Procedural indicators by cogobyte

Support Email: cogobyte@gmail.com

March 21, 2019

Contents

1 Introduction

Procedural indicators is a package for creating ingame 3D UI. For now it supports procedurally generated arrows and selection paths. With this package you can generate a wide variety of 3D arrows. Define the path of the arrow using a list of points or with animation curves and then you can change size, rotation, color, transparency and shape along that path. Procedural indicators package also supports arrows with broken path and arrows made of custom meshes (spheres, boxes, meshes). You can define arrow tips on both ends of the path using similar options that are used for the main path. Procedural selection indicators enable user to define a selection path around the object. The path definition is similar to arrow path (shape, color, broken path). Selection indicator paths can project to ground or any other object specified in the obstacle layer.

2 Getting Started

2.1 Arrow Indicators

Create an empty object. Drag and drop the ArrowObject script on it. ArrowObject script has properties for arrow path, two arrow tips (head and tail) and update options. Arrow object script must have an arrow path asset. Arrow tips are optional, if they are not set, only the main path will be drawn. Tips and path objects are asset objects that can be dragged and dropped to ArrowObject script. Either create your own assets using menu path:

Assets > Create > Cogobyte > ProceduralIndicators > Arrows

or use one of the assets provided in the examples.

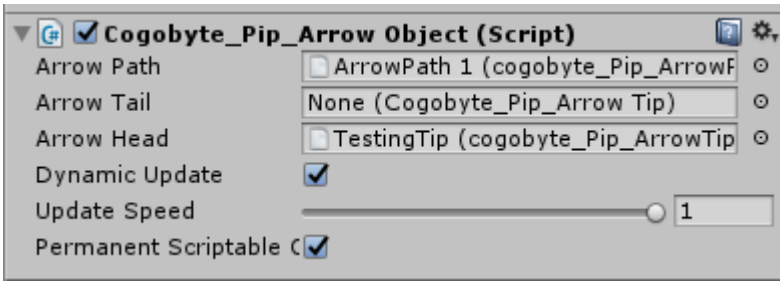

Figure 1: Arrow Object Options

Dynamic update parameter will recalculate the arrow every $1/updateSpeed$ frames. Turn this off if arrow parameters will not change over time.

ArrowObject Script will make a copy of paths and tips on play. If you want to edit these assets and view changes during play mode turn this parameter on and they will not be copied.

Flat shading is the new option that will double the number of vertices used to make sharper corners of mesh. Use this only on low poly primitives and low number of detail.

Run the game. Arrow will be drawn using start and end point of the arrow positions that are relative to the empty object where the script is on. EG. start point 0,0,0 will be at Game Object location. Start and end points are defined in Arrow Path asset object.

2.2 Selection Indicators

Create an empty object. Drag and drop the SelectionIndicator script on it. SelectionIndicator script has properties for path array, selection indicator path and update options. Path arrays and indicator paths are asset objects that can be dragged and dropped to SelectionIndicator script. Either create your own assets using menu path:
Assets » Create » Cogobyte » ProceduralIndicators » Selection
or use one of the assets provided in the examples. Update parameters are the same as arrow object options. Path arrays can be created from the
Assets » Create » Cogobyte » ProceduralLibrary » PathArrays

2.3 Essentials

If you only need the essential classes from the package, then include only the following folders:
cogobyte » cogobyte_ProceduralIndicators » Code, cogobyte » cogobyte_ProceduralIndicators » Materials, and cogobyte » cogobyte_ProceduralLibrary » Code.

3 Creating a custom path

Arrow is defined by an arrow path and two arrow tips from each side. Arrow path has four different modes: None, Extrude, Made Of Shapes and Broken Extrude. Extrude and broken extrude create the path by extruding shapes along a defined path. While extruding, shapes can change size, rotation, shape and offset their position from the defined path. Broken extrude behaves same as extrude with additional options to make spaces along extruded path to create a broken arrow.

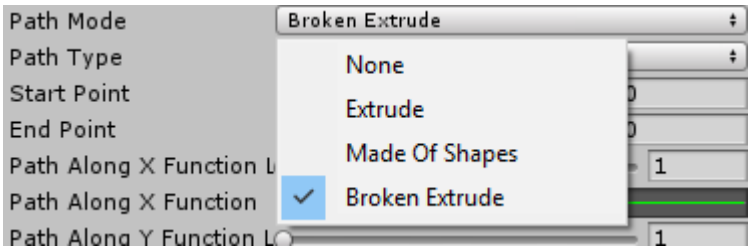

Figure 2: Path Modes

Made of shapes will take a list of defined meshes and copy them along path. None option will draw only the tips at each side of the arrow without the path. With None option path will still need to be defined in order to calculate the tip positions and rotations.

3.1 Extrude path options

To create an arrow path go to
Assets > Create > Cogobyte > ProceduralIndicators > Arrows > ArrowPath

There are three ways to define the path: using a point array, using animation curves or using path array object. Use the Path Type option to choose one of these. Each path type has a different subset of options for path trajectory.

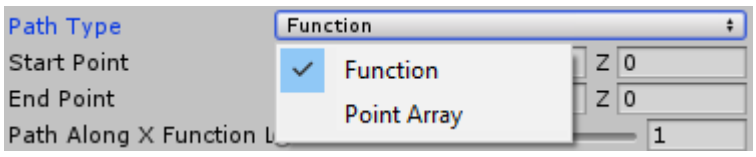

Figure 3: Path Types

Point array requires the Path Points option, which is an array of Vector3 points. Arrow will start at first point and go straight through all the points. Orientation of the arrow will follow the path and rotate accordingly. Alternatively you can use Path Array object. Explanation on path array object is in Procedural Library documentation.

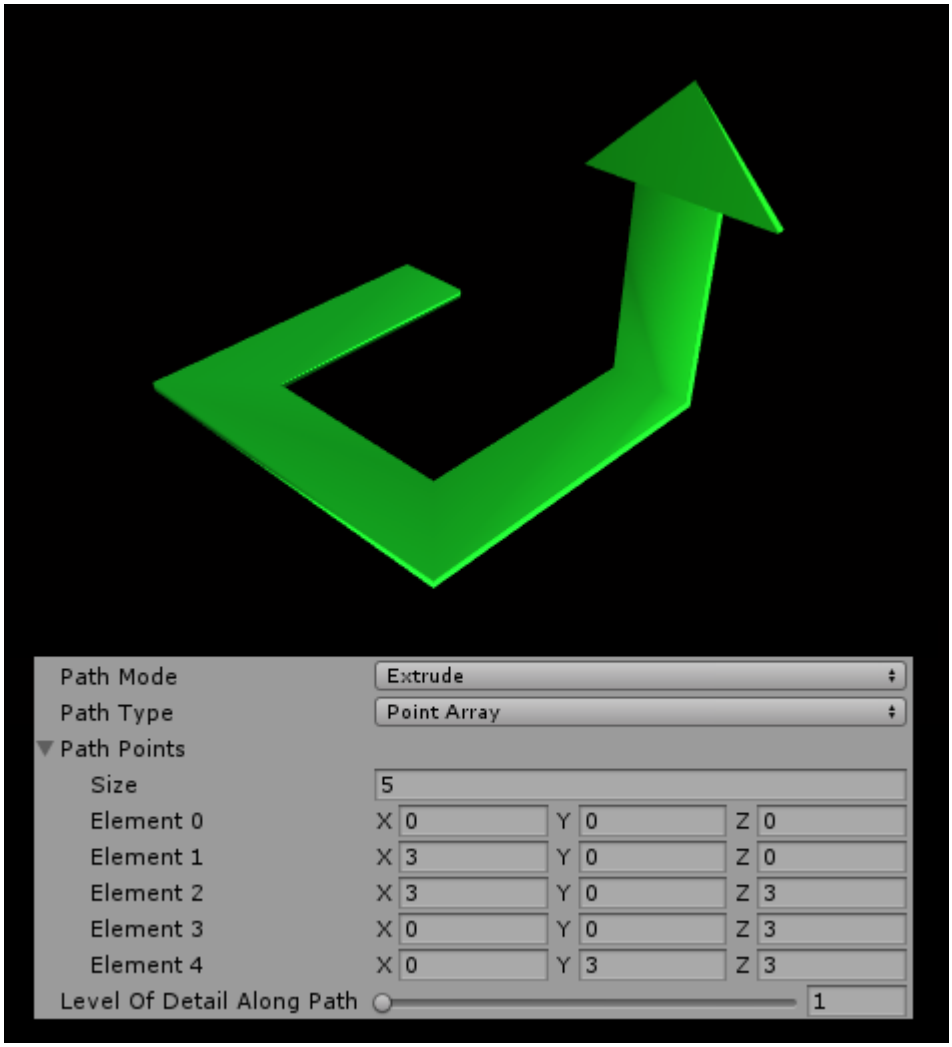

Figure 4: Path Points

For function option, you need to define a start and end point of the path and then use each animation curve to specify offsets for x, y and z axes. This

offset will move the path from the default straight line from start point to end point. Function length specifies the time range from 0 to function length that will be used in each animation curve. Since animation curves determine the offset at percentage of the path, if you set the length to the distance between start point and end point you can specify exactly at each path part where the offset is going to be.

