Particle Playground (v1.20) - Manual

Introduction

Particle Playground is a toolset which enables you to be creative in new ways of altering and rendering particles in your 3d application.

Particle Playground can process and create particles from images, meshes (static, moving, skinned/animated, procedural), painted and projected positions.

On top of this there are manipulators which will alter the particle behavior over their lifetime, adding different types of forces and properties. All of this is optimized for performance, which makes Particle Playground a perfect match for low-end/mobile devices (note that the example scenes is intended for desktop environments).

The Particle Playground framework features can all be called through scripting where both C# and JavaScript is available. It is also fully accessible through the Unity Editor's Inspector, where you can work with the settings through sliders and many times directly into the scene view through the interactive tools.

For more information please visit http://playground.polyfied.com/.

Getting Started

Once Particle Playground is imported to your Unity Project you can create it through the home screen ”*Playground Wizard*”. You’ll find it through **Window > Particle Playground > Playground Wizard**. From here you can create new Particle Playground Systems and presets.

To create a Particle Playground System through script please see the Script Reference section below.

Good to know

- **The Playground Wizard**
  This is where the action begins (Window > Particle Playground > Playground Wizard). Select your preferred script language in the menu. Use this to create new Particle Playground Systems and presets. All presets are physically stored in *Particle Playground/Resources/Presets/*, all icons for the presets are stored in *Particle Playground/Graphics/Editor/Icons/*.

- **The Playground Manager**
  When creating a new Particle Playground System a Playground Manager will automatically be instantiated. The Playground Manager is the object driving all Particle Playground Systems within the scene and is containing all *Global* Manipulators as well.

- **Example Scenes in the project**
  Please see the examples in *Particle Playground/Examples/Example Scenes/* within your *Assets* folder to get a preview of some of Particle Playground’s capabilities. Note that these are intended to run in desktop environments.

- **Local and Global Manipulators**
  A manipulator is an object that will alter velocities and other properties on a particle. The Local Manipulators are held by each Particle Playground System and will only affect the particles within itself. These are marked with cyan colors in the Scene View.
  All Global Manipulators are held by the Playground Manager. These will affect all Particle Playground Systems (within layer). These are marked with blue colors in the Scene View.
  To add one, simply select the Playground Manager (or the end of a Particle Playground’s Inspector) / Particle Playground System, unfold the Manipulator-section, click **Create** and assign a Transform from your scene. Make sure to set type, the layers it will affect, its size and strength to start manipulate the Particle Playground Systems within the scene.

  Always use Local Manipulators when you want the manipulator object to be stored within a prefab/preset. A Global Manipulator can only serialize within the scene along with the Playground Manager.

- **Publishing your preset**
  Selling your own created particle system presets on the Asset Store is encouraged. Make sure to read the Publishing Presets section found on the next page before you submit, for additional information please see playground.polyfied.com.

- **Working with C# and/or JavaScript**
  You choose your preferred language from the Window menu when opening the Playground Wizard. All C# classes ends with a ”C”. For instance calling a function on the Playground Wrapper in C# could look like this: `PlaygroundC.SomeFunction()`. You can run both languages at the same time in your scene if you for instance more easily want to create and publish presets which supports both languages. The languages are living in separate spaces, so you won’t be able to run a particle system made in JS from a C# Playground Manager.

If you don't plan on scripting towards Particle Playground the choice of language doesn't matter. Removing a language is as simple as removing the language specific folders ("JavaScript" and "Csharp") from your project folder.

- **Source Mode - Script**
  You can use a Particle Playground System to script your own source data for each particle. This is good when you want to extend the capabilities of a source but keep the core features such as the particle pool, forces, collisions and manipulators. Please see the Script Reference section for more details.

Publishing Presets

The Particle Playground framework wants to encourage creativity and the possibility to share your solutions amongst the community. Being a Particle Playground owner, you're entitled to sell your own created particle system presets and extensive solutions on the Unity Asset Store.
Creating a preset prepared for publishing is done through the Preset Wizard (found in Playground Wizard > Presets > Create).

To successfully publish your presets, please use the online Publish Guide.

Please note:

- Your customers do not need to own Particle Playground in order to run your preset, but will in that case lack the ability to intuitively edit its settings or to use the Playground Wizard to instantiate your preset. Therefore if you have any functionality to your preset which revolves around being customizable, please make sure that your scripts controlling the preset opens up for customizing. As an example see the Playground Laser preset's prefab.
- Publishing a preset with any of the Particle Playground framework Editor scripts is not allowed. Doing so is a subject for being unapproved during review. The only scripts allowed from the core framework is **Playground.js / PlaygroundC.cs** and **PlaygroundParticles.js / PlaygroundParticlesC.cs**.

Reference Manual

Particle Playground introduces some new possibilities along with some new naming conventions for Particle Systems. Here is a list of descriptions of the current available features, from Playground Editor top, along with script naming convention within parenthesis.

Particle Playground System `(PlaygroundParticles / PlaygroundParticlesC)`

Source

Source `(source)`
The source from which particles will emit in this Particle Playground System. You have several alternatives,

> **State** `(SOURCE.State / SOURCEC.State)`
> Emit from pre-defined positions created from an image or mesh. All Particle Playground Systems uses a list of states to store data from a mesh vertices or an image's pixels, this can be accessed by script through `PlaygroundParticles.states[int]` and will return a `ParticleState / ParticleStateC`.

When creating a new State from an image you can define Texture, Depthmap (and Depthmap Strength), Name, Scale, Offset and a parent Transform. When creating a State from a mesh you can define Mesh, Texture, Name, Scale, Offset and Transform. Using a transform will make you able to position, rotate and scale the State.

Texture `(stateTexture)`
Takes a Texture2D which will structure the State in color and positions from the Texture2D's pixels. If you use a texture when creating the State from a mesh, the positions will be colored from the UV-mapping of the mesh.

Depthmap `(stateDepthmap)`
Takes a Texture2D which will define the Z-value in Units by grayscale. Black = 0.0, White = 1.0.

Depthmap Strength `(stateDepthmapStrength)`
The amount the Depthmap will affect the Z-positioning. A Depthmap Strength of 1.0 will affect the Z-positions one Unit ranging from black to white.

Mesh `(stateMesh)`
The mesh to construct your State from. Each vertex in the mesh will define a position in the State.

Name `(stateName)`
The name of this State. This is only used for your own convenience where seeing the name in the States list is practical.

Scale `(stateScale)`
The world scale of this State. Each pixel will originally be a square of 1x1 Units. To make the final size of a State smaller, use a number below 1.0. Using a number below 0 will invert the State in X- and Y positions.

