Soft)はデスティネーションに補色を描くことにより混ざり具合が緩和されたもの。

```
Blend One OneMinusSrcColor
```

- Additiveはそもそも背景が白いと見えにくい、白飛びしてしまうという特徴がある。実際の炎や光線は明るい場所ではほぼ見えないが、ゲーム上の表現として、白い背景でも見せる必要がある場合がある。このような時にもSoftを試してみよう。

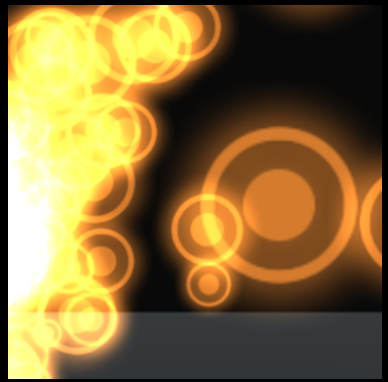


背景が明るいほど差が顕著に

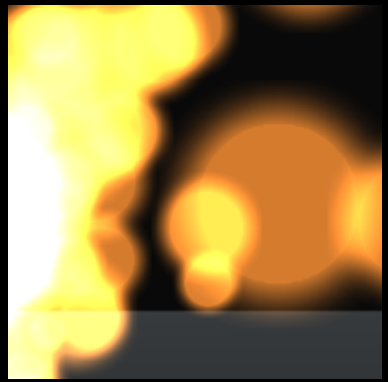
モバイル版のシェーダーと何が違うのか

機能面での違い

- 大きくはこの2点のみ
  - TintColorがある
  - Soft Particleが使える
- よってこの機能を使わなければ、コストの安いモバイル版シェーダで十分だったりする。
- TintColorは微細な色味調整に使うことが前提のようだが、αを200パーセントにブーストさせたりするのも面白い使い方ではないかと思う
- ちなみにTint Colorは基本に設定すべきデフォルト値が(0.5, 0.5, 0.5, 0.5)であることに注意しよう



アルファ100%

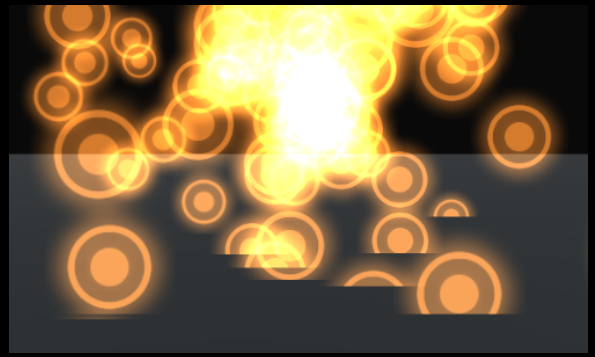


アルファ200%

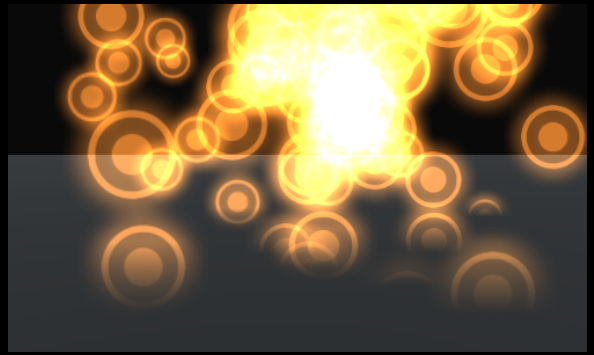
モバイル版のシェーダーと何が違うのか

Soft Particle

- もう1つの違いであるSoft Particleは遮蔽物などによりパーティクルが消える際、ボケ足をつけて綺麗にフェードアウトさせる機能だ。
- Deferred Renderingのみの機能なので、カメラあるいはPlayer Settingでレンダリングパスを Deferredに設定する必要がある。



Soft Particle OFF



Soft Particle ON

スクリプトを併用するケースの例

再生・一時停止・パラメータの変更

- Play On Awake フラグを立てない限り、パーティクルは再生しない。再生、一時停止、再開などのコントロールは基本的には外部のスクリプトから行うことになる。 Unity5.2まではスクリプトからアクセスできないパラメータが多数存在していたが、5.3ではほとんどの変数をコントロールできるようになった。

その他時間に対する諸々

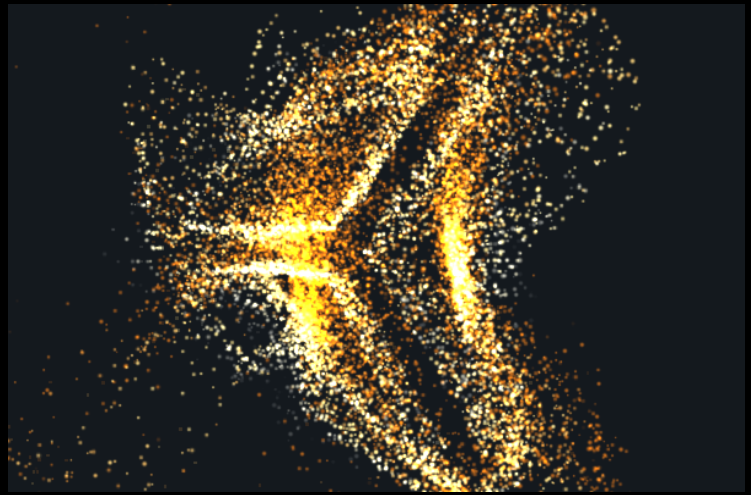
- 早送りやスロー再生したい場合はParticleSystem.Simulate()の引数にデルタタイムに変更を与えた値を取ることで実現する。
- また、ポーズ中などにタイムスケールを気にせずに動かしたい場合も同様。右はタイムスケールを無視する例。

```
float lastTime;

void Start () {
    lastTime = Time.realtimeSinceStartup;
}

void Update () {
    float deltaTime = Time.realtimeSinceStartup - lastTime;
    particle.Simulate(deltaTime, true, false);
    lastTime = Time.realtimeSinceStartup;
}
```

