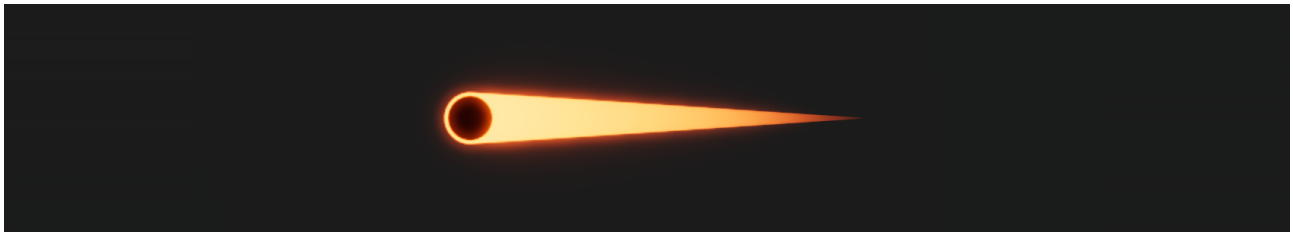


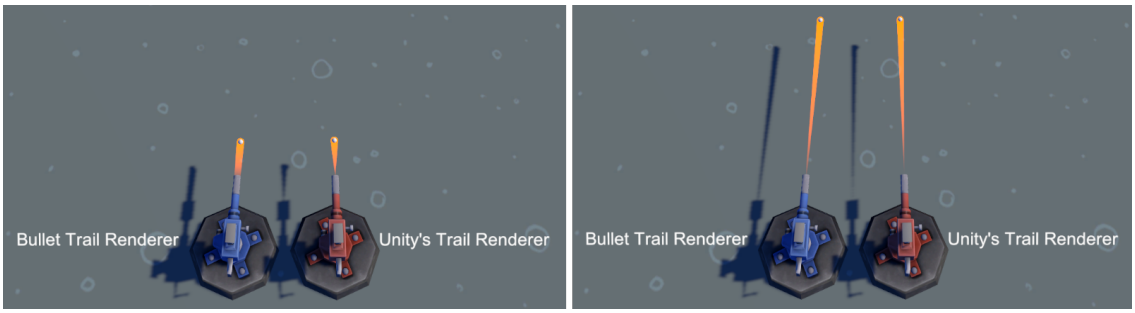
Bullet Trail Renderer

EpsilonDelta

1 Introduction

BulletTrailRenderer component renders a trail behind a moving object. There are two reasons why it's better for bullet trails than standard Unity's trail and also why I created this asset:

1. BulletTrailRenderer provides Width Over Time and Color Over Time effects. These are far better for bullets: more realistic and better looking. Standard Unity TrailRenderer only provides these effects over length and is unable to render bullet trail properly.



Unity built-in TrailRenderer cannot achieve effect of the blue turret - its trail has always or never a thin tail. BulletTrailRenderer can start with wide tail that gets thinner over time

2. BulletTrailRenderer uses simple but more suitable mesh generation strategy for bullet trails.

I also needed hundreds of trails and none of the other trail assets from asset store can do that due to a lot of overhead. Bullet trail renderer can do thousand simple trails with around 100 fps on my laptop.

2 Contact

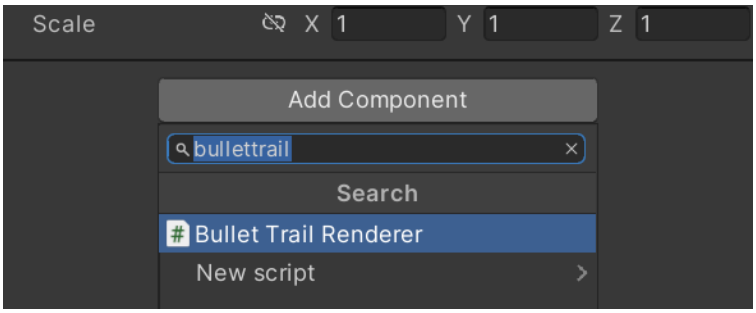
Please, **rate and leave a review** in the Asset Store if you are satisfied with the product or if you have a feature request or any other problem. It will help me to improve this asset. You can also contact me via:
Email: epsildel@gmail.com
Github repo for issues: https://github.com/EpsilonD3lta/Bullet-Trail-Renderer

3 Requirements

Can be used with Unity 2021.3.17 or newer. No other requirements.