Offset `(stateOffset)`
A State will be created in Vector3(0, 0, 0) in world coordinates (or local coordinates if a stateTransform is set) with origin of the image's bottom left or a mesh's pivot. Use the offset to place it elsewhere, with offset from world's or stateTransform's Vector3(0, 0, 0) measured in Units.

Transform `(stateTransform)`
To be able to position, rotate and scale a State

When using State as Source you also have the possibility to have several states and do transitions between them. Active State decides which State currently should be constructed and when you change Active State a transition will occur if Transition isn't set to None.

Active State `(activeState)`
Decides which State should be active in a Particle Playground System's states-list.

Transition `(transition)`
The transition to use when Active State changes.

None `(TRANSITION.None / TRANSITIONC.None)`
No transition will occur when changing Active State.

Lerp `(TRANSITION.Lerp / TRANSITIONC.Lerp)`
A linear interpolation of position and color will occur when changing Active State.

Fade `(TRANSITION.Fade / TRANSITIONC.Fade)`
A fade-out of previous Active State and fade-in of current Active State will occur when changing Active State.

Fade2 `(TRANSITION.Fade2 / TRANSITIONC.Fade2)`
A fade-out of previous Active State and fade-in of current Active State will occur along with a linear interpolation of position when changing Active State.

Transition Time `(transitionTime)`
The time in seconds a transition will take.

Transform (SOURCE.Transform / SOURCEC.Transform)
Emit from a Transform component within your scene.

World Object (SOURCE.WorldObject / SOURCEC.WorldObject)
Emit from a Mesh component within your scene.

> **Update Mesh Normals** (worldObjectUpdateNormals)
> Enable this if the World Object's mesh is procedural and changes vertices and/or normals direction over time.

Skinned World Object (SOURCE.SkinnedWorldObject / SOURCEC.SkinnedWorldObject)
Emit from a Skinned Mesh component within your scene.

> **Source Down Resolution** (sourceDownResolution)
> The source vertex skipping. Use this to lower the distribution of particles needed along your complete skinned mesh. This is in many cases needed to amp performance when using skinned meshes in your scene, especially when you target mobile platforms.

Script (SOURCE.Script / SOURCEC.Script)
Control all particle behaviors through custom scripts (advanced). Use PlaygroundParticles.Emit(position, velocity, color, parent) to create a particle at position.

> **Emission Index** (scriptedEmissionIndex)
> The particle that will emit next time you call Emit() on this PlaygroundParticles object.
>
> **Position** (scriptedEmissionPosition)
> The particle's initial emission position when you call Emit() on this PlaygroundParticles object.
>
> **Velocity** (scriptedEmissionVelocity)
> The velocity of the particle that will emit next time you call Emit() on this PlaygroundParticles object.
>
> **Color** (scriptedEmissionColor)
> The color of the particle that will emit next time you call Emit() on this PlaygroundParticles object.
>
> **Parent** (scriptedEmissionParent)
> The parent transform of the particle that will emit next time you call Emit() on this PlaygroundParticles object. This is used when calculating transform-specific features such as "Only Source Positions" which will make a particle child itself to a determined transform's transform point.

Paint (SOURCE.Paint / SOURCEC.Paint)
Paint positions and color by using your own brushes, or by a single point with color information. Creating your own brushes can be done through the Brush Wizard, which you'll find in the Brush Preset menu when selecting Brush as Paint Mode. All painting features relies on colliders being hit in the scene.

> **Paint Mode**
> This is an Editor-specific feature which lets you switch between painting with a single point (*Dot*), using predefined brushes (*Brush*) or erasing paint positions (*Eraser*). To paint live into the scene in script please see the example scene *Particle Playground - Paint*.
>
> **Brushes**
> Painting with brushes lets you fill a larger area with a predefined texture rather than a single point (like Paint Mode: Dot). You can define the texture yourself along with how the brush should function, such as detail, scale and spacing. When painting, each pixel (depending on chosen Detail level) will represent a ray into the scene at screen position. Each ray need to hit a collider to create a Paint Position at world point.
>
> > **Brush Presets**

A list of predefined brushes with specific settings which will define current paint settings upon selection. You can create your own presets by pressing the ”Create”-button which will open the Brush Wizard. A preset is stored in *Particle Playground/Resources/Brushes* and can be edited through the Inspector. To remove a preset from the Brush Preset list either delete the prefab in the ”Brushes”-folder or change the presentation mode to ”List” instead of ”Icons” and press the button ”-”.

Brush Shape `(texture)`
The texture to project onto the surface where you choose to paint. The texture’s amount of pixels (width and height) will be calculated for the amount of Paint Positions that will be created (amount is shown in the ”Paint” progress bar). All brush textures need to have Read/Write Enabled and use True Color (non-compressed) in their Import Settings.

Brush Detail `(detail)`
The detail represents how many of the original pixels that should be read from the Brush Shape texture. This will affect how many positions that will be created on every brush stroke. Keep in mind that it’s by rare occasions you ever need a perfect copy of your brush’s full pixel amount onto a surface.

Perfect `(BRUSHDETAIL.Perfect / BRUSHDETAILC.Perfect)`
Every pixel will be read (100% of existing texture pixels).

High `(BRUSHDETAIL.High / BRUSHDETAILC.High)`
Every second pixel will be read (50% of existing texture pixels).

Medium `(BRUSHDETAIL.Medium / BRUSHDETAILC.Medium)`
Every forth pixel will be read (25% of existing texture pixels).

Low `(BRUSHDETAIL.Low / BRUSHDETAILC.Low)`
Every sixth pixel will be read (16.6% of existing texture pixels).

Brush Scale `(scale)`
The scale of a brush measured by the ratio of the original Brush Shape texture. For instance, a texture with 32x32 pixels and a scale of 0.5 will represent a screen space area of 16x16 pixels.

Brush Distance `(distance)`
How far the brush sees from its origin position into the scene (in the Editor the origin is the Scene View’s camera). If the distance is shorter than the target collider you want to paint on then no Paint Positions will be created.

Use Brush Color
Each Paint Position is created along with the Brush Shape’s color information at the ray’s pixel coordinate. Disabling this will use the Color chosen by you instead, the alpha information will still be used from the Brush Shape texture.

Color
The color to paint with when using the Dot Paint Mode. If you disable ”Use Brush Color” for a brush then Color will determine each Paint Position color instead.

Paint Mask `(layerMask)`
Determines which layer of colliders the paint function sees in the scene. Available for all Paint Modes (Dot, Brush and Eraser). Use this to see through certain objects or mask out a single object in the scene to avoid spilling paint.

Paint Spacing `(spacing)`
The space needed for next paint position to occur. This is measured from the last paint position’s world point towards where next will end up while painting. A transparent green disc will show the spacing area in Scene View while painting. Use this to distribute paint more evenly along a surface for instance.