スクリプトを併用するケースの例

個々のパーティクルにアクセスする

- パーティクル個々に直接アクセスして座標、色、角度を変更することも可能であり、デモとしてPerlin Noiseによってデフォームするunity Logoを作ってみた。



> パーリンノイズ（英: Perlin noise）とは、コンピュータグラフィックスのリアリティを増すために使われるテクスチャ作成技法。擬似乱数的な見た目であるが、同時に細部のスケール感が一定である。このため制御が容易であり、各種スケールのパーリンノイズを数式に入力することで多彩なテクスチャを表現できる。パーリンノイズによるテクスチャは、CGIで自然な外観を物に与えるためによく使われる
>
> Wikipediaより

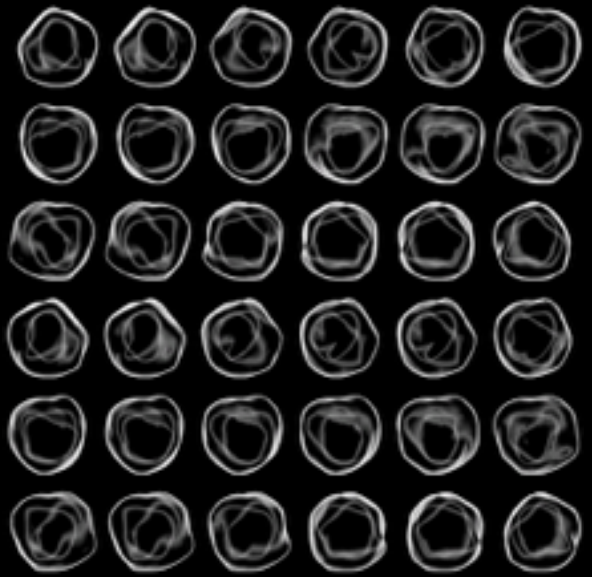
> 池和田の解釈：
> なんか突っ込んどけば自然な感じの動きができるやつ

- ちなみにノイズ関数のオプションは近々実装される予定

負荷を軽くするために

メッシュは極力エミットさせない

- 3Dオブジェクトを飛ばす場合、キャプチャした画像を2Dスプライトアニメーションにするなどし、なるべくビルボードを使おう



負荷を軽くするために

テクスチャをうまく使う

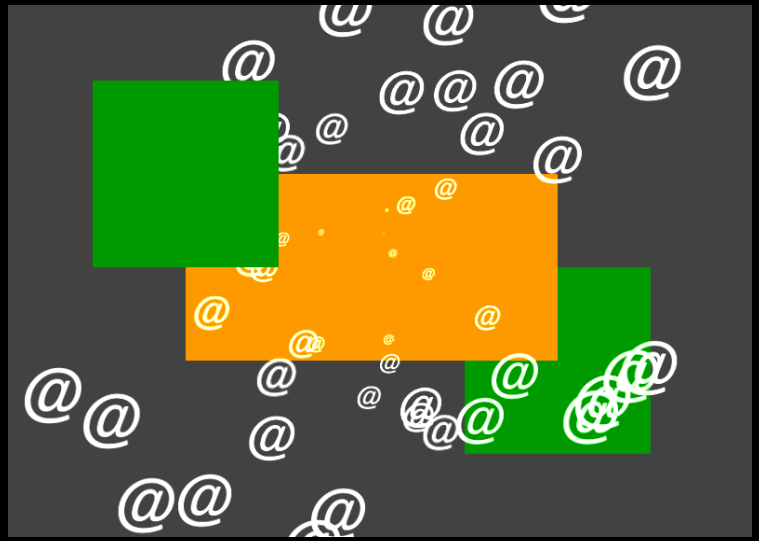
・ブラックホールの例では、パーティクル個々をエミットするのではなく、一枚のテクスチャ内に多数の粒子（この場合は星々）を書き込み、膨大なパーティクルに見せている。これも有効な負荷軽減法だ。

UIを扱う際の注意点

UIのエフェクトとしてパーティクルを使うには

- パーティクルをUIの前面に出す場合はまずCanvasのRender ModeをScreen Space Cameraにしよう。
  ➡もちろん3DUIの場合はWorld Spaceでもいい

- 描画旬はSorting Layer > Order In Rayerで決まる。常にUIのキャンバスより前、あるいは後ろに出したいのであればSorting Layerでコントロールしよう。
- しかし、同一キャンバス上の2つのUIオブジェクトに挟まれる場合などはUIオブジェクト個々にCanvasコンポーネントをアサインし、Order In Rayerのオーダー番号でコントロールすべきだろう。
- いずれにせよスクリプトを書く必要は無くなった :)



以上です