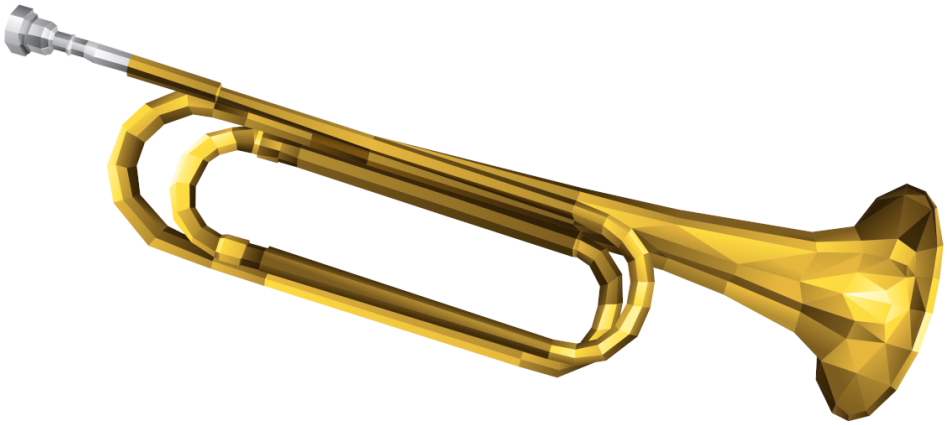


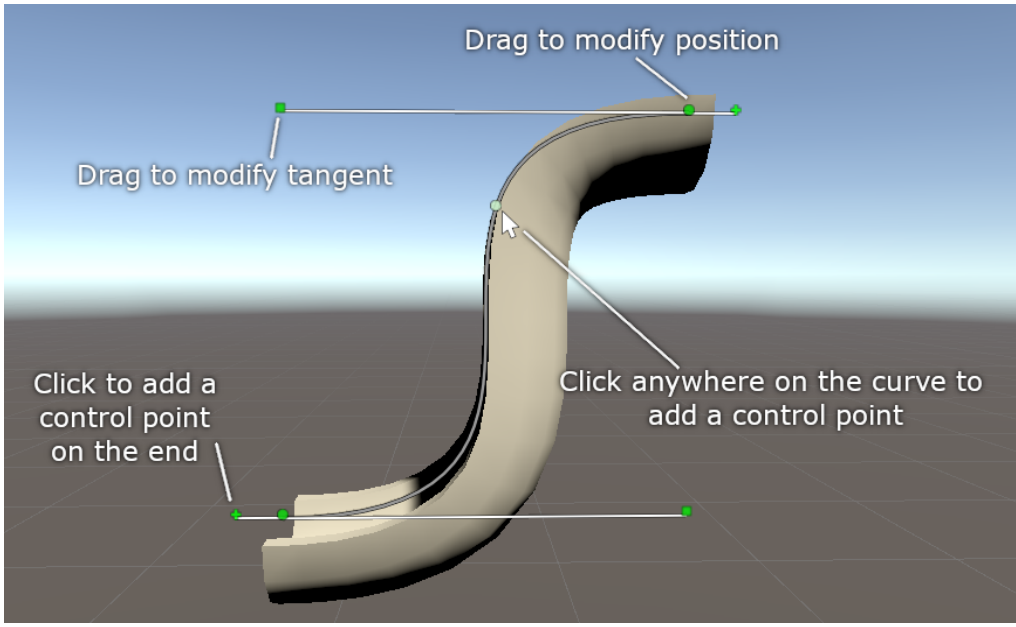
CURVE DESIGNER

About

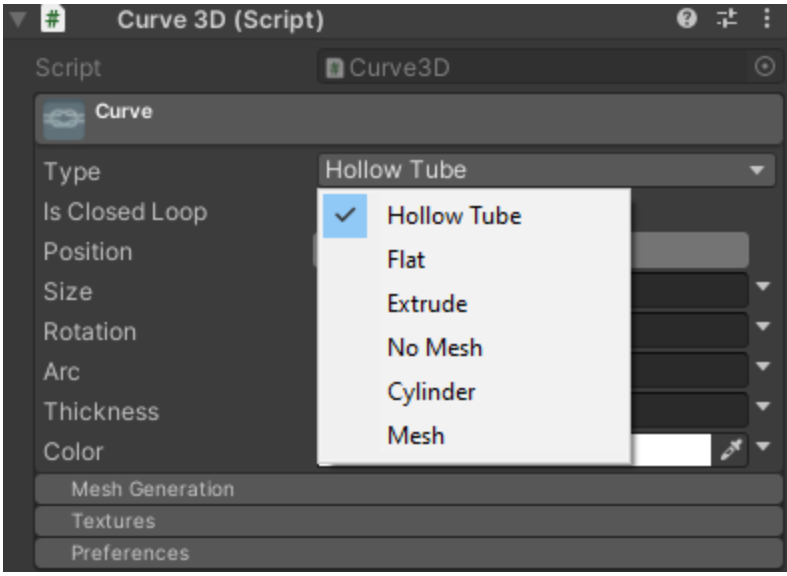
Curve Designer is an open source (MIT License) tool for quickly and easily modelling curves, tubes, ramps, roads, half-pipes and more in Unity. You can also use Curve Designer as a spline tool for drawing out paths for characters and other objects to follow.