Max Paint Positions `(paintMaxPositions – Playground.js/PlaygroundC.cs)`
The max positions allowed to be stored by this Paint object. Use this to limit the existing Paint Positions within a scene. ”Exceed Max Stops Paint” will determine if no more painting can occur or if Paint Positions will be removed (ascending from first position in list) when reaching the max limit.

Paint *-information*

The amount of Paint Positions currently in the scene. ”Max Paint Positions” will determine where 100% of the progress bar is. If you suddenly can’t see all Paint Positions the reason is much likely that you need to extend the amount of particles in ”Particle Settings”.

Start/Stop Paint
Use this to begin or abort painting into the scene.

Clear
Removes all Paint Positions from the scene.

Projection `(SOURCE.Projection / SOURCEC.Projection)`
Project particles from a transform using a texture. This behaves similarly to the Unity Projector. This can be used to for instance create fog, dust or splashes on certain surfaces. Choose to offset the origin texture and the projected source position from the surface using the projected normal’s direction. Use Live Update to update the projection position every frame. You can scale your projection in any direction with the projection transform, but do note that this will affect the projected position on all axes when using local simulation space.

Transform `(projectionTransform)`
The transform to project from.

Live Update `(liveUpdate)`
Determines if the projection should update every frame. Enable this if your projection transform moves or you have projected objects moving inside the projection area.

Origin Offset `(projectionOrigin)`
Offset the texture’s origin in X- and Y values.

Projection Distance `(projectionDistance)`
Determines how far the projection rays can travel into the scene, measured in Units.

Projection Scale `(projectionScale)`
The scale of projection in Units. A scale of one will make each pixel one Unit. When using local simulation space, use this to scale rather than the projection transform to ensure correct projection depth.

Surface Offset `(surfaceOffset)`
Determines how far away the source position will be distributed from surface. Using the projected surface normal.

Projection Mask `(projectionMask)`
The layer mask of which colliders within layer can be seen by the projection rays in the scene.

Particle Settings

Particle Count `(particleCount)`
The amount of particles that should be actively simulated in the system.

Emission Rate `(emissionRate)`
The percentage of flow (normalized from 0 - 1) in burst sequences. This is calculated with the Lifetime Sorting for a Particle Playground System to give a linear consistent behavior.

Overflow Offset `(overflowOffset)`
The offset each overflow iteration from the source’s number of points. This will make the source copy itself in direction. For instance, if you have a sphere that consists of 525 vertices and using Particle Count of 1050, the Overflow Offset will determine where the later 525 overflowing particles will position in relation to the first. The result is that you will get a perfect copy of your sphere at the offset position. This can for instance be used intuitively to recreate the lights on a runway,

especially together with a Lifetime Sorting of *Linear* or *Reversed*. Using a transform will make you able to rotate and scale the Overflow Offset.

Source Scatter `(sourceScatterMin, sourceScatterMax)`
The spread of source positions within minimum- to maximum Vector3 range. Use this to scramble your source positions to make them appear more randomly distributed.

Overflow Mode `(overflowMode)`
Use this to set which method to calculate Overflow Offset by. Available methods are Source Transform (transform point), Particle System Transform (transform point) and World (global).

> **Source Transform** `(OVERFLOWMODE.SourceTransform / OVERFLOWMODEC.SourceTransform)`
> Offset by calculating the source's transform point.
>
> **Particle System Transform** `(OVERFLOWMODE.ParticleSystemTransform / OVERFLOWMODEC.ParticleSystemTransform)`
> Offset by calculating the particle system's transform point.
>
> **World** `(OVERFLOWMODE.World / OVERFLOWMODEC.World)`
> Offset by calculating the world position.
>
> **Source Point** `(OVERFLOWMODE.SourcePoint / OVERFLOWMODEC.SourcePoint)`
> Offset by calculating the source position using the source point's normal direction.

Emit Particles `(emit)`
Determines whether this Particle Playground System should emit particles or not. A Particle Playground System reuses each particle in a cached pool, turning Emit Particles off will make the calculation loop halt upcoming rebirths.

Loop `(loop)`
Determines if this particle system should loop its lifetime cycle or just run the first.

Disable On Done `(disableOnDone)`
When loop is set to false, this determines if the GameObject should disable when the lifetime cycle has run through to the end of last particle.

Calculate Particles `(calculate)`
Determines whether this Particle Playground System should calculate its particles or not. Particle Playground will run all simulation over time in its calculation loop, such as velocities, lifetime coloring and manipulators that should affect this system.

Minimum Size `(sizeMin)`
The minimum size of a particle.

Maximum Size `(sizeMax)`
The maximum size of a particle.

Scale `(scale)`
The scale of minimum- and maximum size.

Initial Rotation Speed `(initialRotationMin, initialRotationMax)`
The minimum- and maximum initial rotation of a spawned particle.

Rotation `(rotationSpeedMin, rotationSpeedMax)`

The minimum- and maximum rotation speed of each particle. To extend above 360 degrees please change `PlaygroundParticleSystemInspector.maximumAllowedRotation`.

Rotate Towards Direction `(rotateTowardsDirection)`
Apply rotation based on each particle's velocity. This creates a direction which the particle will turn towards. To offset the rotation use Initial Rotation Speed.

Rotation Normal `(rotationNormal)`
When using 'Rotate Towards Direction', the rotation normal determines which vector to rotate around. This is always a normalized value set in world coordinates. A common operation would be to rotate around the Main Camera's (negative or positive) transform's forward to make the rotation always appear the same for the user.
For instance, `particles.rotationNormal = Camera.main.transform.forward`.

Lifetime `(lifetime)`
The particles lifetime in seconds.

Lifetime Size `(lifetimeSize)`
The particles size over lifetime. This is determined by a AnimationCurve where x-axis 0.0 to x-axis 1.0 is the complete lifetime and y-axis is the size in Units.

Lifetime Sorting `(sorting)`
The sorting of how the lifetime initially should be structured in this Particle Playground System. Use this to create different patterns in appearance of their source position.