4 Setup

There isn't any technical setup, just import the asset from Unity Asset Store. You can start using BulletTrailRenderer as any other component.



4.1 Demo

There is a **Demo** packed in .unitypackage for three different render pipelines: Built-in, High Definition and Universal. Unpack the one based on what render pipeline your project is using. Demo has dependencies on Shader Graph and Unity UI packages. If you don't have them, you can install them in Unity Package Manager.

5 How it works

BulletTrailRenderer works as any trail renderer - checks for changes of its GameObject's position every frame and generates mesh accordingly. Mesh is calculated on CPU. BulletTrailRenderer is compatible with Built-in Render Pipeline, URP and HDRP. Should in be theory compatible with any custom SRP - the SRP just have to properly implement beginCameraRendering event.

5.1 Algorithm

1. Trail is composed of points, part between two points is regarded as a trail segment. Each point has two vertices, each segment therefore has four vertices which together create two triangles.
2. When trail is first created (with first slightest movement) two initial points are added (first segment) - tail and head point.
3. When GameObject moves, head point is moved with it. If the head point is too far away from previous point or the angle of movement (frame delta position) with previous segment is too big, new (head) point is emitted and old head point stays fixed in place
4. Trail has a lifeTime, each point's life starts in the moment it is fixed in place and exists for duration equal to Trail's *lifeTime* property. During it's life, the width and vertex color of this point and its vertices are governed by width curves and color gradients (over time and over length) that are set in the inspector. When point's remaining lifetime expires, it is removed from trail. Head point's remaining lifetime is refreshed when it moves.

5.2 Rendering

BulletTrailRenderer uses Unity's Graphics.RenderMesh() API to draw the mesh. It does not use MeshRenderer and MeshFilter. You can have multiple instances on one GameObject and you can have MeshRenderer and BulletTrailRenderer on one GameObject. Dynamic batching and SRP batcher is supported out of the box by using Graphics.RenderMesh().

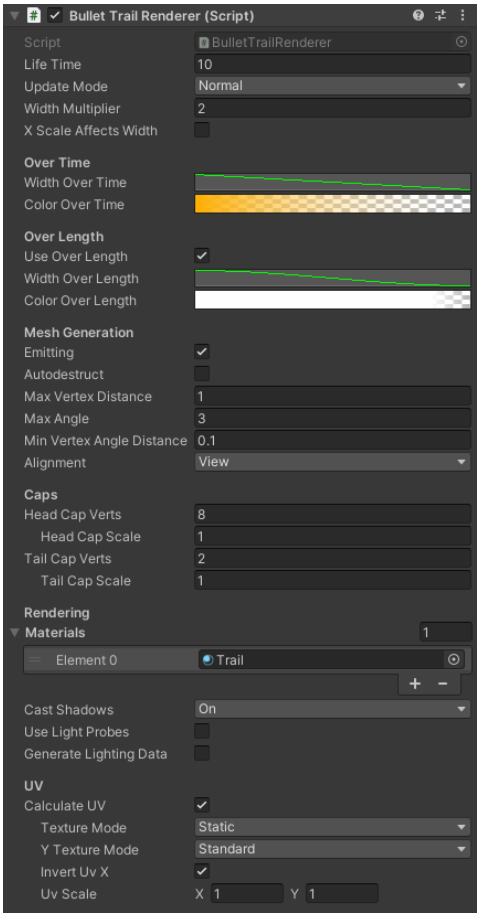
6 Manual

6.1 How to use

Workflow is exactly the same as in Unity's built-in TrailRenderer. Add BulletTrailRenderer component as any other component to any GameObject. Now, when you move GameObject either in PlayMode or in EditMode there should be a trail visible. It should work out of the box, maybe you need to set a material to render properly, depends on your version of Unity. Note that the material (its shader) has to support vertex colors if you want to use *ColorOverTime* and *ColorOverLength* gradients. There are example materials and shaders for each render pipeline in provided Demo - shaders were created in ShaderGraph.