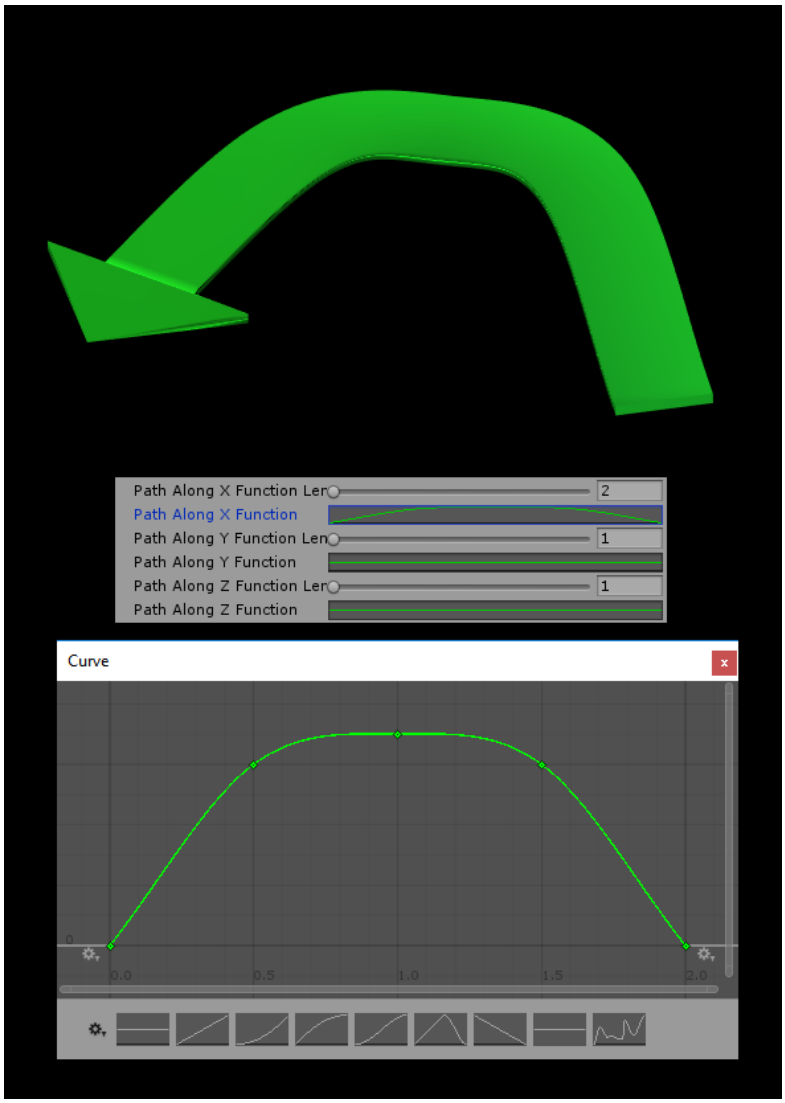

Figure 5: Path Modes

Level of detail will increase the rate at which vertices are generated. This option is used for both path array and function path type. For path array, level of detail will put a vertice on $1/levelOfDetail$ percentage between each array point. For functions it will place a vertice on $1/levelOfDetail$ percentage of path between start point and end point while also adding vertices for each path function key for each axis. For Figure 5 and level of detail 10 there would be 13 vertices in total, ten for level of detail and three for keys that are placed on time 0.5f, 1f and 1.5f of the animation function for x axis. For smooth curves specify many keys around the curve. Using high level of detail will generate smooth arrows, but it will have a performance impact. Arrows can hold a high level of detail (500 for a 4 square shape will have no framerate drop even if called each frame), but keep in mind that there could be a framerate drop for arrows that use shapes with high number of vertices and high level of detail if arrow is updated each frame.

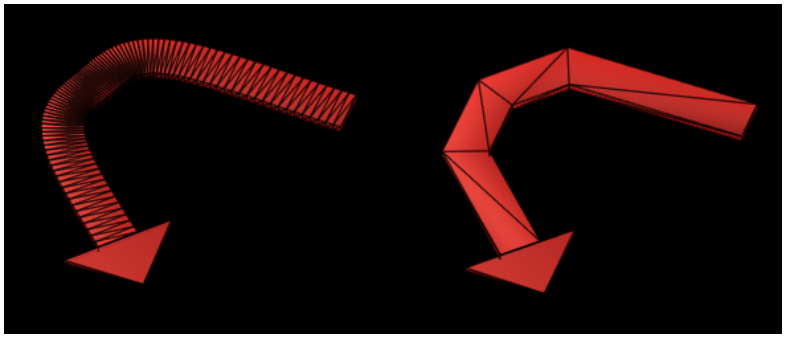

Figure 6: Different levels of detail 100 left and 1 right

Size along path is defined the same way as path function. Function length specifies the time range that will be used in width and height animation curves. Width defines the width along path and height defines height along path. Following figure changes the width of the path from 0.2f to 1f. These values determine the scale for width of the shape that is being extruded.

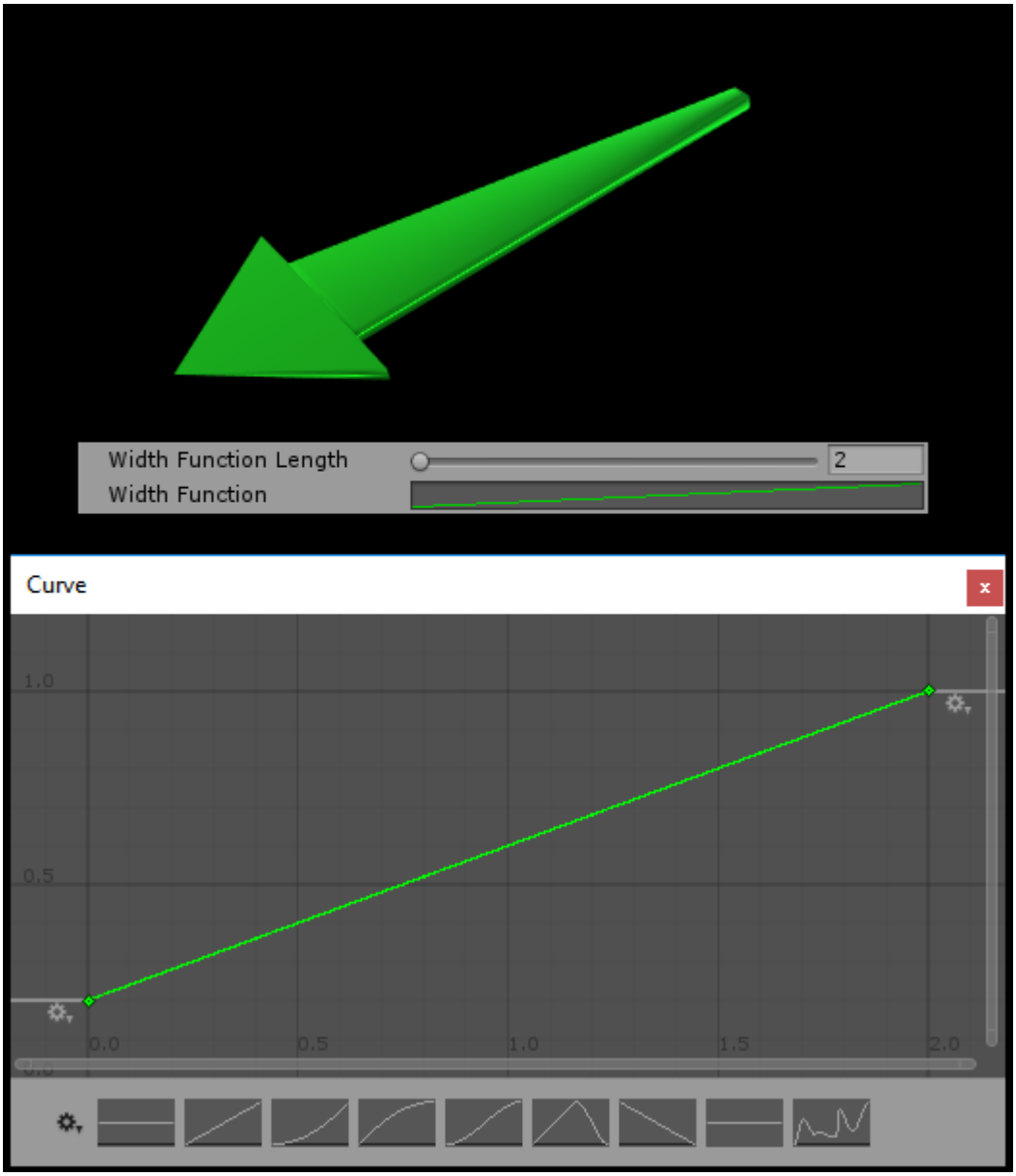

Figure 7: Width along path

Following figure shows the height along path with its corresponding function.