> **Scrambled** `(SORTING.Scrambled / SORTINGC.Scrambled)`
> Particles will be randomly distributed.
>
> **ScrambledLinear** `(SORTING.ScrambledLinear / SORTINGC.ScrambledLinear)`
> Particles will be randomly distributed but ensured to never appear at the same time.
>
> **Burst** `(SORTING.Burst / SORTINGC.Burst)`
> Particles will be created all at once.
>
> **Linear** `(SORTING.Linear / SORTINGC.Linear)`
> Particles will be distributed linearly over their lifetime with sorting from source positions.
>
> **Reversed** `(SORTING.Reversed / SORTINGC.Reversed)`
> Particles will be distributed linearly reversed with sorting from source positions.
>
> **Nearest Neighbor** `(SORTING.NearestNeighbor / SORTINGC.NearestNeighbor)`
> Particles will be distributed by distance to nearestNeighborOrigin. This will create a water ripple effect in their lifetime appearance from origin and out.
>
> **Nearest Neighbor Reversed** `(SORTING.NearestNeighborReversed / SORTINGC.NearestNeighborReversed)`
> Particles will be distributed by distance from nearestNeighborOrigin. This will create an inverse water ripple effect in their lifetime appearance from max distance towards origin.
>
> **Custom**`(SORTING.Custom / SORTINGC.Custom)`
> Particles will be distributed with a curve where X is total amount of particles and Y is total lifetime.
>
> Examples when using two positions on the curve:
> X1Y1, X0Y0: Linear

X0Y0, X1Y1: Reversed
X1Y1, X1Y1: Burst

Lifetime Offset `(lifetimeOffset)`
Offsets the lifetime cycle. This can be used to set particle systems in sequences to each other. For instance, using the Playground Runway preset you can now determine if two (or more) runways should be similarly synced or offset in their blinking lights. You can also use this to annihilate any fade-ins at first particle cycle by setting negative values. This can for instance be useful in a situation where you want clouds similar to the Cloud preset to be fully visible from first frame in the first particle cycle.

Forces

Only Source Positions `(onlySourcePositioning)`
Overrides all velocities and set every particle towards their source position every Update-cycle. This can be a desired behavior when particles doesn't move by force, but by their attached source transform. You can still set all other type of lifetime behaviors and offsets. Please see the example preset "Matrix Cube" or "Holobot" for basic usage.

Calculate Delta Movement `(calculateDeltaMovement)`
A Playground Particle System that uses World Object, Skinned World Object or Transform as Source can calculate birth velocity from each points delta movement. What this practically does is to give the particle an extra knock in the direction of the vertex- or position movement. Use Delta Movement Strength to set the velocity scale.

Delta Movement Strength `(deltaMovementStrength)`
The strength of the calculated delta movement. Particles with high Delta Movement Strength will appear lighter than particles with low Delta Movement Strength.

Lifetime Velocity `(lifetimeVelocity)`
The particles velocity over time. This is represented by three AnimationCurves in X-, Y- and Z values. Use this to create controlled movement patterns for your particles such as waves or wind. Use `PlaygroundParticles.applyLifetimeVelocity = true;` to turn this behavior on.

Initial Velocity `(initialVelocityMin, initialVelocityMax)`
The initial velocity for each particle in world coordinates. Use this to create a constant initial force towards direction. These are set by minimum and maximum value to create a spread within range.

Initial Local Velocity `(initialLocalVelocityMin, initialLocalVelocityMax)`
The initial velocity for each particle in local coordinates. This can be used together with all types of sources that has a transform attached. Use this to emit in the direction of the normals of a mesh or in the local position with rotation of a transform. These are set by minimum and maximum value to create a spread within range.

Initial Velocity Shape `(initialVelocityShape)`
Shape your own initial velocity by Vector3AnimationCurves. The shape applies to an emitted particle's force where X is total amount of source positions and Y is multiplier for total initial velocity. Use this to create shapes in how your particles spread out in the scene.
For instance, try a transform with overflow offset to see the basics of how this distributes velocities to each particle's initial velocity at birth with respect to source positions.

Velocity Bending `(velocityBending)`
Possibility to bend a particle's velocity using the reflected value of current velocity multiplied with bending. The direction from each particle's source position towards their current is seen as the normal plane. Each particle's current velocity path is altered with the bended direction. You can use

this to create interesting movement patterns without having to use Manipulators (combine them to create really interesting behavior).

Gravity `(gravity)`
Creates a constant force towards this Vector3.

Damping `(damping)`
The inertia over time of each particle.

Max Velocity `(maxVelocity)`
The maximum velocity magnitude allowed for a particle.

Axis Constraints `(axisConstraints)`
The world axises (boolean values in X, Y and Z) to constraint forces for a particle.

Collision

Collision `(collision)`
Determines whether the particles should collide or not. This will calculate collisions with colliders within your scene.

Collision Mask `(collisionMask)`
A LayerMask which determines which objects these particles can collide with.

Collide With Rigidbodies `(affectRigidbodies)`
Determines whether each collision should affect rigidbodies and apply forces to them. This can only happen if a particle collides with a collider that also has a rigidbody as component.

Mass `(mass)`
The mass of each particle. This is used when calculating how much each particle will affect a rigidbody. A particle with a higher mass will affect the rigidbody more than a particle with a lower mass.

Collision Radius `(collisionRadius)`
The collision radius of each particle.

Lifetime Loss `(lifetimeLoss)`
The amount of lifetime (energy) to loose on collision measured by remaining lifetime span in a normalized value. A lifetime loss of 0.5 on a particle in 50% of its lifetime will set it to 75% of its lifetime.

Bounciness `(bounciness)`
The bounciness of each particle. This value will determine how much of the original force the particle will detain after collision. For instance, using a value of 0.5 will make the particle loose half its force, using a value of 1.0 will make the particle have all force in remain.

Random Bounce `(bounceRandomMin, bounceRandomMax)`
The random offset bounce determined within minimum and maximum Vector3-value from the collision surface's normal. Use this to simulate uneven surfaces.

Collision Planes `(colliders)`
The infinite collision planes. These are created from transforms within the scene using the transform's upward axis to determine if a particle is within or passed the infinite plane. A particle cannot live outside of the passed plane when using collisions. Use this to contain particles within a

determined space. The planes are updated within the calculation cycle to always match their assigned transform.

Rendering

Material (particleSystemRenderer.sharedMaterial)
The material each particle uses in this Particle Playground System.

Lifetime Color (lifetimeColor)
The color each particle uses in this Particle Playground System during their lifetime. This is determined by a Gradient. When using State as Source and an image, the alpha from Lifetime Color will be used. If you don't plan to use alpha over lifetime you can disable this behavior in the Playground Manager by setting Lifetime Alpha for States to false (Playground.statesUsesLifetimeAlpha).

Color Source (colorSource)
Choose which type of method to colorize your particles with Color Source (found in Rendering). Using a state with a texture or painted positions from a brush with a texture is an example of a Color Source. If no source is used a fallback to Lifetime Color will occur. You also have the possibility to only set alpha from Lifetime Color while the colors are picked up from the source positions.

Source Uses Lifetime Alpha (sourceUsesLifetimeAlpha)
Determines if the source color should use alpha from its source or from Lifetime Color.

Render Mode (particleSystemRenderer2.renderMode)
The graphical presentation of a particle. This is directly connected to the Shuriken particle system.

Max Particle Size (particleSystemRenderer2.maxParticleSize)
The normalized screen size of a particle. This is directly connected to the Shuriken particle system.