6.2 Component description

- **LifeTime**: How long does new trail segment (more precisely it's newly emitted point) stay alive
- **UpdateMode**:
  a) **Normal**: uses *Time.deltaTime*
  b) **UnscaledTime**: uses *Time.unscaledDeltaTime* and works when game is paused (*Time.timeScale* is 0)
- **WidthMultiplier**: Trail width. Multiplied with *WidthOverTime* and optionally *WidhtOverLength* curves
- **XScaleAffectsWidth**: If turned on, width is multiplied with transform's "x" scale
- **WidthOverTime**: Trail width at a point over the point's lifetime. Horizontal axis is normalized lifetime and vertical axis is width multiplier. Curve editor is clamped to interval [0, 1] and to minimum value of 0
- **ColorOverTime**: Vertex color over point's lifetime. Material's shader needs to support vertex colors
- **UseOverLength**: Whether to use OverLength effects. Turn off if you don't need over length effects to save some overhead



- **WidthOverLength**: Trail width based on point's normalized distance from head. Horizontal axis is normalized distance and vertical axis is width multiplier. Curve editor is clamped to interval [0, 1] and to minimum value of 0
- **ColorOverLength**: Vertex color based on point's normalized distance from head. Material's shader needs to support vertex colors
- **Emitting**: Pause/unpause trail generation
- **Autodestruct**: Destroy GameObject after trail vanishes (only destructs when emitting is on - same as Unity built-in TrailRenderer)
- **MaxVertexDistance**: Emit new trail point when distance from previous point is more this value
- **MaxAngle**: Emit new trail point when frame delta position angle with last segment is more than this value
- **MinVertexAngleDistance**: Test against *MaxAngle* only when distance from last point is more this value
- **Alignment**:
  a) **View**: Face Camera
  b) **TransformZ**: Trail mesh aligns with XY Plane
- **HeadCapVerts**: How many vertices to add to head end
- **TailCapVerts**: How many vertices to add to tail end
- **HeadCapScale**: Scale cap alongside trail
- **TailCapScale**: Scale cap alongside trail
- **Materials**: Materials with which to render mesh. Their shader should support vertex color for *ColorOverTime/Length* effects. Multiple materials render the same trail mesh multiple times
- **CastShadows**: Shadow casting mode, exactly the same as Unity's mesh/trail renderer options
- **UseLigthProbes**: Whether to use light probes to illuminate mesh. Works only if lit shaders are used
- **GenerateLightingData**: Toggle generation of normals and tangents, for use in lit shaders. Turn off for less overhead. Note that *GenerateLightingData* only works correctly when UVs are calculated, therefore they are autmatically turned on if *GenerateLightingData* is on

- **ReceiveShadows**: Only for Built-in Render pipeline. For Scriptable Render Pipelines set this in material
- **CalculateUV**: Whether to calculate UVs. Turn off for less overhead if you don't need UVs
- **TextureMode**: UVs along trail's length. Modes are the same as in Unity's built-in TrailRenderer. Note that for Static mode, some changes in settings (that somehow affects UVs calculation) will be applied only to the new trail segments, created after changing the particular setting
- **YTextureMode**: UVs along trail's width
  a) **Standard**: UVs are mapped to actual quad
  b) **Fixed**: UVs are always mapped to quad as if it had full width
- **InvertUvX**: For compatibility with Unity's TrailRender UV orientation along trail's length
- **UvScale**: UVs tilling scale

6.3 Notes

- The smaller the *MaxVertexDistance*, *MaxAngle*, and *MinVertexAngleDistance* are, the finer is the trail. Note that too fine trail can have performance impact.
- If you want to trail to update in lockstep with physics, just change LateUpdate to FixedUpdate method in the script.
- BulletTrailRenderer (and especially it's mesh generation) is meant for trails behind fast moving objects in relatively straight lines. But it can serve as a general trail renderer with additional features. Also you can modify the source code as opposed to built-in trail renderer to exactly match your needs. I tried to comment every tricky code for easier modification.