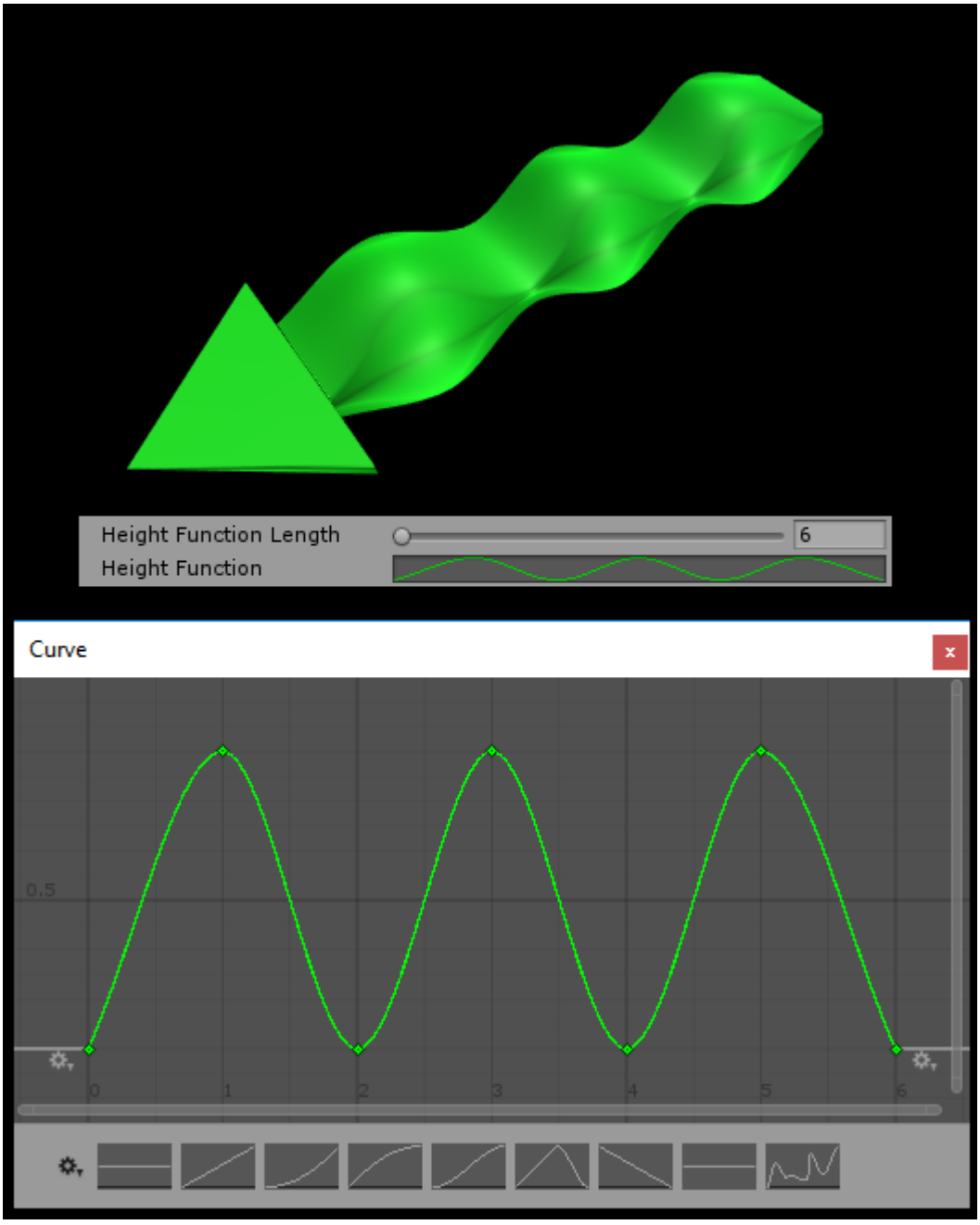

Figure 8: Height along path

Starting rotation of the arrow is defined using the up vector and arrow

path starting direction. Rotation function defines the percentage of 360 (from 0 to 1) shape rotation around current path direction starting with up vector. Zero value will keep the up vector intact. Value 0.5f will rotate the up vector 180 degrees around the arrow path direction. Rising linear rotation function spins the shape while extruding as shown in the Figure.

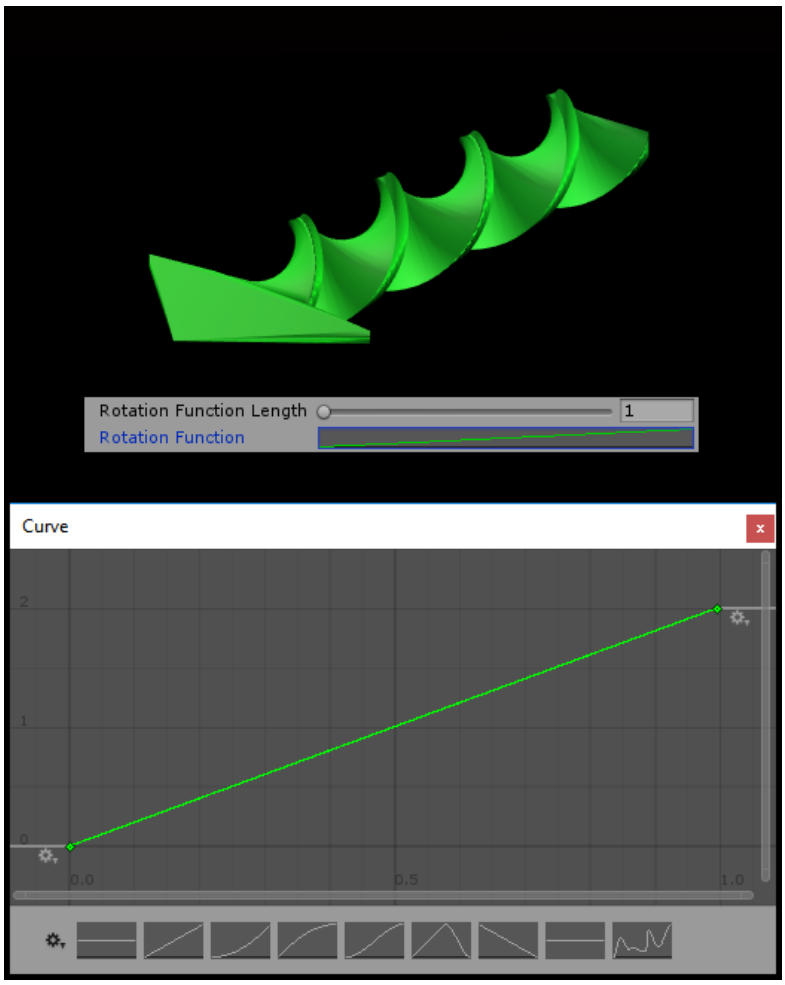

Figure 9: Rotation function

Procedural Arrow generator can generate vertex colors for shaders that use vertex colors. It will also generate uv maps for textures. Vertex colors can be taken from shape itself or defined by color gradient along path. If there are multiple gradients, extruded arrow shape will be split evenly for each gradient. It also supports transparency using alpha colors. Use shape colors option will use the vertex colors of the shape that is being extruded instead of the gradient.

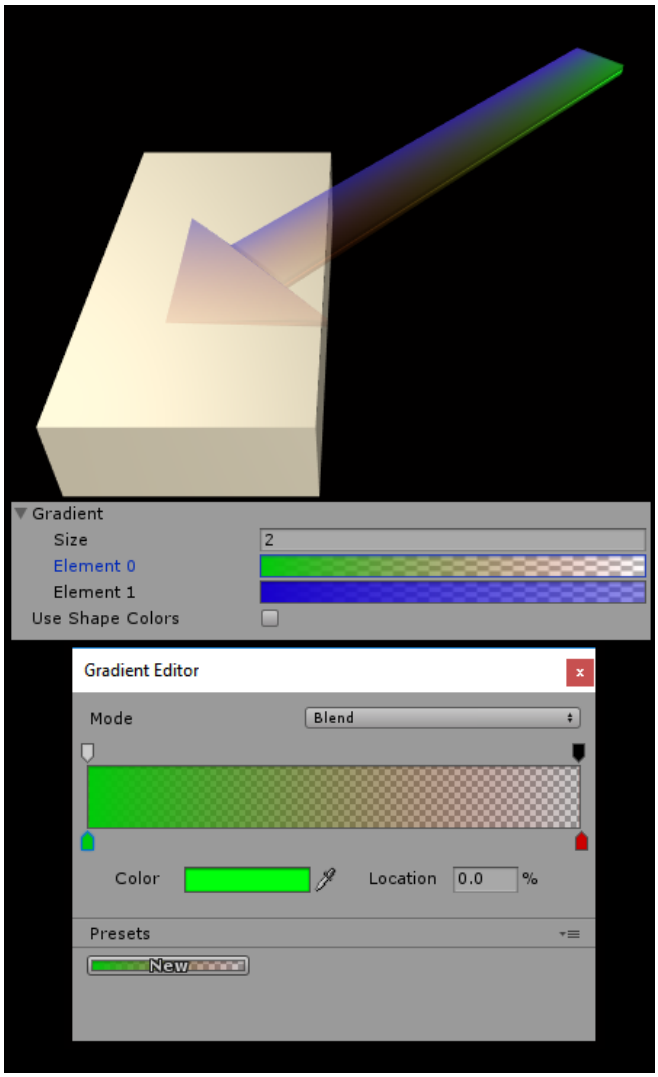

Figure 10: Gradient color of the arrow

Shape along path function choses from a list of predefined shapes that will be used for extrusion for each path point. This means that arrow can start as circle and end as square. Shape along path animation curve will round to the lowest integer value. To use this function, you need to define the primitive list of shapes that will be used. Create a Primitive List using Assets > Create > Cogobyte > ProceduralLibrary > Primitives > PrimitiveList. Add primitives that will be used and then set the Template Primitives option to the created asset.