Advanced

Update Rate (updateRate)
The update rate of this Particle Playground System. This determines how often the calculation loop will run. For instance, 1 will make the calculation run each frame, 2 will make it run each second frame. The higher the number the more choppy the particles will move over time, but hog up less of the main thread. Use this to balance quality of appearance with performance.

Simulation Space (shurikenParticleSystem.simulationSpace)
Determines if particles are simulated in world- or local space. This is directly connected to the Shuriken particle system. However, it affects how particles are calculated where world to local space is converted throughout the whole framework.

Rebirth Options
Control if certain actions should be run upon particle rebirth.

> **Random Size** (applyRandomSizeOnRebirth)
> Particle will get a new size within minimum and maximum range of sizeMin and sizeMax.
>
> **Random Rotation** (applyRandomRotationOnRebirth)
> Particle will get a new initial rotation within minimum and maximum range of initialRotationMin and initialRotationMax.
>
> **Random Scatter** (applyRandomScatterOnRebirth)

Particle will get a new scatter position within minimum and maximum range of sourceScatterMin and sourceScatterMax.

Particle Pool
This is an Editor feature where you can *Clear* out all currently simulated particles (source positions will remain intact) and *Rebuild* them towards their source positions.

Playground Manager `(Playground / PlaygroundC)`

Particle Systems

Particle Systems `(particleSystems)`
The list of Particle Playground Systems within the scene. Use this list in the Editor to create new, jump between (by pressing the name), to copy, sort or remove the particle systems. To access a particle system through script you can use `Playground.GetParticles(int)`, where `int` will be the position in the list. This will return a `PlaygroundParticles / PlaygroundParticlesC` object.

Manipulators

Manipulators `(manipulators)`
The list of Manipulators used within the scene. A Manipulator has abilities to alter the Particle Playground Systems within the scene. Press ”Create” to create a new Manipulator, then Enable it and assign a Transform from your scene. Make sure to set the Type, what it Affects, its Size and Strength. Having the Manipulator unfolded in the list will render handles in the Scene View where you can change size and strength. To access a particular Manipulator through script you can use `Playground.GetManipulator(int)` which will return a `ManipulatorObject` in `int` position.

Enabled `(enabled)`
Determines if this Manipulator is active.

Transform `(transform)`
The Transform to assign to this Manipulator. Use an object from your Scene, it’s the Transform object that will be the source for this Manipulator.

Type `(type)`
The behavior of this manipulator.

None `(MANIPULATORTYPE.None / MANIPULATORTYPEC.None)`
The behavior will be inactive.

Attractor `(MANIPULATORTYPE.Attractor / MANIPULATORTYPEC.Attractor)`
The behavior will attract particles with funnel-like features.

AttractorGravitational `(MANIPULATORTYPE.AttractorGravitational / MANIPULATORTYPEC.AttractorGravitational)`
The behavior will attract particles with gravity-like features.

Repellent `(MANIPULATORTYPE.Repellent / MANIPULATORTYPEC.Repellent)`
The behavior will repel particles with magnetic repellent-like features.

Property `(MANIPULATORTYPE.Property / MANIPULATORTYPEC.Property)`
The behavior will alter the property of each particle within range.

None `(MANIPULATORPROPERTYTYPE.None / MANIPULATORPROPERTYTYPEC.None)`

Don't alter any properties. This will however flag each particle within range in the particle pool and set its value of `changedByProperty` to true. You can use this to give particles your own property logic.

Color `(MANIPULATORPROPERTYTYPE.Color / MANIPULATORPROPERTYTYPEC.Color)`
LifetimeColor `(MANIPULATORPROPERTYTYPE.LifetimeColor / MANIPULATORPROPERTYTYPEC.LifetimeColor)`
Alter the color of particle within range.

Only Color In Range `(onlyColorInRange)`
Determines if the new color will be kept by the particle or go back to its original when out of the manipulator's range.

Keep Color Alphas `(keepColorAlphas)`
Determines if the new color will inherit the original alpha from the particle's source color or get the alpha from the new color.

Velocity `(MANIPULATORPROPERTYTYPE.Velocity / MANIPULATORPROPERTYTYPEC.Velocity)`
Alter the velocity of particle within range.

Local Rotation `(useLocalRotation)`
Determines if the velocity should be calculated from the transform direction of the manipulator.

Additive Velocity `(MANIPULATORPROPERTYTYPE.AdditiveVelocity / MANIPULATORPROPERTYTYPEC.AdditiveVelocity)`
Add velocity to particle's current velocity.

Size `(MANIPULATORPROPERTYTYPE.Size / MANIPULATORPROPERTYTYPEC.Size)`
Alter the size of particle within range.

Target `(MANIPULATORPROPERTYTYPE.Target / MANIPULATORPROPERTYTYPEC.Target)`
Set node targets in form of transforms for particles.

Death `(MANIPULATORPROPERTYTYPE.Death / MANIPULATORPROPERTYTYPEC.Death)`
Force a sooner death upon particles.

Combined `(MANIPULATORTYPE.Combined / MANIPULATORTYPEC.Combined)`
Combine manipulator properties into one manipulator call using the same position and radius.

Use **Transition Lerp/Linear** to change a property over time. The manipulator's **Strength** will determine how fast the property changes.

Affects `(affects)`
The layers this Manipulator will affect.

Size `(size)`
The spherical size of this Manipulator in Units. For instance, a Manipulator at the world position Vector3(0, 0, 0) and a value of 1 will make the spherical extents reach between Vector3(-0.5, -0.5, -0.5) and Vector3(0.5, 0.5, 0.5). All particles outside of the spherical extent will be ignored. Use the Scene View along with the handles when a Manipulator is unfolded from the Playground Manager's list to see and edit how far it reaches.

Strength `(strength)`
The strength of this Manipulator. All particles within the spherical size of this Manipulator will be affected. The outcome of the behavior is dependent on the selected Type. Use the Scene View along with the handles when a Manipulator is unfolded from the Playground Manager's list to see and edit their strength.

Inverse Bounds (inverseBounds)
Invert the bounding space this Manipulator affects.

Advanced

Calculate Particles (calculate)
Turn this off to override calculation settings for all Particle Playground Systems and turn calculation off in the entire scene.

Garbage Collection (garbageCollectOnResize)
Determines if a GC.Collect should be run when resizing of all arrays occur when changing particle count.

Group Automatically (autoGroup)
Determines if a newly created or non-childed Particle Playground System should automatically get Playground Manager as parent. This behavior is only for user convenience.

Build Zero Alpha Pixels (buildZeroAlphaPixels)
Determines if an image's completely transparent pixels should be built as particles or not. Have this setting off if you want to spare the number of particles. Turn it on if you for instance want to linearly interpolate from one state that has transparent pixels to another which doesn't (or has transparent pixels on different locations as the previous state).