6.4 Scripting

BulletTrailRenderer provides API to manipulate trail via scripts. API is designed for ease of use, but you can change the script to get direct access to anything.
As explained in How it works section, BulletTrailRenderer emits points. Point is saved as a custom struct (named Point). Trail keeps these points in a custom point array together with some other internal properties. Important properties of a Point for the outside manipulations are Position and TimeCreated which enable you to modify position and remianing lifetime of a Point. The remaining lifetime of a Point is calculated like this:

$$RemainingLifeTime = LifeTime - (ElapsedTime - TimeCreated)$$

ElapsedTime is a trail's internal time which increases every frame (by unscaled/deltaTime) if trail is enabled and has some points. *TimeCreated* is a timestamp in this trail's internal time when point is emitted (more precisely when it's fixed in place). Points are always younger the closer they are to the beginning of the trail (head).
You can modify points via provided methods and their overloads:

- **CreatePoint**: Helper methods to create a point struct. Point is not added to the trail with this method, for that use AddPoint methods
- **GetPoints**: Get list of all trail's points
- **GetPoint**: Get point at index
- **AddPoints**: Add points to the beginning (head) of trail
- **AddPoint**: Add point to the beginning of trail
- **SetPoints**: Set your custom list of points. Clears any previous points
- **SetPoint**: Manipulate point at index
- **Teleport**: Move transform with its trail
- **Clear**: Clears mesh and points

BulletTrailRenderer updates trail's points in LateUpdate and then later calculates and renders meshes in camera callbacks. You can :

1. manipulate trail in Update - before the trail own calculations
2. or manipullate trail using its event *pointsUpdated* - after the trail's own calculations but before mesh creation and rendering

There is an example script in Demo that applies gravity to the trail.

Notes:

- Point has a TrailBreak property, which is set to true for the last point when trail emitting property is set to false
- Besides point manipulation API, all the inspector fields are exposed as properties (lifeTime, updateMode, ...)
- When using get/set on materials array property via script, actual array is returned and not a copy as in Unity Renderers

6.5 Profiling

To properly profile BulletTrailRenderer in Editor you have to close all Scenes tabs - this is because some trail mesh properties are calculated for each camera individually, but common data are only calculated once for all cameras when first camera renders a frame. The first camera to render in Editor might be a Scene camera so this overhead would be hidden from PlayerLoop. Alternatively uncomment "#define PROFILE_IN_EDITOR" in the script or profile in Build.

6.6 Limitations

1. Sorting Layer and Order in Layer are not supported because they are not currently supported by Unity when using Graphics.RenderMesh()
2. When moving GameObject with trail in editor in Scene View with move handles, you might experience weird mesh generation sometimes. This is caused by move handles implementation - it seems their movement is not precise, but rather jagged. This together with used mesh generation strategy can create unwanted artifacts. This does not happen in Build/PlayMode when trail's GameObject is moving normally in gameplay. But keep in mind that this trail mesh generation strategy is specifically designed for bullets or for moving objects in relatively straight lines.

7 Asset structure

There is only one script in BulletTrailRenderer/Scripts/ folder called BulletTrailRenderer.cs. Folder also contains assembly definition file (.asmdef). All other files are related to Demo and are not necessary. You can move BulletTrailRenderer folder wherever you like (though you should of course avoid some special Unity folders like Streaming assets etc.). You can also move the script however you like, even delete the .asmdef file. The script is encapsulated in its own namespace so there shouldn't be any naming clashes.

8 Known Issues

1. All the listed limitations in Manual section

9 Changelog

v1.0.0

- First release