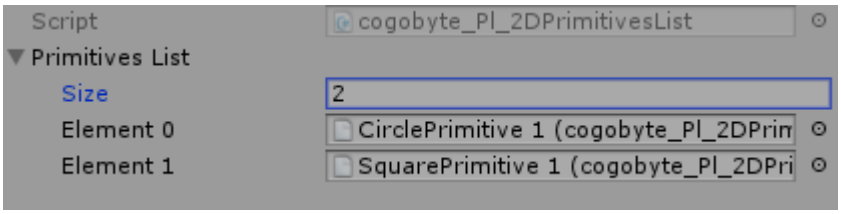

Figure 11: Primitive List

Shape function works the same way as path and scale, only it always rounds to the lowest integer.

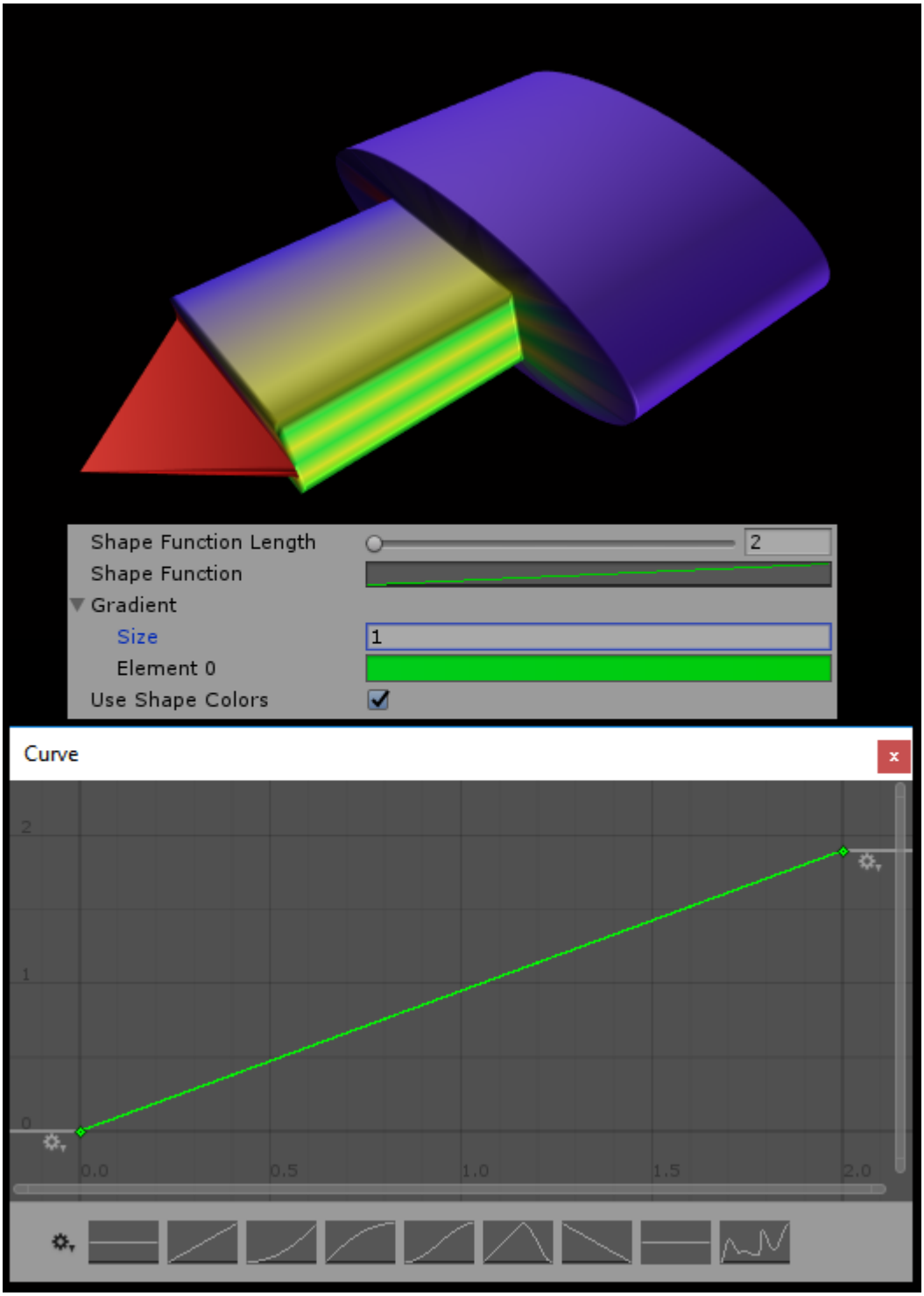

Figure 12: Shapes along path

3.2 Broken extrude options

Broken extrude has two additional parameters: line length for each line part and brake length for space between two lines.

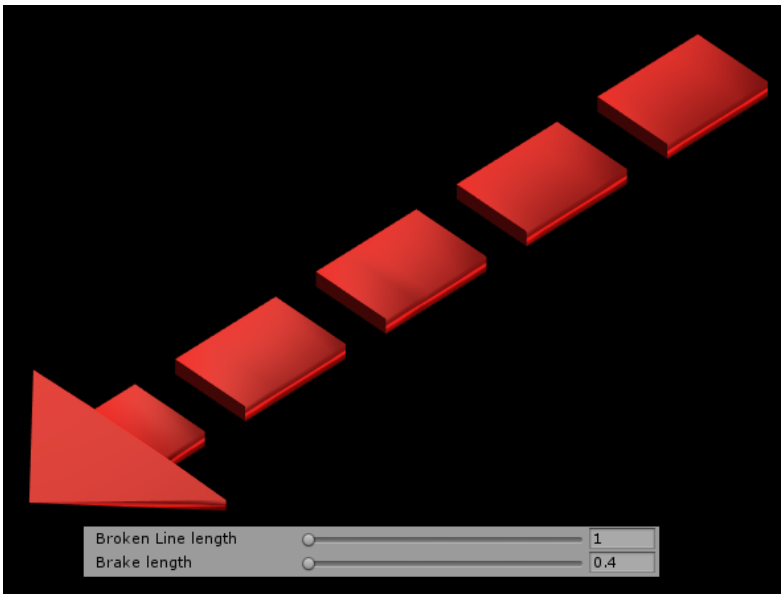

Figure 13: Broken extrude

3.3 Made of Shape options

Made of shapes mode creates instances of meshes along path and generates arrow tail and arrow head tips. It uses meshes from a predefined list similar to the primitive list for shapes. To create the list asset use the Assets » Create » Cogobyte » ProceduralLibrary » 3DMeshes » MeshesList menu. Add the meshes you want to use and add it to Custom Shape option. You also need to specify the distance between each shape from the meshes list.

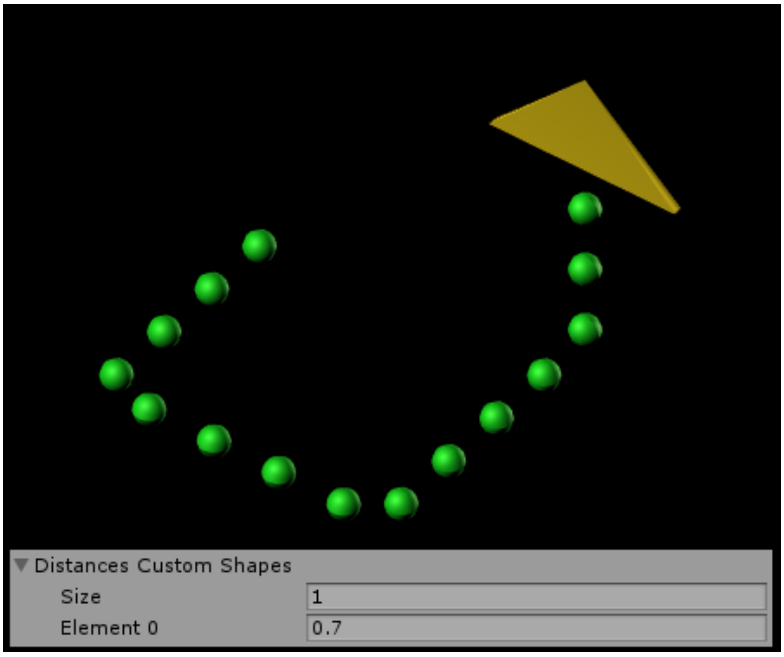

Figure 14: Made of Shapes Path Type

4 Creating a custom tip

Arrow tip has similar options to the path. It has an additional Size option that scales the arrow tip. Y is the length of the tip (it will take up space from the main path), X the width of the tip and Z is the height of the tip. Path options are a bit different since points can't be specified. End point will be the last point of the path. Start point will be somewhere on the arrow path with its distance equal to Size Y option. Arrow tip has three modes: None, Extrude and Mesh. Mode None will not include arrow tip in main arrow. Instead, the arrow path will be longer and take its space. Mode Extrude is similar to Extrude option of the arrow path. Mode Mesh will render a custom mesh as the arrow tip. For custom mesh the size parameter will be $size * boundingboxofamesh$. Path type can be Function or Follow main path. Function acts the same as the arrow path function type. Follow main path will use the calculated path of the arrow path instead.