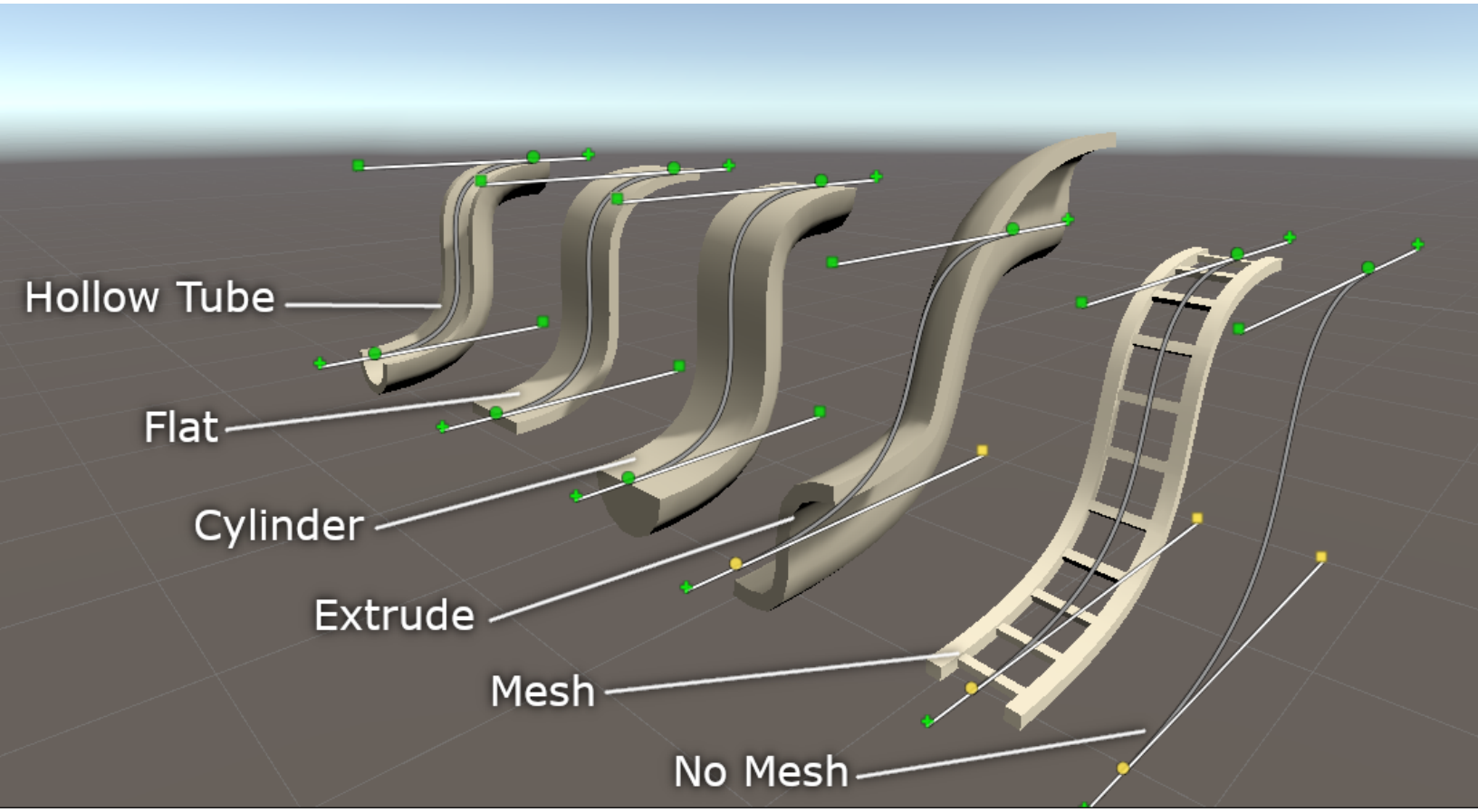
Basic Usage

To start using Curve Designer, just add the Curve3D script to a GameObject. You should see a simple curve as well as some control handles for manipulating the shape of the curve and adding control points. Make sure you have gizmos enabled! If you'd like the curve to have collision, make sure to attach a mesh collider.



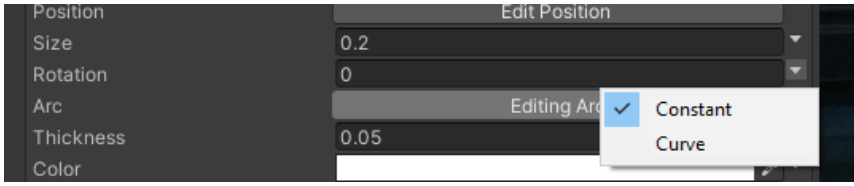
You can further customize the curve via the settings in the inspector. **Type** controls which type of curve the script will generate. **Hollow Tube** will produce a cylinder, tube or half-pipe shape. **Cylinder** produces a solid filled cylinder. **Flat** produces a flat ribbon/road shape. **Extrude** allows you to extrude a secondary curve along the length of the curve to produce highly customized curves. **Mesh** will stretch and deform a mesh that you provide along the length of the curve, perfect for train tracks or other more detailed custom meshes. **No Mesh** will not generate any mesh, useful for producing splines for objects to follow.