Scene Gizmos (drawGizmos)
Show Gizmos from Manipulators within Scene View.

Paint Toolbox (paintToolbox)
Show toolbox when painting in Scene View.

Pixel Filter Mode (pixelFilterMode)
The method to filter pixels with when reading textures.

Bilinear (PIXELMODE.Bilinear / PIXELMODEC.Bilinear)
Bilinear filtering mode.

Pixel32 (PIXELMODE.Pixel32 / PIXELMODEC.Pixel32)
Pixel32 filtering mode.

Time Simulation Reset
Set time simulation to current time.

Editor Limits
All limits constraining the Editor GUI controls in terms of minimum and maximum values.

Script reference

Particle Playground is written in C# and JavaScript. The language you prefer to work with can either be determined by the choice of Playground Wizard (language) or by script, where all C# classes ends with a ”C”. You can alter all settings on a Particle Playground System by calling variables and functions on a PlaygroundParticles / PlaygroundParticlesC object, a wrapper through the Playground / PlaygroundC object is also available for convenience. To create a new Particle Playground System through script you use **Playground.Particle()** (JavaScript) or **PlaygroundC.Particle()** (C#) which will return a PlaygroundParticles/C object. Please see the example scene *Particle Playground - Features* for more detailed information on how to create and alter a Particle Playground System through script. To learn the basics of painting during runtime please see the example scene *Particle Playground - Paint*.

Running in Source Mode - Script
Particle Playground can utilize a scripted version of a source by setting a particle system’s Source to Script. In this mode you will bypass certain automatic features and be expected to run your own methods of source positioning, color, velocity and parent (if using transform-specific features such as ”Only Source Positions”. Lifetime Sorting, Overflow Offset and Delta Movement will not be applicable in this mode. However, what this gives you is methods to run a highly controlled particle emission with calculated forces, collisions and manipulations by the framework.

Emission can be done by calling the Emit() function on a PlaygroundParticles object. Example:

```
playgroundParticleSystem.Emit(position, velocity, color, parent);
```

- Position is set in world coordinates by a Vector3.
- Velocity is set in units by a Vector3. This overrides the Initial Local Velocity.
- Color is set by Color. Use Rendering > Color Source > Source to enable this color.
- Parent is set by a Transform. This tells which object in the scene this particle belongs to when using transform-specific features such as ”Only Source Positions”. This can be used to attach particles to specified objects.

This can be used in many ways, for instance turning a Particle Playground System into a particle pool where you have great control over when and where a particle exists, at what speed and color. For instance - high speed projectiles with precise collisions, voxel-like objects/landscapes and any kind of predetermined data.

Playground.js / PlaygroundC.cs - Public Variables

These are the public variables exposed from the Playground Manager. Note that any C# specific classes and enums ends with a ”C”.

Variable Name	Type	Description
particleSystems	List.<PlaygroundParticles>	List of particle systems handled by Playground Manager.
manipulators	List.<ManipulatorObject>	List of manipulator objects handled by Playground Manager.
calculate	boolean	Calculate forces on PlaygroundParticles objects.
pixelFilterMode	PIXELMODE	Color filtering mode used when constructing states.
garbageCollectOnResize	boolean	Issue a GC.Collect when particle lists are resized.
autoGroup	boolean	Automatically parent a PlaygroundParticles object to Playground if it has no parent.
buildZeroAlphaPixels	boolean	Turn this on if you want to build particles from 0 alpha pixels into states.
drawGizmos	boolean	Draw gizmos for manipulators in Scene View.
paintToolbox	boolean	Show toolbox in Scene View when Source is set to Paint on PlaygroundParticles objects.

Playground.js / PlaygroundC.cs - Functions

These are the static wrapper- and shared functions exposed on a Playground Manager for altering a Particle Playground System. Please see Playground.js or PlaygroundC.cs for available overloads and passed parameters. Note that any C# specific classes and enums ends with a ”C”.

Function Name	Return Type	Description
Particle	PlaygroundParticles	Create a PlaygroundParticles object.
Emit	int	Spawns a particle when Source is set to Script.
Random	null	Random position all particles within a PlaygroundParticles object.
Lerp	null	Linear interpolation of position and color of a PlaygroundParticles object.
ColorLerp	null	Linear interpolation of color of a PlaygroundParticles object.
PositionLerp	null	Linear interpolation of position of a PlaygroundParticles object.

Function Name	Return Type	Description
SetPosition	null	Set new image/mesh/worldobject position instantly.
GetPosition	null	Get position from a world object in a PlaygroundParticles object.
GetNormals	null	Get normals from a world object in Vector3[] format.
SetColor	null	Set new color to passed color instantly.
SetAlpha	null	Set alpha of particles instantly.
SetSize	null	Set particle size.
Translate	null	Translate all particles in Particle System.
Update	null	Refresh and calculate particles in this Playground Particles object.
Add	null	Add State.
AddCollider	PlaygroundCollider	Add a plane collider to a Particle System.
SetParticleCount	null	Set amount of particles for this Particle System.
SetLifetime	null	Set lifetime for this Particle System.
SetMaterial	null	Set material for this Particle System.
Destroy	null	Destroy this Particle System.
WorldObject	WorldObject	Create a world object reference (used for live world positioning of particles towards a mesh).
SkinnedWorldObject	SkinnedWorldObject	Create a skinned world object reference (used for live world positioning of particles towards a mesh).
ManipulatorObject	ManipulatorObject	Create a ManipulatorObject.
GetManipulator	ManipulatorObject	Return a ManipulatorObject in array position of manipulators-list.
GetParticles	PlaygroundParticles	Return a PlaygroundParticles object in array position of particleSystems-list.
PaintObject	PaintObject	Create a new PaintObject.
Paint	null	Paint into a PaintObject/ PlaygroundParticles-object.
Erase	boolean	Erase paint in a PaintObject/ PlaygroundParticles-object. Returns true if a position were erased.
SetInitialTargetPosition	null	Set initial target position for this Particle System.
Emission	null	Set emission for this Particle System.
Clear	null	Clear out this Particle System.
InstantiatePreset	PlaygroundParticles	Instantiates a preset by name reference. The preset must be stored in ”Particle Playground/Resources/Presets/”
GetPixels	Color32[]	Return pixels from a texture.
Offset	Vector3	Return offset based on image size

Function Name	Return Type	Description
RandomVector3	Vector3[]	Return random vector3-array.
RandomFloat	float[]	Returns a float array by random values.
ShuffleFloat	null	Shuffle an existing float array (alters object by reference).
Largest	int	Compare and return largest array.
CountZeroAlphasInTexture	int	Count the completely transparent pixels in a Texture2D.
ResourceInstantiate	GameObject	Instantiate from Resources Folder.