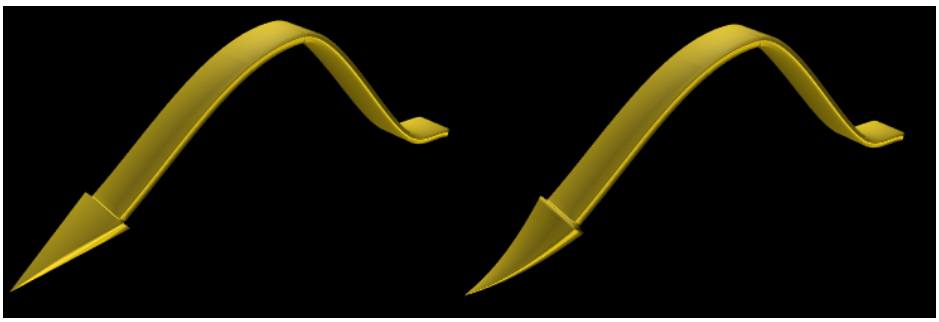

Figure 15: Path Type Function(left) and Follow Main path(right)

Level of detail, scale, rotate, and shape options are the same as the arrow path. Base rotation is the rotation of the last path point.

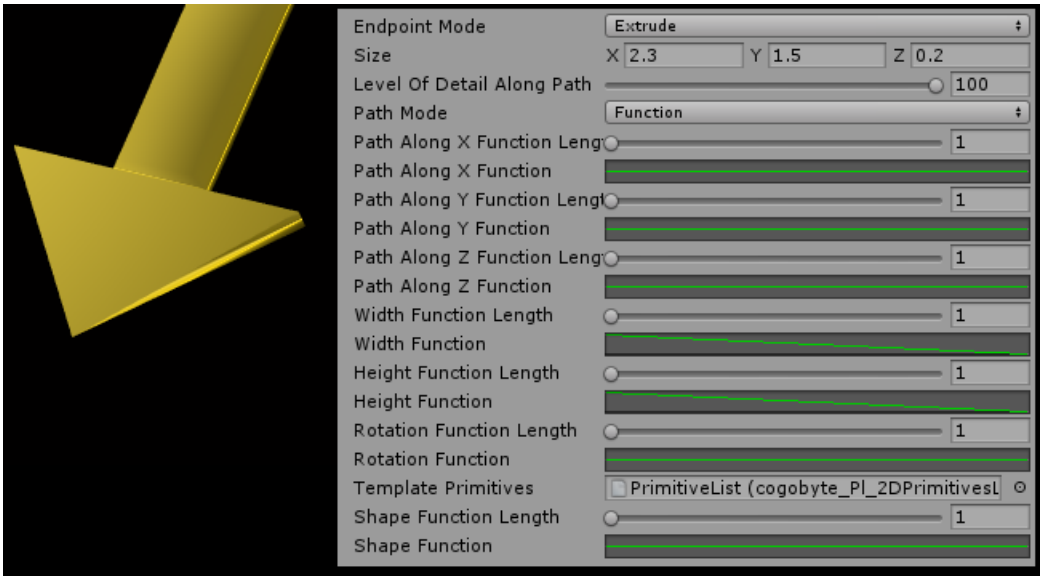

Figure 16: Custom tip

Custom mesh can use Sphere, WingedArrowMesh and CustomMesh. When choosing a custom mesh, keep in mind that the rotation of the mesh needs to be oriented in way that makes Y the tip direction(+), X the right(+) and left(-) side of the tip and Z front(+) and back(-) of the mesh.

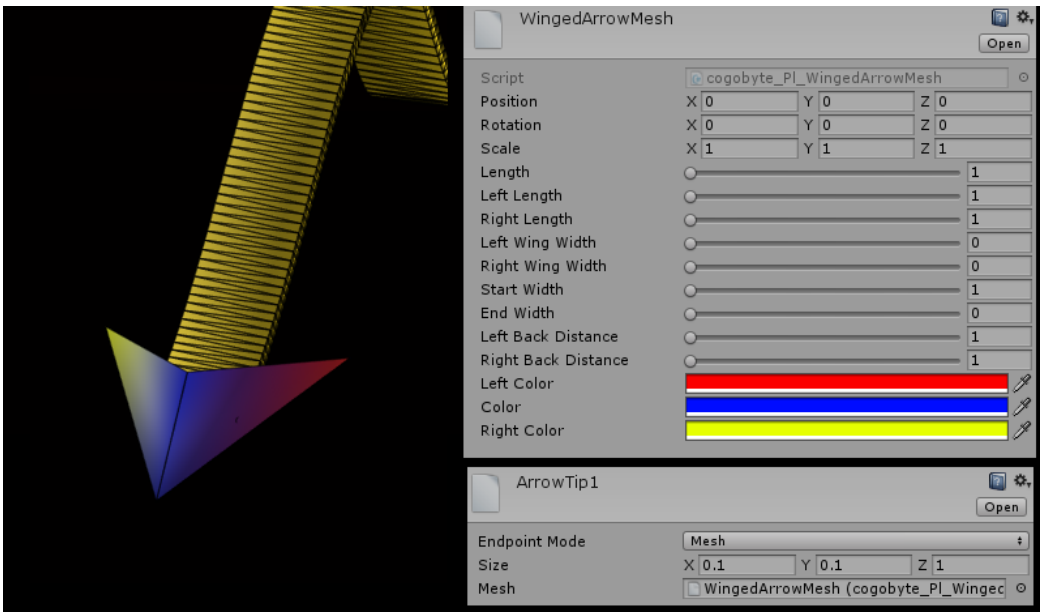

Figure 17: Using a custom mesh as arrow tip

5 Scripting and animating the arrow

Arrow options can also be set up using scripts. First create a script and add this member:

```
public ArrowObject arrowObject;
```

Create an empty GameObject. Add the ArrowObject script to it and set the arrow path and tails. Set the arrow path and tail options the usual way for options that are easier to set over inspector. Add your script to the same object that has the ArrowObject script in it. Set the arrowObject atribute to the created ArrowObject script. Set the path parameters of arrow using arrowObject.arrowPath, tail parameters using arrowObject.arrowTail and arrow head parameters using arrowObject.arrowPath. After setting the parameters call the update function using arrowObject.updateArrowMesh();

If arrow parameters will not change over time, call arrow generate function only once to save performance. If you want to animate the arrow, change these parameters in update function and always call updateArrowMesh(). Of course, always call updateArrowMesh() only if there are parameter changes to save up performance.

Txt Version of examples is provided in the CodeExamples.txt file.

```
public class CircleArrowDemo : MonoBehaviour {
        //Reference to ArrowObject script
        public ArrowObject arrowObject;
        //points that will be calculated
        Vector3[] points = new Vector3[10];
        //Not an animated arrow, just a setup over code to
           calculate points
        void Start () {
        //creates an arrow of 10 points from (0,0,0) to
           (10,0,0)
             for(int i = 0; i < 10; i++)
             {
                     points[i] = new Vector3(i, 0, 0);
             }
        //changes the arrowPath options, new points for
           point path
             arrowObject.arrowPath.editedPath = new
                List<Vector3>(points);
```

```
                arrowObject.updateArrowMesh();
                //generates the arrow again and updates the
                    mesh
        }
}
```

5.1 Arrow Path Parameters

5.1.1 Arrow Tail

```
//Field
public ArrowTip arrowTail;
//Example - Creates a new Default Extrude Tip
arrowObject.updateArrowMesh();
arrowObject.arrowPath.arrowHead =
    ScriptableObject.CreateInstance<ArrowTip>();
arrowObject.arrowPath.arrowHead.arrowTipMode =
    ArrowTip.ArrowTipMode.Extrude;
```

5.1.2 Arrow Head

```
//Field
public ArrowTip arrowHead;
arrowObject.updateArrowMesh();
//Example - Creates a new Default extrude Tip
arrowObject.arrowPath.arrowTail =
    ScriptableObject.CreateInstance<ArrowTip>();
arrowObject.arrowPath.arrowTail.arrowTipMode =
    ArrowTip.ArrowTipMode.Extrude;
```

5.1.3 Path Mode

```
//Fields
public enum ArrowPathMode { None, Extrude, MadeOfShapes,
    BrokenExtrude }
public ArrowPathMode arrowPathMode;
//Example - Set Mode to Broken Extrude
arrowObject.arrowPath.arrowPathMode =
    ArrowPath.ArrowPathMode.BrokenExtrude;
```

```
arrowObject.updateArrowMesh();
```

5.1.4 Path Type

```
//Fields
public enum ArrowPathType { Function, PointArray }
public ArrowPathType arrowPathType;
//Example - Set the Path Type to Function
arrowObject.arrowPath.arrowPathType =
   ArrowPath.ArrowPathType.Function;
arrowObject.updateArrowMesh();
```