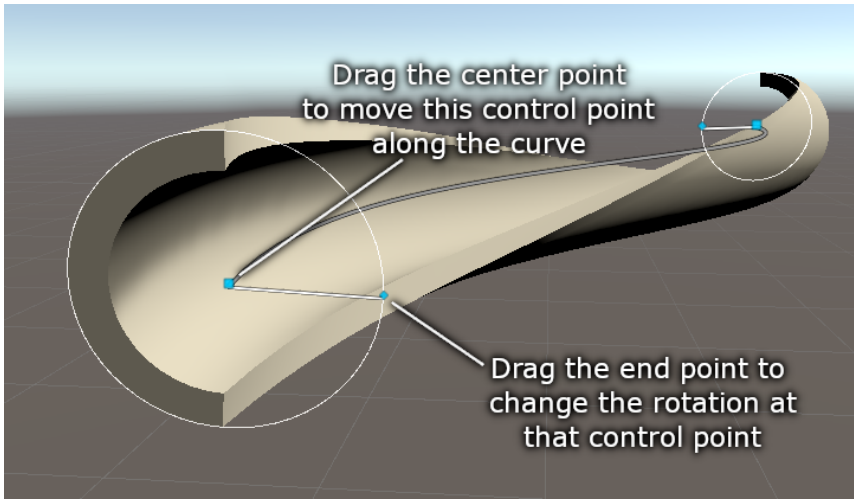


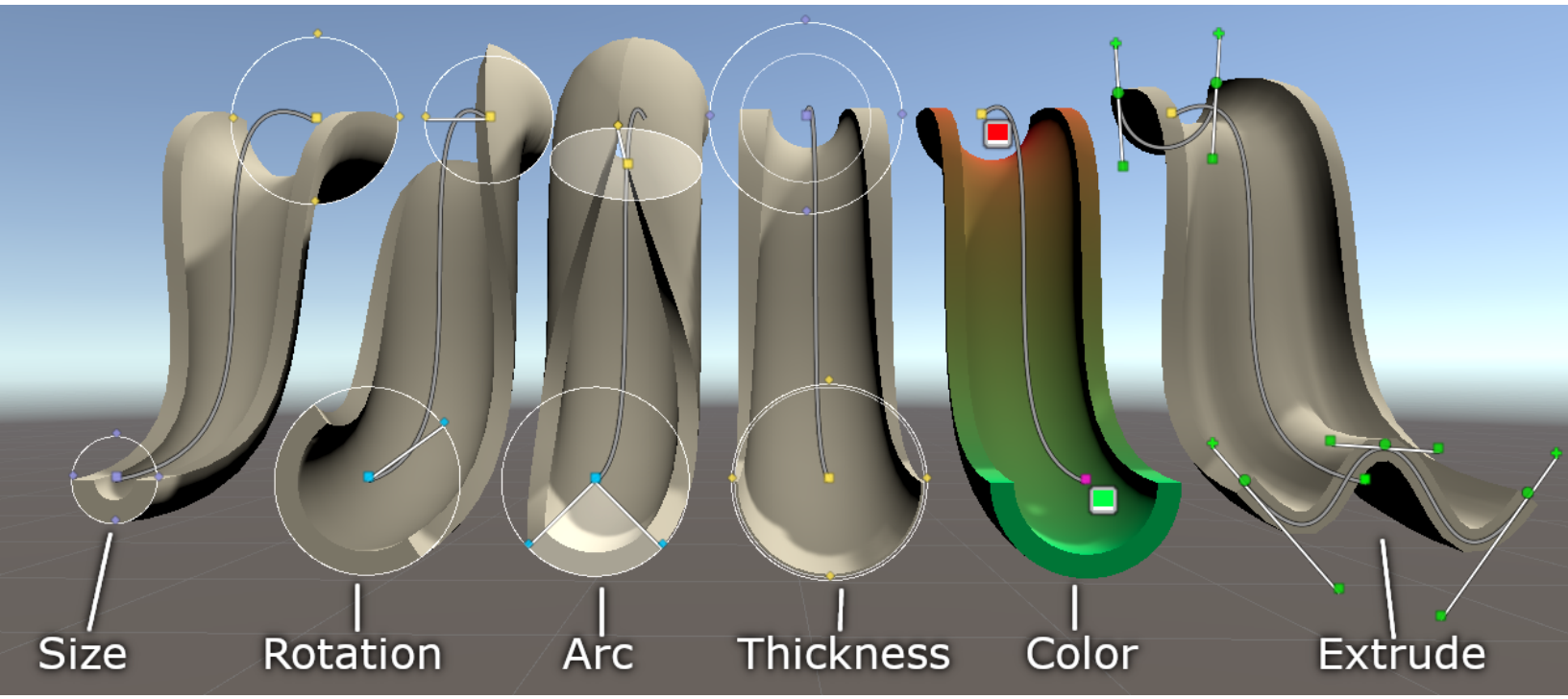


Other settings include **Size** which controls the scale of the curve, **Thickness** which controls the thickness of the walls of the curve, **Rotation** which controls the rotation around the curve itself, **Color** which controls the color stored in the vertices (note in order to display this color on the mesh you'll need to use a material which reads from the vertex color, try Art/Materials/DrawColorSurface.mat), **Arc** which controls the arc of the tube from 0-360 degrees. Depending on your selected curve type, some irrelevant settings may disappear (such as arc for non-tubes).

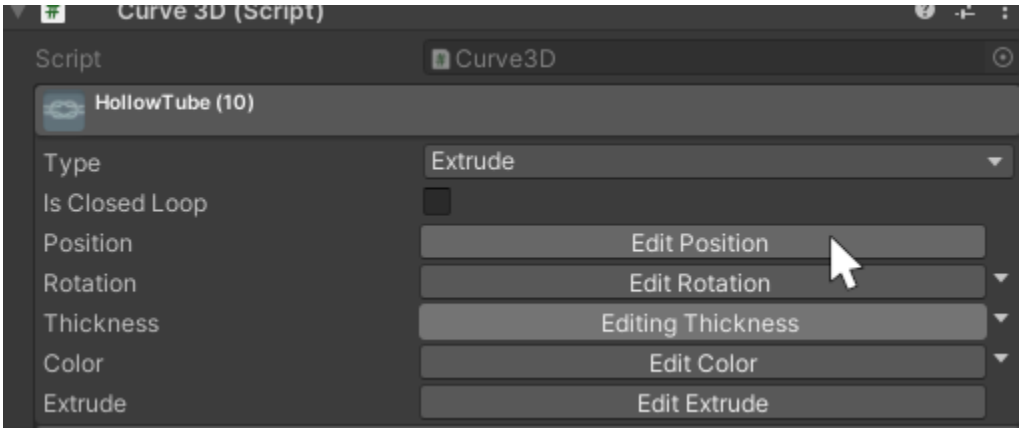
You may have noticed that all of the settings mentioned above have dropdowns next to them that allow you to select between **Constant** and **Curve**. These types of settings are called **Samplers**. This is because all of these properties can be set either uniformly along the length of the curve, or can be set at different points along the curve and interpolated between, allowing you to vary the value across the length of the curve. You can insert new sampler points by clicking on the curve.



