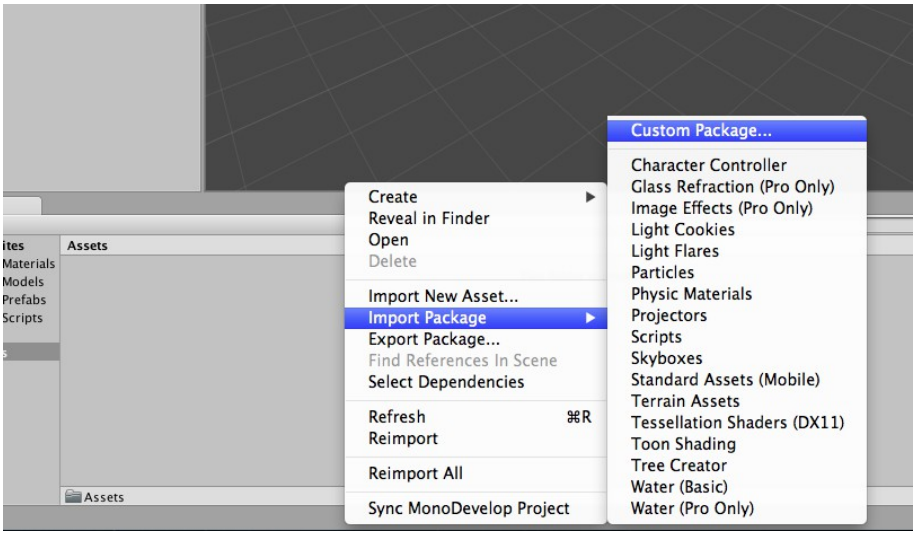
# Quickstart Guide

Include the Ara package in your project. Right click in the project window, “Import Package” “Custom Package”, then select AraTrails.unitypackage.




To add a trail to your object, simply add the AraTrail component to it.

# Support / Contact

If you have any suggestions, questions or issues, drop us a line:

ara@virtualmethodstudio.com

# User manual

Ara consists of a single component: AraTrail. It works in a very similar way to Unity's built-in trail renderer, extending and improving certain parts of it.

The inspector is divided in multiple sections (Overall, Rendering, Texturing, and so on). Parameters are grouped together depending on their context. For instance, all parameters that control trail emission are grouped under “Emission”.

## Overall

*Section:*

Cross-section asset extruded along the trail's path. If no custom section is provided, the trail will render a triangle strip. This is best used in conjuction with Local Alignment. Note that the High quality corners and corner roundness parameters will have no effect when using a custom section, they only work for the default one (triangle strip).

You can create your own section assets by right clicking in any project folder, and selecting Create->Ara Trails->Trail Section.

*Space:*

determines the reference frame used to simulate and render the trail. By default this is done in world space. However in certain situations it is useful keep the trail in local space, for instance smoke from a gun in a first person shooter: you want the trail to always go up, even if the player rotates the character around.

*Alignment:*

determines the facing direction of the trail.

- View: the trail always faces the camera.
- Velocity: the trail always faces the direction of the transform's velocity vector. All of the trail will rotate to reflect the new forward direction.
- Local: the trail always faces the direction of the transform's forward vector at the time of emission. This is useful for sword trails, for instance.

*Thickness*:

Master thickness control. The actual thickness is calculated by multiplying all thickness values: initial thickness, thickness over time, and this.

*Smoothness:*

**Amount of smoothing applied to the trail (uses Catmull-Rom spline interpolation). Higher values will yield a smoother curve. This is specially useful for fast moving trails, or trails with few segments.**

*High quality corners:*

**Enables high-quality calculation of trail corners. Specially useful for line rendering or low-res trails.**

*Corner roundness:*

**Set to 0 for sharp corners, 1 for blunt corners and any value > 1 for round corners.**

# Rendering

*Materials:*

List of materials used to render the trail. The trail will be rendered once per material.

*Cast Shadows:*

Shadow casting mode used when rendering the trail.

*Receive Shadows:*

Should this trail receive shadows from other objects? (if the shader allows it)

*Use light probes:*

Whether or not to use light probes for rendering (if the shader allows it)

# Texture

*Quad Mapping:*

Whether to use quad mapping (enabled) or classic texture mapping (disabled). Quad mapping will eliminate texture distortion caused by triangles of varying size. Must be used together with a shader that makes use of tex2Dproj() to read textures. (see the included Unlit/PerspectiveMapping shader for an example of this).

*Texture Mode:*

Determines how the texture is applied to the trail:

- Stretch: the texture is stretched to cover the entirety of the trail. You can control how many times the texture is repeated along the trail by using the "Uv Factor" parameter (see below)
- Tile: the texture will be repeated at a regular interval along the trail. You can control the size of each tile by using the "Uv Factor" parameter (see below)

*Uv Factor:*

Controls how the texture coordinates are generated for each texture mode. Behaves differently depending on the Texture Mode currently selected.

- For "stretch" it determines how many  times the texture is repeated along the trail's length.
- For "tile" it determines the size of each tile.

*Uv Width Factor:*

Same as Uv Factor, but relative to the trail's width.