5.1.5 Function Start and End Point

```
//Fields
public Vector3 startPoint;
public Vector3 endPoint;
//Example set the arrow from (0,0,0) to (10,0,0)
arrowObject.arrowPath.arrowPathType =
   ArrowPath.ArrowPathType.Function;
arrowObject.arrowPath.startPoint = new Vector3(0,0,0);
arrowObject.arrowPath.endPoint = new Vector3(10,0,0);
arrowObject.updateArrowMesh();
```

5.1.6 Level Of Detail

```
//Field
public int levelOfDetailAlongPath = 1;
//Example - Set number of points to at least 100
arrowObject.arrowPath.levelOfDetailAlongPath = 100;
arrowObject.updateArrowMesh();
```

5.1.7 Point Array

```
//Field
public List<Vector3> editedPath;
//Example 1 - A Half circle path
Vector3[] points = new Vector3[1000];
for (int i = 0; i < 1000; i++)
{
```

```
        points[i] = new Vector3(5 *  Mathf.Sin(i/500f *
            1.8f * Mathf.PI), i/500, 5 * Mathf.Cos(i / 500f
            * 1.8f * Mathf.PI));
}
arrowObject.arrowPath.arrowPathType =
    ArrowPath.ArrowPathType.PointArray;
arrowObject.arrowPath.editedPath = new
    List<Vector3>(points);
arrowObject.updateArrowMesh();
//Example 2 - 5 point path (square path that goeas up at
    end)
arrowObject.arrowPath.arrowPathMode =
    ArrowPath.ArrowPathMode.Extrude;
arrowObject.arrowPath.arrowPathType =
    ArrowPath.ArrowPathType.PointArray;
arrowObject.arrowPath.levelOfDetailAlongPath = 1;
points = new Vector3[5];
points[0] = new Vector3(0,0,0);
points[1] = new Vector3(3, 0, 0);
points[2] = new Vector3(3, 0, 3);
points[3] = new Vector3(0, 0, 3);
points[4] = new Vector3(0, 3, 3);
arrowObject.arrowPath.editedPath = new
    List<Vector3>(points);
arrowObject.updateArrowMesh();
```

5.1.8 Function Mode, Path Along X Function, Path Along X Function Length

```
//Fields
public float pathAlongXFunctionLength = 1;
public AnimationCurve pathAlongXFunction =
    AnimationCurve.Linear(0, 0, 1, 0);
//Example - Set 3 KeyPoints on the path x function that has
    a timeline of 2
arrowObject.arrowPath.arrowPathMode =
    ArrowPath.ArrowPathMode.Extrude;
arrowObject.arrowPath.arrowPathType =
```

```
   ArrowPath.ArrowPathType.Function;
arrowObject.arrowPath.levelOfDetailAlongPath = 100;
arrowObject.arrowPath.pathAlongXFunctionLength = 2;
arrowObject.arrowPath.startPoint = new Vector3(0, 0, 0);
arrowObject.arrowPath.endPoint = new Vector3(0, 0, 5);
AnimationCurve curve = AnimationCurve.Linear(0, 0, 2, 0);
curve.AddKey(0.5f, 3);
curve.AddKey(1, 3.5f);
curve.AddKey(1.5f, 3);
arrowObject.arrowPath.pathAlongXFunction = curve;
arrowObject.updateArrowMesh();
```

5.1.9 Function Mode, Path Along Y Function, Path Along Y Function Length

```
//Fields
public float pathAlongYFunctionLength = 1;
public AnimationCurve pathAlongYFunction =
   AnimationCurve.Linear(0, 0, 1, 0);
//Example - Set one key for arc path. Keep in mind that if
   arrow direction is same as axis. This function will make
   the path go faster(+) or slower(-).
arrowObject.arrowPath.arrowPathType =
   ArrowPath.ArrowPathType.Function;
arrowObject.arrowPath.startPoint = new Vector3(0, 0, 0);
arrowObject.arrowPath.endPoint = new Vector3(10, 0, 0);
arrowObject.arrowPath.levelOfDetailAlongPath = 100;
arrowObject.arrowPath.pathAlongYFunctionLength = 2;
AnimationCurve curve = AnimationCurve.Linear(0, 0, 2, 0);
curve.AddKey(1, 5);
arrowObject.arrowPath.pathAlongYFunction = curve;
arrowObject.updateArrowMesh();
```

5.1.10 Function Mode, Path Along Z Function, Path Along Z Function Length

```
//Fields
public float pathAlongZFunctionLength = 1;
```

```
public AnimationCurve pathAlongZFunction =
   AnimationCurve.Linear(0, 0, 1, 0);
//Example - Combined y and z curves
arrowObject.arrowPath.arrowPathType =
   ArrowPath.ArrowPathType.Function;
arrowObject.arrowPath.startPoint = new Vector3(0, 0, 0);
arrowObject.arrowPath.endPoint = new Vector3(2, 0, 0);
arrowObject.arrowPath.levelOfDetailAlongPath = 100;
arrowObject.arrowPath.pathAlongZFunctionLength = 2;
AnimationCurve curve = AnimationCurve.Linear(0, 0, 2, 0);
curve.AddKey(1, 10);
arrowObject.arrowPath.pathAlongZFunction = curve;
AnimationCurve curveY = AnimationCurve.Linear(0, 0, 1, 0);
curveY.AddKey(0.2f, 5);
curveY.AddKey(0.5f, 5);
curveY.AddKey(0.2f, 5);
arrowObject.arrowPath.pathAlongYFunction = curveY;
arrowObject.updateArrowMesh();
```

5.1.11 Scale Width

```
//Fields
public float widthFunctionLength = 1;
public AnimationCurve widthFunction =
   AnimationCurve.Linear(0, 0.85f, 1, 0.85f);
//Example - Scale path width from narrow to wide
arrowObject.arrowPath.arrowPathMode =
   ArrowPath.ArrowPathMode.Extrude;
arrowObject.arrowPath.arrowPathType =
   ArrowPath.ArrowPathType.Function;
arrowObject.arrowPath.levelOfDetailAlongPath = 100;
arrowObject.arrowPath.pathAlongXFunctionLength = 2;
arrowObject.arrowPath.startPoint = new Vector3(0, 0, 0);
arrowObject.arrowPath.endPoint = new Vector3(0, 0, 5);
AnimationCurve curve = AnimationCurve.Linear(0f, 0.2f, 2,
   1);
arrowObject.arrowPath.widthFunctionLength = 2;
arrowObject.arrowPath.widthFunction = curve;
```

```
arrowObject.updateArrowMesh();
```

5.1.12 Scale Height

```
//Fields
public float heightFunctionLength = 1;
public AnimationCurve heightFunction =
   AnimationCurve.Linear(0, 0.1f, 1, 0.1f);
//Example - Scale path height to wave function
arrowObject.arrowPath.arrowPathMode =
   ArrowPath.ArrowPathMode.Extrude;
arrowObject.arrowPath.arrowPathType =
   ArrowPath.ArrowPathType.Function;
arrowObject.arrowPath.levelOfDetailAlongPath = 100;
arrowObject.arrowPath.pathAlongXFunctionLength = 2;
arrowObject.arrowPath.startPoint = new Vector3(0, 0, 0);
arrowObject.arrowPath.endPoint = new Vector3(0, 0, 5);
arrowObject.arrowPath.heightFunctionLength = 6;
AnimationCurve curve = AnimationCurve.Linear(0f, 0.2f, 6f,
   0.2f);
curve.AddKey(1, 0.8f);
curve.AddKey(2, 0.2f);
curve.AddKey(3, 0.8f);
curve.AddKey(4, 0.2f);
curve.AddKey(5, 0.8f);
arrowObject.arrowPath.heightFunction = curve;
arrowObject.updateArrowMesh();
```

5.1.13 Rotation Function

```
//Fields
public float rotationFunctionLength = 1;
public AnimationCurve rotateFunction =
   AnimationCurve.Linear(0, 0, 1, 0);
//Example - create a spiral arrow
arrowObject.arrowPath.rotationFunctionLength = 2;
AnimationCurve curve = AnimationCurve.Linear(0f, 0f, 2f,
   1f);
arrowObject.arrowPath.rotateFunction = curve;
```

```
arrowObject.updateArrowMesh();
```

5.1.14 Shape Function and Custom Shapes

```
//Fields
public float shapeFunctionLength = 1;
public PrimitivesList templatePrimitives;
public AnimationCurve shapeFunction =
   AnimationCurve.Linear(0, 0, 1, 0);
//Example - Create a new Primitive List and define an
   animation function
PrimitivesList list =
   ScriptableObject.CreateInstance<PrimitivesList>();
list.primitivesList = new List<Primitive>();
CirclePrimitive circle =
   ScriptableObject.CreateInstance<CirclePrimitive>();
circle.numberOfSections = 10;
PlanePrimitive square =
   ScriptableObject.CreateInstance<PlanePrimitive>();
list.primitivesList.Add(circle);
list.primitivesList.Add(square);
arrowObject.arrowPath.templatePrimitives = list;
arrowObject.arrowPath.shapeFunctionLength = 2;
AnimationCurve curve = AnimationCurve.Linear(0f, 0f, 2f,
   1.9f);
arrowObject.arrowPath.shapeFunction = curve;
arrowObject.updateArrowMesh();
```