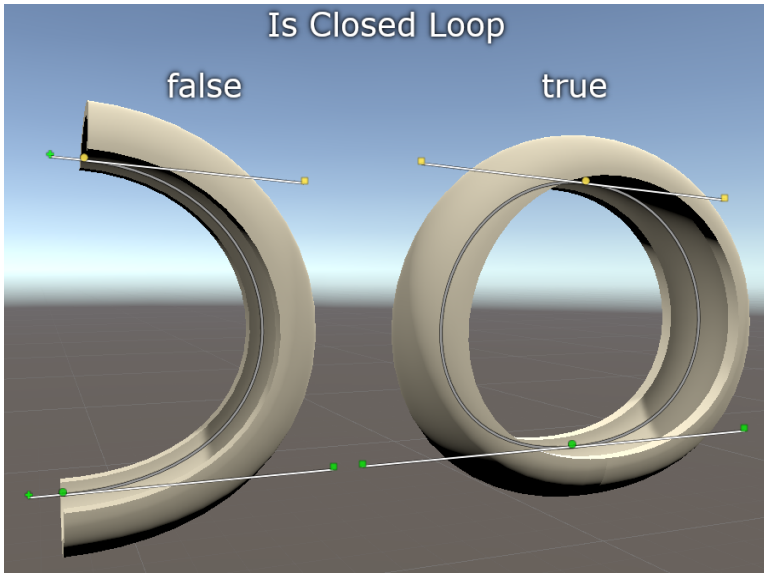




You can only edit one sampler at a time. The active sampler is labeled as “Editing” in the inspector. To switch to another sampler, click on the corresponding button. To return to editing the overall shape of the curve click “Edit Position”. You can select all the points in a sampler with ctrl+a, and delete points with del. You can multi-select points with ctrl or shift to edit multiple points at once. Ctrl selects only the points that you click, whereas shift selects the points that you click as well as all of the points in-between.

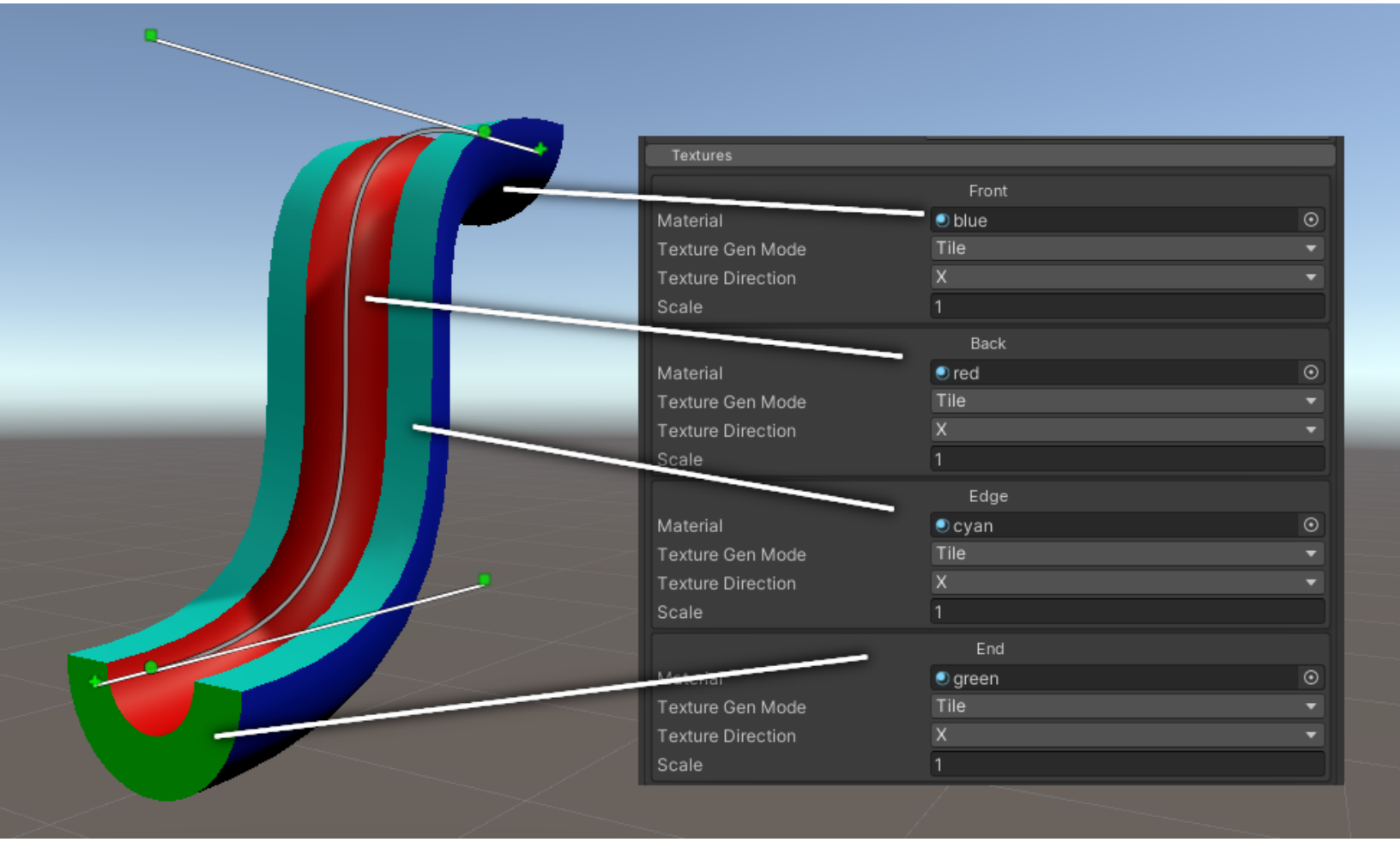


The **IsClosedLoop** toggle allows you to control whether or not the curve should be a closed loop.