Playground.js / PlaygroundC.cs - Classes

These are the classes available in Playground.js. Please see classes in Playground.js for function overloads and further details.

PaintObject / PaintObjectC

Contains and alters data for particle paint.

Name	Type	Description
Initialize	function	Initializes a PaintObject for painting.
Paint	function	Live paint into this PaintObject using a Ray and color information. See the PaintObject class for overloads.
Erase	function	Erase in this PaintObject using a position and radius, returns true if position was erased.
GetPosition	function	Return position at index of PaintObject's PaintPosition.
GetColor	function	Return color at index of PaintObject's PaintPosition.
GetNormal	function	Return normal at index of PaintObject's PaintPosition.
GetParent	function	Return parent at index of PaintObject's PaintPosition.
Update	function	Live positioning of paintPositions regarding their parent. Pass in an int to update a specific position.
RemoveNonParented	function	Clear out all emission positions where the parent transform has been removed.
ClearPaint	function	Clear out the painted positions.
spacing	float	The required space between the last and current paint position.
layerMask	LayerMask	The layers this PaintObject sees when painting
brush	Brush	The brush data for this PaintObject

Name	Type	Description
exceedMaxStopsPaint	boolean	Should painting stop when paintPositions is equal to maxPositions (if false paint positions will be removed from list when painting new ones)
initialized	boolean	Is this PaintObject initialized yet?

PlaygroundBrush / PlaygroundBrushC

Contains data for a brush used when painting.

Name	Type	Description
SetTexture	function	Set the texture of this brush
Construct	function	Cache the color information from this brush
GetColor	function	Return color at index of Brush
SetColor	function	Set color at index of Brush
texture	Texture2D	The texture to construct this Brush from
scale	float	The scale of this Brush (measured in Units)
detail	BRUSHDETAIL	The detail level of this brush
distance	float	The distance the brush reaches

ParticleState / ParticleStateC

Contains and alters data for a single state.

Name	Type	Description
Initialize	function	Initializes a ParticleState for construction.
ConstructParticles	function	Construct data for particle position and color.
SetDepthmap	function	Set depth map to image ParticleState.
GetColor	function	Return color at index of ParticleState.
GetColors	function	Return colors in ParticleState.
GetPosition	function	Return position at index of ParticleState.
GetPositions	function	Return positions in ParticleState.
SetColor	function	Set color at index of ParticleState.
SetPosition	function	Set position at index of ParticleState.
Clone	function	Return a copy of this ParticleState.
stateTexture	Texture2D	The texture to construct this state from (used to color each vertex if mesh is used).
stateDepthmap	Texture2D	The texture to use as depthmap for this state. A grayscale image of the same size as stateTexture is required.

Name	Type	Description
stateDepthmapStrength	float	How much the grayscale from stateDepthmap will affect z-value.
stateMesh	Mesh	The mesh used to set this state's positions. Positions will be calculated per vertex.
stateName	String	The name of this state.
stateScale	float	The scale of this state (measured in units).
stateOffset	Vector3	The offset of this state in world- or local- (when stateTransform is used) position (measured in units).
stateTransform	Transform	The transform that will act as parent to this state.
colorLength	int	The length of color array.
positionLength	int	The length of position array.

Vector3AnimationCurve / Vector3AnimationCurveC

Holds AnimationCurves in X, Y and Z variables.

Name	Type	Description
Evaluate	function	Return a vector3 at time.
Clone	function	Return a copy of this Vector3AnimationCurve
x	AnimationCurve	AnimationCurve for X-axis.
y	AnimationCurve	AnimationCurve for Y-axis.
z	AnimationCurve	AnimationCurve for Z-axis.

WorldObject / WorldObjectC

Holds information about a World Object.

Name	Type	Description
gameObject	GameObject	The GameObject of this World Object.
transform	Transform	The Transform of this World Object.
rigidbody	Rigidbody	The Rigidbody of this World Object.
renderer	Renderer	The Renderer of this World Object.
cachedId	int	The id of this World Object (used to keep track when this object changes).
mesh	mesh	The mesh of this World Object.
vertexPositions	Vector3[]	The vertices of this World Object.
normals	Vector3[]	The normals of this World Object.

ManipulatorObject / ManipulatorObjectC

Holds information about a Manipulator Object. Note that any C# specific classes and enums ends with a ”C”.

Name	Type	Description
Contains	function	Check if manipulator contains position.
Clone	function	Clone this ManipulatorObject.
type	MANIPULATORTYPE	The type of this manipulator. Available values: None, Attractor, AttractorGravitational, Repellent
property	ManipulatorProperty	The property settings (if type is property).
affects	LayerMask	The layers this manipulator will affect.
transform	Transform	The transform of this manipulator.
shape	MANIPULATORSHAPE	The shape of this manipulator (sphere or box).
size	float	The size of this manipulator.
bounds	Bounds	The bounds of this manipulator (if shape is box).
strength	float	The strength of this manipulator.
enabled	boolean	Is this manipulator enabled?

ManipulatorProperty / ManipulatorPropertyC

Holds information about a Manipulator Property. Note that any C# specific classes and enums ends with a ”C”.

Name	Type	Description
Clone	function	Clone this ManipulatorProperty.
type	MANIPULATORPROPERTYTYPE	The type of this ManipulatorProperty.
transition	MANIPULATORTPROPERTYRANSITION	The transition of this ManipulatorProperty.
velocity	Vector3	The velocity of this ManipulatorProperty.
color	Color	The color of this ManipulatorProperty.
size	float	The size of this ManipulatorProperty.
targets	List.<Transform>	The target transforms to position towards.
targetPointer	int	The next target pointer (self incremented).

Name	Type	Description
targetId	int	The target id for this manipulator. This value pairs a particle with a manipulator.
useLocalRotation	boolean	Should the manipulator's transform direction be used to apply velocity?
onlyColorInRange	boolean	Should the particles go back to original color when out of range?
keepColorAlphas	boolean	Should the particles keep their original alpha?
onlyPositionInRange	boolean	Should the particles stop positioning towards target when out of range?
zeroVelocityStrength	float	The strength to zero velocity on target positioning when using transitions.

PlaygroundCollider / PlaygroundColliderC

Holds information about a Playground Collider.

Name	Type	Description
Clone	function	Clone this PlaygroundCollider.
UpdatePlane	function	Updates this PlaygroundCollider's plane.
enabled	boolean	Is this PlaygroundCollider enabled?
transform	Transform	The transform that makes this PlaygroundCollider.
plane	Plane	The plane of this PlaygroundCollider.

ParticleProjection / ParticleProjectionC

Holds information about a Particle Projection object.