5.1.15 Made Of Shapes Mode, Custom Shapes

```
//Fields
public MeshesList customShapes;
public List<float> distanceBetweenShapes = new
   List<float>();
//Example - Create a Custom Shape List and set distance
   between meshes to 0.7f
arrowObject.arrowPath.arrowPathMode =
   ArrowPath.ArrowPathMode.MadeOfShapes;
```

```
arrowObject.arrowPath.arrowPathType =
   ArrowPath.ArrowPathType.Function;
arrowObject.arrowPath.levelOfDetailAlongPath = 1;
MeshesList meshList =
   ScriptableObject.CreateInstance<MeshesList>();
SphereMesh sphere =
   ScriptableObject.CreateInstance<SphereMesh>();
sphere.radius = 0.3f;
sphere.numberOfSectionsMeridian = 10;
sphere.numberOfSectionsParallel = 10;
meshList.meshesList = new List<ProceduralMesh>();
meshList.meshesList.Add(sphere);
arrowObject.arrowPath.customShapes = meshList;
arrowObject.arrowPath.distanceBetweenShapes = new
   List<float>() { 0.7f };
arrowObject.updateArrowMesh();
```

5.1.16 Broken Extrude Mode

```
//Fields
public float brokenLineLength = 1;
public float brakeLength = 1;
//Example - create a broken arrow with each segment length
   of 1 with 0.3f space between segments
arrowObject.arrowPath.arrowPathType =
   ArrowPath.ArrowPathType.Function;
arrowObject.arrowPath.levelOfDetailAlongPath = 25;
arrowObject.arrowPath.arrowPathMode =
   ArrowPath.ArrowPathMode.BrokenExtrude;
arrowObject.arrowPath.startPoint = new Vector3(0, 0, 0);
arrowObject.arrowPath.endPoint = new Vector3(0, 0, 7);
arrowObject.arrowPath.brokenLineLength = 1;
arrowObject.arrowPath.brakeLength = 0.3f;
arrowObject.updateArrowMesh();
```

5.1.17 Arrow Color

```
//Fields
public Gradient[] colorFunctions;
```

```
public bool useShapeColors;
//Example - Set the red gradient with no transparency
arrowObject.arrowPath.useShapeColors = false;
Gradient[] gradient = new Gradient[1];
gradient[0] = new Gradient();
GradientColorKey[] gck;
GradientAlphaKey[] gak;
gck = new GradientColorKey[2];
gck[0].color = Color.red;
gck[0].time = 0.0F;
gck[1].color = Color.red;
gck[1].time = 1.0F;
gak = new GradientAlphaKey[2];
gak[0].alpha = 1.0F;
gak[0].time = 0.0F;
gak[1].alpha = 1.0F;
gak[1].time = 1.0F;
gradient[0].SetKeys(gck, gak);
arrowObject.arrowPath.colorFunctions = gradient;
arrowObject.updateArrowMesh();
```

5.2 Arrow Tail/Head Parameters

5.2.1 Tip Mode

```
//Fields
public enum ArrowPointerMode { None, Mesh, Extrude }
public ArrowPointerMode arrowTipMode;
//Example - Sets the mode to extrude
//To use arrowHead of arrowObject without the path, first
   set it to the new one
arrowObject.arrowPath.arrowHead =
   ScriptableObject.CreateInstance<ArrowTip>();
arrowObject.arrowHead = arrowObject.arrowPath.arrowHead;
arrowObject.arrowHead.arrowTipMode =
   ArrowTip.ArrowTipMode.Extrude;
arrowObject.updateArrowMesh();
```

5.2.2 Arrow Size

```
//Field
public Vector3 size = new Vector3(1, 1, 1);
//Example - Make tip 3 times wider, 2 times longer and 4
   times higher than default tip
arrowObject.arrowPath.arrowHead =
   ScriptableObject.CreateInstance<ArrowTip>();
arrowObject.arrowHead = arrowObject.arrowPath.arrowHead;
arrowObject.arrowHead.arrowTipMode =
   ArrowTip.ArrowTipMode.Extrude;
arrowObject.arrowHead.size = new Vector3(3, 2, 4);
arrowObject.updateArrowMesh();
```

5.2.3 Arrow Level Of Detail

```
//Field
public int levelOfDetailAlongPath = 1;
//Example - set the level of detail to 100;
arrowObject.arrowHead.arrowTipMode =
   ArrowTip.ArrowTipMode.Extrude;
arrowObject.arrowHead.levelOfDetailAlongPath = 100;
arrowObject.updateArrowMesh();
```

5.2.4 Tip Path Type

```
//Fields
public enum ArrowPointerPathType { Function, FollowMainPath
   }
public ArrowPointerPathType arrowTipPathMode;
//Example Set the arrow tip path mode to FolloWMainPath
arrowObject.arrowHead.arrowTipMode =
   ArrowTip.ArrowTipMode.Extrude;
arrowObject.arrowHead.arrowTipPathType =
   ArrowTip.ArrowTipPathType.FollowMainPath;
arrowObject.updateArrowMesh();
```

5.2.5 Arrow Tip Path Functions

```
//Fields
public float pathAlongXFunctionLength = 1;
```

```
public AnimationCurve pathAlongXFunction =
   AnimationCurve.Linear(0, 0, 1, 0);
public float pathAlongYFunctionLength = 1;
public AnimationCurve pathAlongYFunction =
   AnimationCurve.Linear(0, 0, 1, 0);
public float pathAlongZFunctionLength = 1;
public AnimationCurve pathAlongZFunction =
   AnimationCurve.Linear(0, 0, 1, 0);
//Example Hook path. Keep in mind that Y is the tip
   lenghth, w width and z height
//and that these offsets are calculated at tip default
   position (Y up, Z forward, X right)
arrowObject.arrowHead.arrowTipMode =
   ArrowTip.ArrowTipMode.Extrude;
arrowObject.arrowHead.arrowTipPathType =
   ArrowTip.ArrowTipPathType.Function;
arrowObject.arrowHead.pathAlongXFunctionLength = 2;
arrowObject.arrowHead.levelOfDetailAlongPath = 100;
AnimationCurve curve = AnimationCurve.Linear(0, 0, 2, 0);
curve.AddKey(0.5f, 0.5f);
arrowObject.arrowHead.pathAlongXFunction = curve;
AnimationCurve curveZ = AnimationCurve.Linear(0, 0, 1, 0);
curveZ.AddKey(0.5f, 1);
arrowObject.arrowHead.pathAlongYFunction = curveZ;
arrowObject.updateArrowMesh();
```

5.2.6 Arrow Tip Scale Functions

```
//Fields
public float widthFunctionLength = 1;
public AnimationCurve widthFunction =
   AnimationCurve.Linear(0, 1, 1, 0);
public float heightFunctionLength = 1;
public AnimationCurve heightFunction =
   AnimationCurve.Linear(0, 1, 1, 0);
//Example - Create a tip that doesnt change height
arrowObject.arrowHead.arrowTipMode =
   ArrowTip.ArrowTipMode.Extrude;
```

```
arrowObject.arrowHead.arrowTipPathType =
   ArrowTip.ArrowTipPathType.Function;
arrowObject.arrowHead.levelOfDetailAlongPath = 100;
arrowObject.arrowHead.heightFunction =
   AnimationCurve.Linear(0, 0.5f , 1, 0.5f);
arrowObject.updateArrowMesh();
```

5.2.7 Arrow Tip Rotation Function

```
//Fields
public float rotationFunctionLength = 1;
public AnimationCurve rotateFunction =
   AnimationCurve.Linear(0, 0, 1, 0);
//Example Create a spiral tip
arrowObject.arrowHead.arrowTipMode =
   ArrowTip.ArrowTipMode.Extrude;
arrowObject.arrowHead.arrowTipPathType =
   ArrowTip.ArrowTipPathType.Function;
arrowObject.arrowHead.levelOfDetailAlongPath = 100;
arrowObject.arrowHead.rotateFunction =
   AnimationCurve.Linear(0, 0 , 1, 1);
arrowObject.updateArrowMesh();
```

5.2.8 Arrow Shape Function

```
//Fields
public PrimitivesList templatePrimitives;
public float shapeFunctionLength = 1;
public AnimationCurve shapeFunction =
   AnimationCurve.Linear(0, 0, 1, 0);
//Example Create a tip that starts as circle and ends as
   square
arrowObject.arrowHead.arrowTipMode =
   ArrowTip.ArrowTipMode.Extrude;
arrowObject.arrowHead.arrowTipPathType =
   ArrowTip.ArrowTipPathType.Function;
arrowObject.arrowHead.levelOfDetailAlongPath = 100;
PrimitivesList list =
   ScriptableObject.CreateInstance<PrimitivesList>();
```

```
list.primitivesList = new List<Primitive>();
CirclePrimitive circle =
   ScriptableObject.CreateInstance<CirclePrimitive>();
circle.numberOfSections = 10;
PlanePrimitive square =
   ScriptableObject.CreateInstance<PlanePrimitive>();
list.primitivesList.Add(circle);
list.primitivesList.Add(square);
arrowObject.arrowHead.templatePrimitives = list;
arrowObject.arrowHead.shapeFunctionLength = 2;
AnimationCurve curve = AnimationCurve.Linear(0f, 0f, 2f,
   1.9f);
arrowObject.arrowHead.shapeFunction = curve;
arrowObject.updateArrowMesh();
```