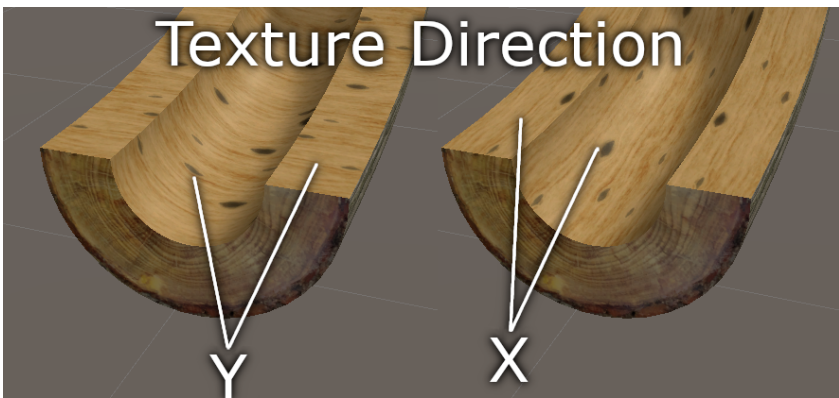


Texturing

When generating a mesh, Curve Designer generates several sub-meshes, each with their own set of uvs. This allows each surface of a curve to have its own material. You can assign materials in the inspector, or by dragging materials directly onto the model. Most curve types have 4 surfaces: front, back, edge and end (both ends share the same material). However Cylinders lack edges so they only have 3 surfaces.



You can think of all the surfaces as having an axis that runs along the surface (lengthwise), and an axis that runs across the surface (crosswise). You can control whether the lengthwise axis maps to the x or y direction of your texture with the **Texture Direction** setting. This is useful if your textures appear to be rotated by 90°.



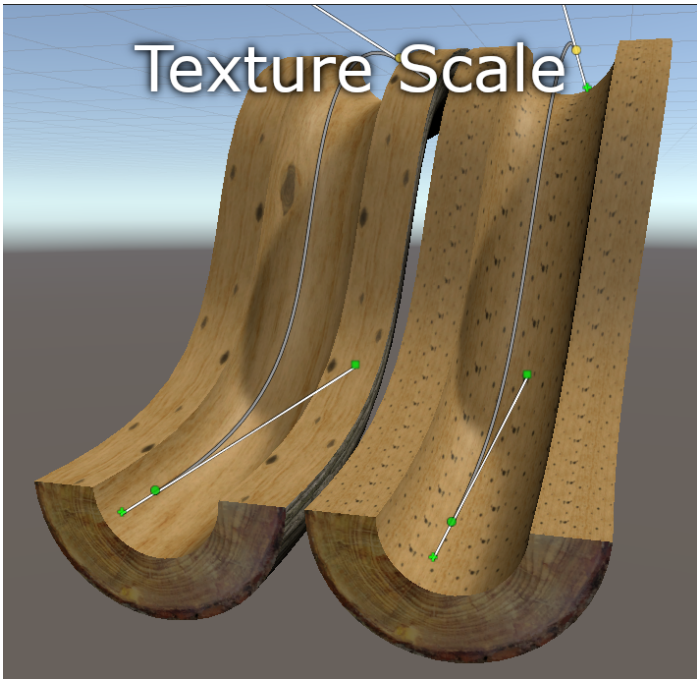
You can also scale textures up and down with the **Scale** setting.

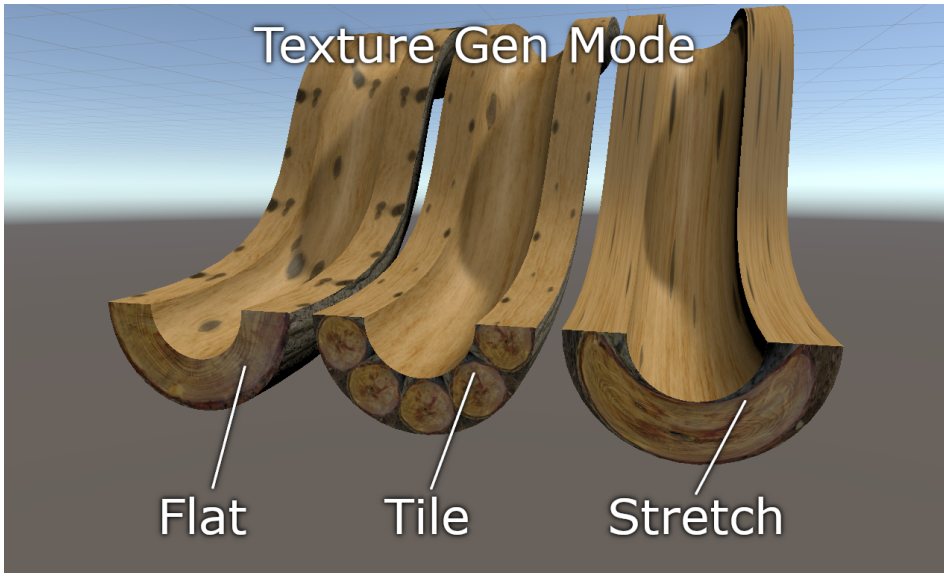
There are several different methods for applying uvs to a surface. You can control which method is used via the **Texture Gen Mode** setting. Generally you can just try out all the texture gen modes and see what produces the best looking results for your use case.