*Tile Anchor:*

Only used when Texture Mode is set to "tile". Determines the starting point of the tiles along the trail:

- A value of 0 means the tiles will start at the source of the trail.
- A value of 1 means the tiles will start at the end of the trail.

Values between 0 and 1 are also valid.

## Emission

*Emit:*

**Disable this to stop emiitting trail. Enable it again to resume emission._Doing this will result in multiple disjoint trail segments, all lenght-related parameters (thickness over lenght, color over lenght) will refer to the lenght of each individual segment.**

*Initial thickness:*

**Thickness of the trail points when first emitted. Does not affect already emitted trail points.**

*Initial color:*

**Color of the trail points when first emitted. Does not affect already emitted trail points.**

*Initial velocity:*

**Velocity of the trail points when first emitted. Does not affect already emitted trail points.**

*Time interval:*

**Minimum amount of time that should pass between emitted points.**

*Min Distance :*

**Minimum distance that should be kept between emitted points.**

*Time:*

**Duration of the trail in seconds.**

## Time

*Thickness over time:*

**This curve determines the thickness of the trail over time. Note that this value is multiplied by thickness over lenght (see "Lenght" section) and the master thickness control (see "Overall" section).**

*Color over time:*

**This curve determines the color of the trail over time. Note that this value is multiplied by color over lenght (see "Lenght" section) and the initial color control (see "Emission" section).**

## Lenght

*Thickness over lenght:*

This curve determines the thickness of the trail along its lenght. Note that this value is multiplied by thickness over time (see “Time” section) and the master thickness control (see “Overall” section).

*Color over lenght:*

This curve determines the color of the trail along its lenght. Note that this value is multiplied by color over time (see “Time” section) and the initial color control (see “Emission” section).

## Physics

*Enable physics:*

Toggles physics on or off.

*Gravity:*

Intensity and direction of the gravity applied to the trail.

*Inertia:*

Amount of transform's speed inherited by the trail.

- If set to 0,  the trail initial velocity will not be affected by the transform movements.
- If set to 1, all of the tranform's kinetic energy will be transferred to the trail.

Values between 0 and 1 let you control the intensity of the effect.

*Velocity smoothing:*

Amount of smoothing applied to the transforms movements when calculating its velocity. If your inertia value is high but the trail movements seem choppy, increase the amounf of velocity smoothing.

*Damping:*

Amount of kinetic energy lost per second. Low values will cause the trail to never lose energy. Increasing the value will cause the trail to lose energy more quickly, eventually stopping completely when damping equals 1.

# Scripting

Here's a short guide on how to customize your Ara trails via C# scripting. Trails are made of a sequence of points (structs of type AraTrail.Point). You can modify the existing points, add new points, remove them of provide your own. Each point has the following variables:

- *position*
- *velocity*
- *tangent:* tangent vector (average of the vector from the previous point and the vector to the next point).
- *normal:* normal vector, always orthogonal to the tangent vector. Together they define the orientation of the point
- *color*
- *thickness*
- *life:* lifetime in seconds. When this value reaches zero, the point is automatically removed from the trail.
- *discontinuous:* true if the trail is discontinuous at this point (e.g, emission was switched off and then continued elsewhere), false otherwise.

Note: All of these are interpolated to generate additional points if the “smoothing” property of your trail is > 1.

# Post-processing trail

Sometimes you will want to modify your trail each frame, after it has been generated. Ara emits new trail points in LateUpdate(), then renders the trail in each camera's OnPreCull().

If you want to modify the points after they've been created but before they are rendered, you must somehow inject your code right after the trail's LateUpdate(). Ara provides a convenient way to do this: the OnUpdatePoints event.

You can subscribe to this event, and safely modify the existing trail points there.

Here's how a trail post-processing class should look. Also, see the ColorFromSpeed.cs helper script for a working example.

```
[RequireComponent(typeof(AraTrail))]
public class PostProcessTrail: MonoBehaviour {

	AraTrail trail;

	void OnEnable () {
		trail = GetComponent<AraTrail>();
		trail.onUpdatePoints += UpdatePoints;
	}

	void OnDisable () {
		trail.onUpdatePoints -= UpdatePoints;
	}

	void UpdatePoints(){
		for (int i = 0; i < trail.points.Count; ++i){
			AraTrail.Point point = trail.points[i];
			// do some custom point processing here.
			trail.points[i] = point;
		}
		// you can also add or remove points here
	}
}
```

# Providing your own points

Instead of modifying the points created by Ara, you can hijack the emission system or even provide your own point each frame. For both purposes it is best if you disable automatic emission in the trail.

To emit a new point, you can call trail.EmitPoint(Vector3 position).

To provide new points, you can access the list of points like this: trail.points

An example on how to do this:

```
[RequireComponent(typeof(AraTrail))]
public class CustomTrail : MonoBehaviour {

	AraTrail trail;

	void Awake () {
		trail = GetComponent<AraTrail>();
		trail.emit = false;
	}

	void Update(){
		trail.points.Clear();
		trail.points.Add(new AraTrail.Point(	Vector3.zero,		//position
							Vector3.zero,		//velocity
							Vector3.up,		//tangent
							Vector3.forward,	//normal
							Color.white,		//color
							1,			//thickness
							1			//lifetime
				));
	}
}
```