Name	Type	Description
Initialize	function	Initialize this ParticleProjection object.
Construct	function	Build the source data.
Update	function	Update the projection.
GetColor	function	Return color at index.
GetPosition	function	Return position at index.
GetNormal	function	Return normal at index.
GetParent	function	Return parent at index.
projectionTexture	Texture2D	The texture to project.

Name	Type	Description
projectionOrigin	Vector2	The origin offset in Units.
projectionTransform	Transform	Transform to project from.
projectionDistance	float	The distance in Units the projection travels.
projectionScale	float	The scale of projection in Units.
projectionMask	LayerMask	Layers seen by projection
surfaceOffset	float	The offset from projected surface.
liveUpdate	boolean	Determines whether this projector should update each frame.

PlaygroundParticles.js / PlaygroundParticlesC.cs

These are the public variables exposed on a Particle Playground System. Note that any C# specific classes and enums ends with a ”C”.

Variable Name	Type	Description
source	SOURCE	The particle source. This determines how particles are initially created. Available values: State, Transform, WorldObject, SkinnedWorldObject, Script
sourceDownResolution	int	The source distribution over vertices used to down resolution a skinned mesh.
activeState	int	Current active state (when using state as source).
transition	TRANSITION	The type of transition to use when switching activeState. Available values: None, Lerp, Fade, Fade2
transitionTime	float	The time it takes to complete a transition.
emit	boolean	If emission of particles is active on this PlaygroundParticles.
loop	boolean	Should the emission cycle loop or just run one time after initiation?
disableOnDone	boolean	Should the GameObject disable when first lifetime cycle has finished?
updateRate	int	The rate to update this PlaygroundParticles.
calculate	boolean	Calculate forces on this PlaygroundParticles (can be overrided by Playground.calculate).

Variable Name	Type	Description
calculateDeltaMovement	boolean	Calculate the delta movement force of this particle system.
deltaMovementStrength	float	The strength to multiply delta movement with.
worldObjectUpdateNormals	boolean	If the current world object will change its vertices over time enable this to get the updated normals.
nearestNeighborOrigin	int	The initial source position when using lifetime sorting of Nearest Neighbor / Nearest Neighbor Reversed
particleCount	int	The amount of particles within this PlaygroundParticles object.
overflowOffset	Vector3	Offset when particle count exceeds source count.
overflowMode	OVERFLOWMODE	The method to calculate overflow with.
applySourceScatter	boolean	Should source position scattering be applied?
sourceScatterMin	Vector3	The minimum spread of source position scattering.
sourceScatterMax	Vector3	The maximum spread of source position scattering.
emissionRate	float	The percentage to emit of particleCount in bursts from this PlaygroundParticles.
sorting	SORTING	Sort mode for particle lifetime. Use this to alter the initial structure of the particles at birth. Available values: Scrambled, ScrambledLinear, Burst, Linear, Reversed, NearestNeighbor, NearestNeighborReversed
sizeMin	float	Minimum particle size.
sizeMax	float	Maximum particle size.
scale	float	The scale of minimum- and maximum size.
rotationSpeedMin	float	Minimum amount to rotate.
rotationSpeedMax	float	Maximum amount to rotate.
lifetime	float	The life of a particle in seconds.
lifetimeSize	AnimationCurve	The size over lifetime of each particle.
lifetimeOffset	float	The offset of lifetime cycles. Use this to offset particle systems lifetime with each other.
onlySourcePositioning	boolean	Should the particles only position on their source (and not apply any forces)?
applyLifetimeVelocity	boolean	Should lifetime velocity affect particles?

Variable Name	Type	Description
lifetimeVelocity	Vector3AnimationCurve	The velocity over lifetime of each particle.
applyInitialVelocity	boolean	Should initial velocity affect particles?
initialVelocityMin	Vector3	The minimum starting velocity of each particle.
initialVelocityMax	Vector3	The maximum starting velocity of each particle.
applyInitialLocalVelocity	boolean	Should initial local velocity affect particles?
initialLocalVelocityMin	Vector3	The starting minimum velocity of each particle with normal or transform direction.
initialLocalVelocityMax	Vector3	The starting maximum velocity of each particle with normal or transform direction.
applyInitialVelocityShape	boolean	Should the initial velocity shape be applied on particle re/birth?
initalVelocityShape	Vector3AnimationCurve	The amount of velocity to apply of the spawning particle's initial/local velocity in form of a Vector3AnimationCurve.
applyVelocityBending	boolean	Should bending affect particles velocity?
velocityBending	Vector3	The amount to bend the velocity path of each particle.
gravity	Vector3	The constant force towards gravitational vector.
damping	float	Particles inertia over time.
maxVelocity	float	Maximum allowed velocity magnitude.
lifetimeColor	Gradient	The color over lifetime.
axisConstraints	PlaygroundAxisConstraints	The force axis constraints of each particle. Set by x, y and z booleans.
colorSource	COLORSOURCE	The source to read color from (fallback on Lifetime Color if no source color is available)
sourceUsesLifetimeAlpha	boolean	Should the source color use alpha from Lifetime Color instead of the source's original alpha?
collision	boolean	Can particles collide?
affectRigidbodies	boolean	Should particles affect rigidbodies?
mass	float	The mass of a particle (calculated in collision with rigidbodies).
collisionRadius	float	The spherical radius of a particle.
collisionMask	LayerMask	The layers these particles will collide with.
lifetimeLoss	float	The amount a particle will loose of its lifetime on collision.

Variable Name	Type	Description
bounciness	float	The amount a particle will bounce on collision.
bounceRandomMin	Vector3	The minimum amount of random bounciness (seen as negative offset from the collided surface's normal direction).
bounceRandomMax	Vector3	The maximum amount of random bounciness (seen as positive offset from the collided surface's normal direction).
colliders	PlaygroundCollider	The Playground Colliders of this particle system.
states	List.<ParticleState>	The states of this PlaygroundParticles. A state is a snapshot from an image or mesh with position- and color data.
worldObject	WorldObject	A mesh calculated within the scene.
skinnedWorldObject	SkinnedWorldObject	A skinned mesh calculated within the scene.
sourceTransform	Transform	A transform calculated within the scene.
paint	PaintObject	The paint source of this PlaygroundParticles.
projection	ParticleProjection	The projection source of this PlaygroundParticles.
manipulators	ManipulatorObject	The list of manipulators handled by this particle system.
playgroundCache	PlaygroundCache	The internally used data for each particle.
particleCache	ParticleCache	The internally used particle pool.

Please see PlaygroundParticles.js or PlaygroundParticlesC.cs for available internal functions for a PlaygroundParticles/C object.

Support

For questions, requests and bug-reporting please send a mail to support@polyfied.com.
Please visit http://playground.polyfied.com/ for more information.

Particle Playground Version: 1.20
Document updated: April 20, 2014

Polyfied. Stockholm, Sweden 2014.
polyfied.com