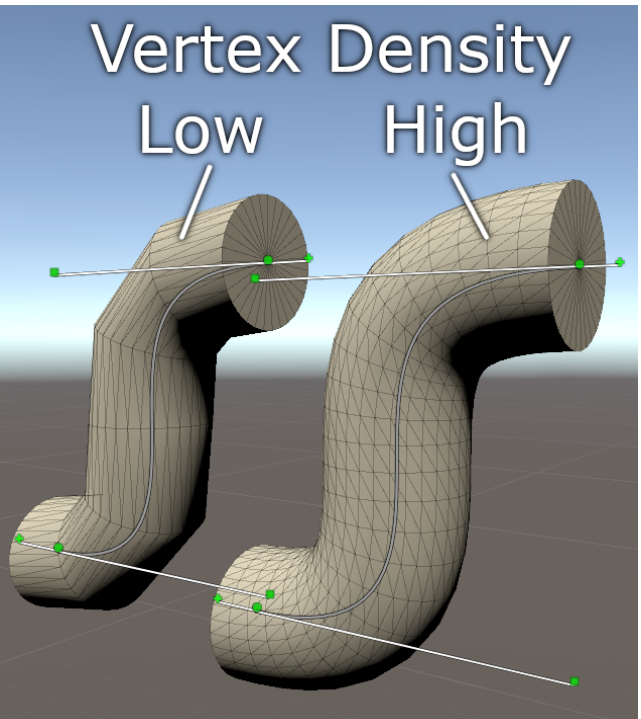
Tile stretches the crosswise axis to range from 0-1, and then scales the texture in the lengthwise axis to maintain the correct aspect ratio. This means that if the surface gets wider the texture will appear to increase in scale.

Flat is very similar to Tile, except the texture maintains a constant size regardless of surface width. This is useful if you'd like to maintain constant texture scale across a surface. Flat also has slightly specialized behaviour when used on an end surface, projecting itself onto the center of the curve rather than repeating along the surface. Flat is also the only valid Texture Gen Mode for cylinder end surfaces, and will always be used.

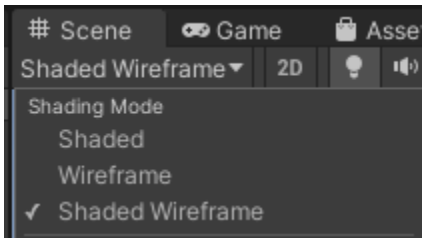
Stretch stretches the uvs across the entire surface, so they range from 0-1 in both the x and y direction. This is mostly useful for visual effects or non-repeating textures. The texture scale modifier has no effect when using stretch.







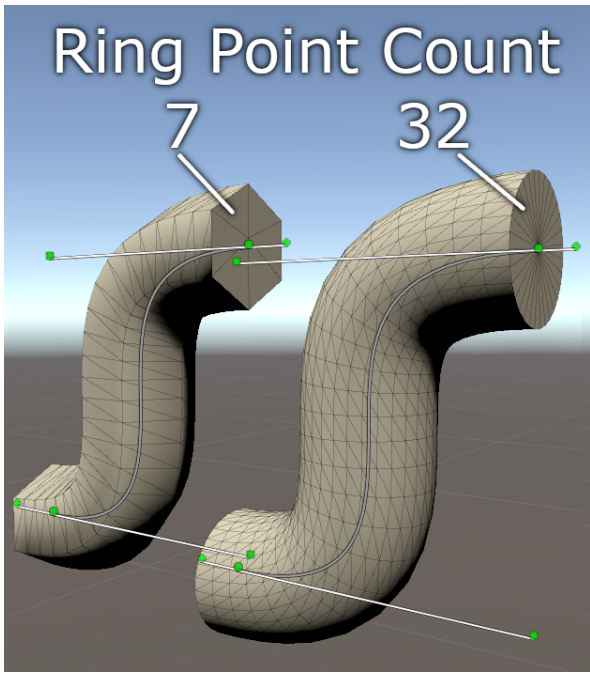
Other Settings



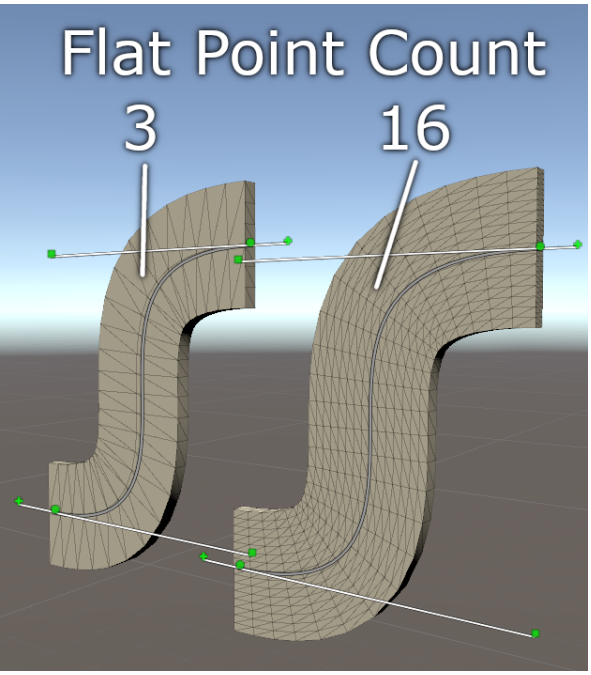
I recommend you enable Shaded Wireframe in the Scene view when modifying the following settings in order to accurately preview them.

Vertex Density controls the density of vertices running lengthwise along the curve. A higher value will produce more vertices which will make your curve look smoother, but a lower value will both generate and render faster. Unlike with Ring Point Count and Flat Point Count, this value is not an integer that specifically specifies a precise number of vertices, rather this number is multiplied by the length of the curve to determine the number of vertices along the curve to generate.

Ring Point Count controls the number of vertices in the crosswise direction for Cylinders, Extrude Curves and Hollow Tubes. However Cylinders also contain a flat surface, the vertex count of which is controlled by Flat Point Count.



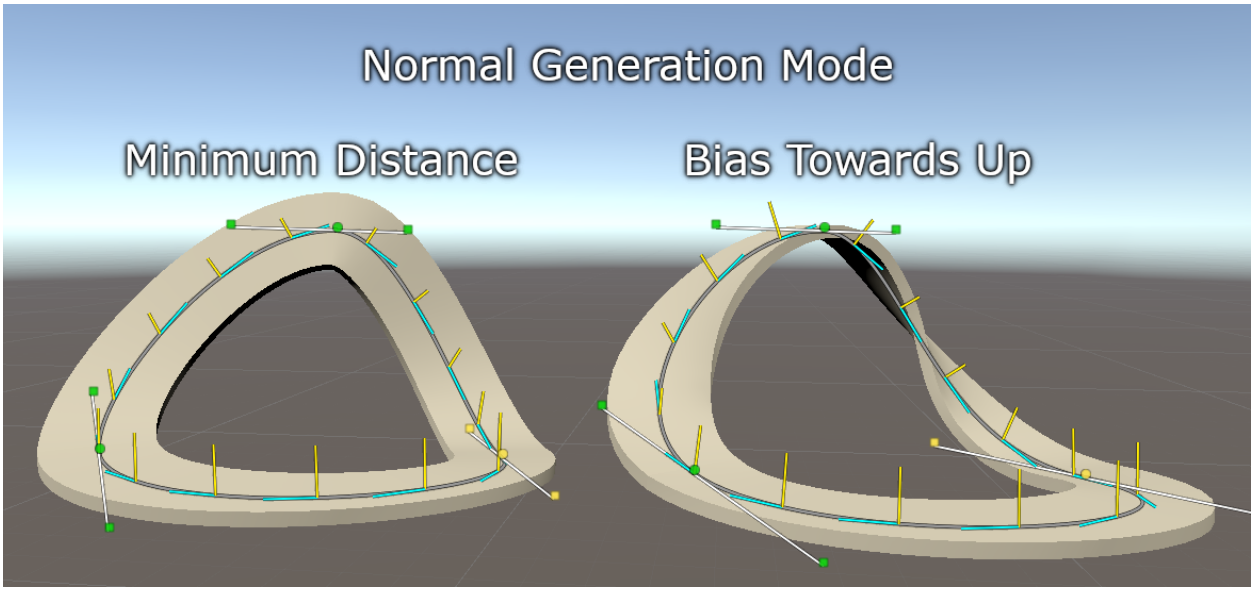
Flat Point Count controls the number of vertices in the crosswise direction for Flat Curves, as well as for the flat surface of Cylinders.

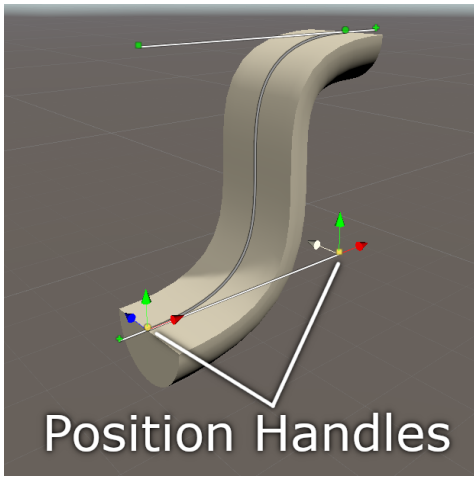


Lock To Position Zero allows you to lock all of the position control points in the curve to 0 in a particular axis. This is useful in case you'd like to keep your curve perfectly aligned in a single axis, such as when constructing a perfectly flat road.

Show Normals and **Show Tangents** show the normal and tangent directions of the curve at various distances, which appear in the Scene View as small yellow and cyan lines respectively.

Normal Generation Mode determines how the curve generates its normals. These normals run along the length of the curve itself and should not be confused with the normals of the mesh that will be generated. **Minimum Distance** attempts to minimize the amount of twisting in the curve, which works well for most use cases, but this can lead to normals that twist away from a particular axis. This can be fixed by using **Bias Towards Up/Forward/Right** which attempt to point all normals as close to their respective axes as possible. This can produce very extreme twisting in certain situations, particularly if the curve's slope starts to align with the axis.





Show Position Handles allows you to toggle whether or not position handles appear when editing position points.

Show Point Selection Window allows you to toggle whether or not the point selection window appears in the top left corner when a point is selected. The point selection window lets you edit the properties of an individual curve point.

Place Locked Points controls whether or not position points will have their tangents locked by default when added. **Tangents Locked** forces the two tangents to have the same length and opposing directions.

Samples For Cursor Collision Check: To perform cursor collision checks with the curve, Curve Designer approximates the curve as a series of line segments, then calculates the cursor's distance to those. This generally works well, but can be imprecise in certain situations, leading to your cursor not being exactly where you would expect on the curve. This issue can be fixed by increasing this value, but be warned, this value can negatively affect your performance when editing the curve if set too high.

Samples Per Segment: Sampling from a bezier curve by distance requires an approximation where you break the curve into a series of line segments. This value controls the number of segments used. A higher value results in a more accurate approximation, but can negatively affect performance.

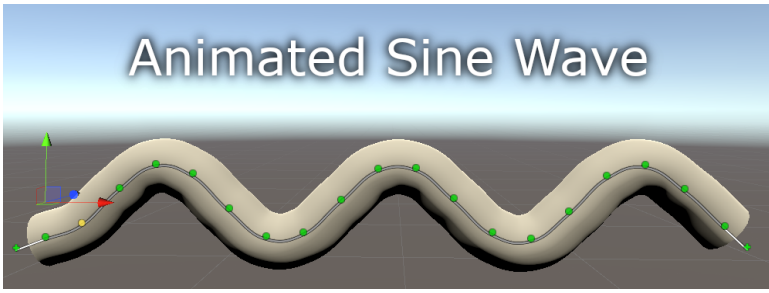
Scripting

The Curve3D class is split across two files, Curve3D.cs, which contains a simple public interface for reading data from curves, and Curve3DCore.cs, which contains the actual implementation of CurveDesigner. Unless you need to modify the curve at runtime, you should be able to accomplish most tasks with just the functions in Curve3D.cs.

```
//here is an example of how you could make an object follow along a curve
//and point in the direction tangent to the curve
using UnityEngine;
using ChaseMacMillan.CurveDesigner;
public class CurveFollower : MonoBehaviour
{
    public Curve3D curve; //Assign this in the inspector
    public float distanceAlongCurve = 0;
    public float speed = 1;
    public void Update()
    {
        distanceAlongCurve += Time.deltaTime * speed;
        PointOnCurve point = curve.GetPointAtDistanceAlongCurve(distanceAlongCurve);
        transform.position = point.position;
        transform.rotation = Quaternion.LookRotation(point.tangent,point.reference);
    }
}
```

However, if you do need to modify the properties of the curve at runtime, you'll need to dive a bit deeper.

The shape of the Curve3D is stored in **positionCurve**, which contains a List<PointGroup> **PointGroups**. A PointGroup is simply a point on the curve containing a **position, leftTangent** and **rightTangent.** Position is the position in the curve's local space. The left and right tangents are stored as offsets from the position. However if you use **SetPositionLocal** and **GetPositionLocal**, you can access the tangents as positions, rather than as offsets from the center position. You can get a point's distance from the start of the curve with **GetDistance.** After modifying the shape of the curve, either by manipulating PointGroups, or by toggling IsClosedLoop, you'll need to call **Recalculate**() to ensure the curve's data structures are up to date. And if you would then like the curve's mesh to update, you'll need to call **RequestMeshUpdate.**




```
//Here is a script which manipulates the shape of a curve at runtime
//to make the curve move in a sine wave
//if you add more points to the curve the sine wave will get longer
using UnityEngine;
using ChaseMacMillan.CurveDesigner;
public class SineWaver : MonoBehaviour
{
    public Curve3D curve; //Assign this in the inspector
    public float frequency = 1;
    public float amplitude = 1;
    public float speed = 1;
    public void Start()
    {
        //automatic tangents lets us avoid having to manually set the tangents
        curve.positionCurve.automaticTangents = true;
    }
    public void Update()
    {
        var pointGroups = curve.positionCurve.PointGroups;
        for (int i = 0; i < curve.positionCurve.PointGroups.Count; i++)
        {
            PointGroup point = pointGroups[i];
            float yOffset = Mathf.Sin((Time.time*speed+i)*frequency)*amplitude;
            Vector3 position = new Vector3(i,yOffset,0);
            point.SetPositionLocal(PointGroupIndex.Position,position);
        }
        //After modifying the curve's shape we need to recalculate to rebuild data structures
        curve.Recalculate();
        //Then we want the mesh of the curve to update
        curve.RequestMeshUpdate();
    }
}
```

As mentioned previously, a sampler is an object whose points are stored along the curve and whose values can be interpolated along the length of the curve. With the exception of **ExtrudeSampler**, all samplers inherit from **ValueSampler**. For example ColorSampler inherits from ValueSampler<Color>. You can use **GetValueAtDistance** to evaluate a ValueSampler at a particular distance.

If a sampler's **UseKeyframes** field is set to false, the sampler is in constant mode and GetValueAtDistance will return the contents of the **ConstValue** variable. However if UseKeyframes is set to true, GetValueAtDistance will interpolate between the values stored in **points**, which is a list of **SamplerPoints.** In the case of ColorSampler this would be a List<SamplerPoint<Color>>. You can set the **value** of a SamplerPoint, and you can get/set the distance of the SamplerPoint along the curve with **GetDistance** & **SetDistance.** After modifying a sampler make sure to call **Sort** to ensure the points are correctly ordered.


```
//This script will produce an animated rainbow along a curve
//Make sure you've assigned materials in the texture category that read from vertex color
//such as CurveDesigner/Art/Materials/DrawColorSurface
using UnityEngine;
using ChaseMacMillan.CurveDesigner;
public class AnimatedRainbow : MonoBehaviour {
    public Curve3D curve; //Assign this in the inspector
    public int colorPointCount = 24;
    public float length = 1;
    public float speed = 1;
    public void Start()
    {
        curve.colorSampler.UseKeyframes = true;//lets us interpolate a value along the curve
        curve.colorSampler.points.Clear();
        for (int i = 0; i < colorPointCount; i++)
        {
            float distance = i*curve.CurveLength/(colorPointCount-1);
            curve.colorSampler.InsertPointAtDistance(distance,curve.positionCurve);
        }
        //must be called whenever sampler points get reordered/added/removed
        curve.colorSampler.Sort(curve.positionCurve);
    }
    public void Update()
    {
        foreach (var i in curve.colorSampler.GetPoints(curve.positionCurve))
        {
            float distance = i.GetDistance(curve.positionCurve);
            float hue = (Time.time*speed+length*distance/curve.CurveLength)%1.0f;
            i.value = Color.HSVToRGB(hue,1,1);
        }
        //We haven't changed the shape of the curve so we don't need to call Recalculate()
        //but we do want the curve's mesh to update to display the new colors
        curve.RequestMeshUpdate();
    }
}
```

Misc

You can export your curve to a .obj file by right clicking the script in the inspector and selecting ExportToObj.

If you'd like to contribute code, report issues or get the latest version of the asset, check out my github https://github.com/cmacmillan/CurveDesigner or you can email me at support@chasemacmillan.com