5.2.9 Arrow Mesh Option

```
//Field
public 3DMesh mesh;
//Example - Create a winged arrow mesh custom mesh as tip
arrowObject.arrowHead.arrowTipMode =
   ArrowTip.ArrowTipMode.Mesh;
WingedArrowMesh wingedMesh =
   ScriptableObject.CreateInstance<WingedArrowMesh>();
arrowObject.arrowHead.mesh = wingedMesh;
arrowObject.updateArrowMesh();
```

6 Creating shapes and meshes

Extrude and broken extrude use Primitves (Shapes) that are extruded along path. Keep in mind that creating a high number of vertices will increase number of arrow object vertices by each extrude point. All 2D primitive shapes have their faces faced upwards, x axis is right(+) and left(-) orientation and z is front(+) and back(-). You can create more custom shapes by creating a custom script for shape generation. Warning: Shapes should be defined in a 2D plane (all y coordinates of vertices should be zero). 2D shapes with various y positions will still work, but only for straight arrows. To create a custom shape, create a script that extends cogobytePl2DPrimitive class. Implement the getVertexCount() - number of vertices of your object, get-

TrisCount() number of triangles of your shape, Generate() - calculation of vertices, triangles, uvs and colors for object and updateOutline() - defines an outline in a counter clockwise order and calculates outlinemax length (length of the outline).

ArrowObject will generate an uv map for the mesh. Use this to apply textures. Arrow Generator will split uv map into four sections. For Made Od Shape shapes it will generate uv in upper left corner, for tail and head meshes it will generate uv map in the upper right and lower right corners respectively. For Extrude part it will generate uv map in lower left section. Uv map for Extrude is split into three parts, lower part from 0 to 0.05 is for arrow tail, midsection from 0.05 to 0.45 is for arrow path and upper part from 0.45 to 0.5 is for arrow head.

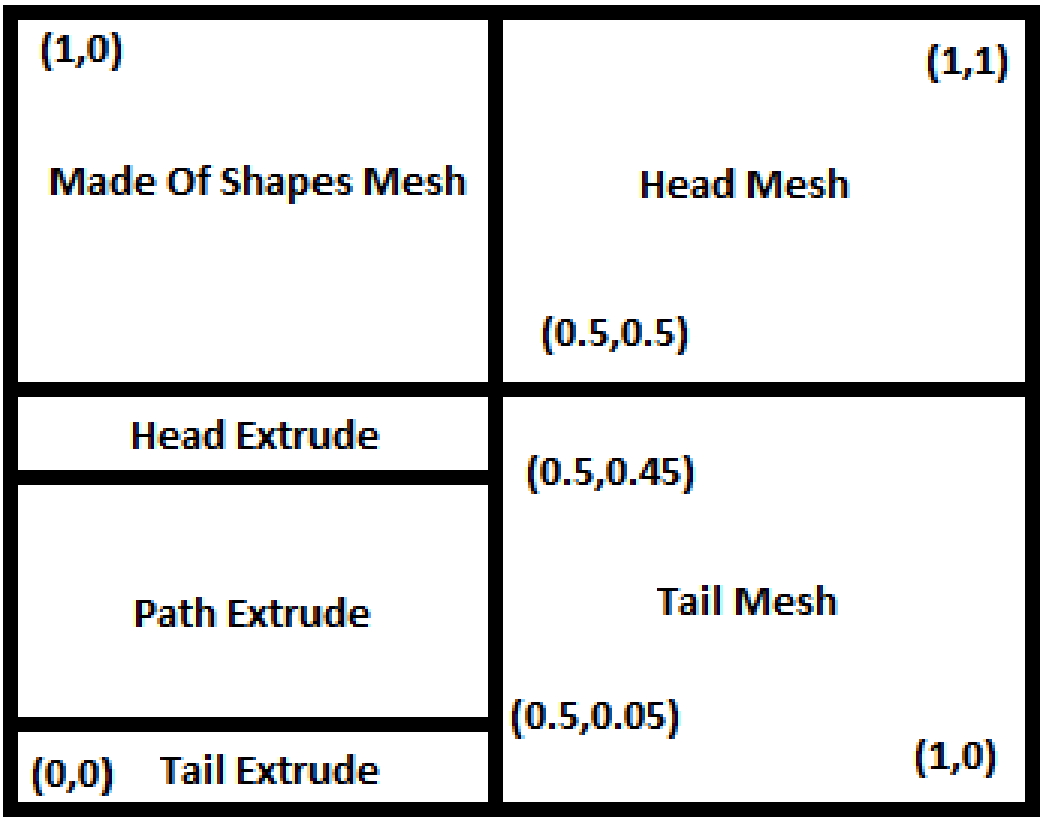

Figure 18: Uv map generation

7 Procedural selection indicators

Procedural selection indicators are used to draw a selection path around an object. Selection indicators use path arrays to specify the path and have options that define the shape of the path. There are three types of selection indicators: Normal, Broken and MadeOfShapes. They act similarly to the path of the arrow indicators.

7.1 Normal

Path of the selection is made of four sides. Selection has outer,inner,lower and upper functions that specify the distance of all sides from the center of the path. Color gradient can be specified for inner and outer half of the path. If is3D parameter is false only the upper side will be generated.

Figure 19: Normal Selection

7.2 Broken

Broken path has additional option for number of broken lines and percentage of empty space. Whole length of the path will be divided by the number of broken lines to create path parts. Percentage of empty space will also be divided by the number of broken lines and set between each part.

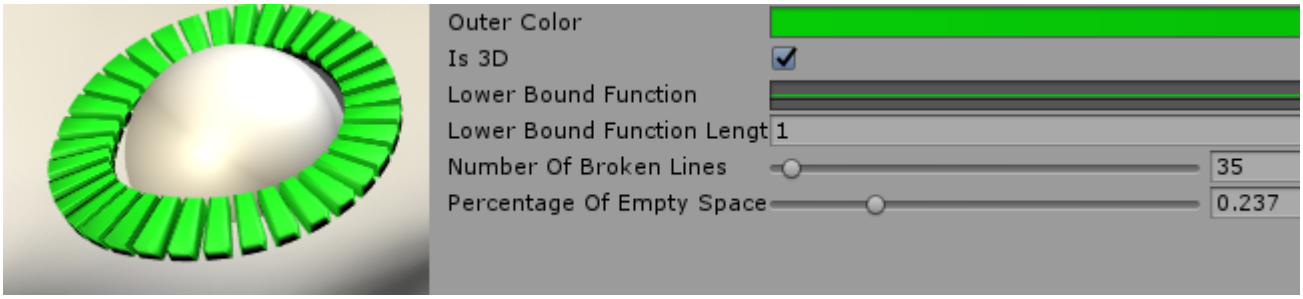

Figure 20: Broken Selection

7.3 Made of shapes

Made of shapes options include number of shapes and space taken by each shape. It uses the meshes list to specify all shapes that will be used. Color function will paint the meshes along path by gradient or use default mesh colors using the Use Shape Colors option.

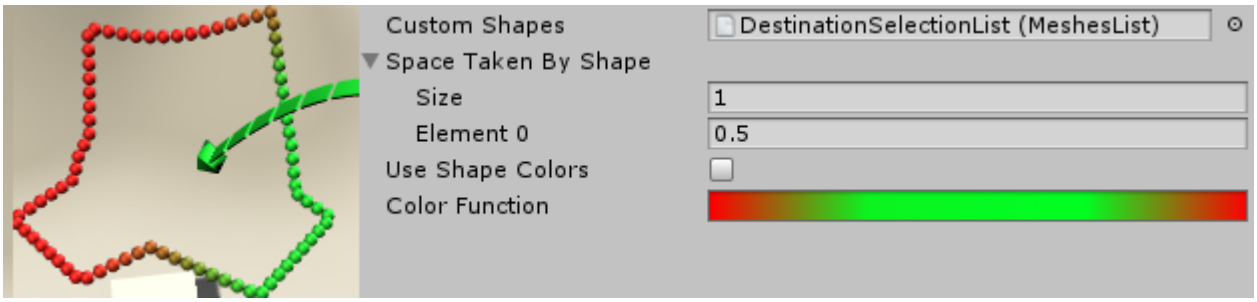

Figure 21: Made of shapes Selection

7.4 Scripting and animating selection indicators

```
public class TestPathsScript : MonoBehaviour {
public SelectionIndicator selectionIndicator;
// Use this for initialization
NormalSelectionIndicatorPath path;
CircularPathArray pathArray;
void Start () {
//Add a path array
```

```
pathArray =
   ScriptableObject.CreateInstance<CircularPathArray>();
//Level of detail must be high enough to make functions
   Effective
pathArray.levelOfDetail = 100;
pathArray.radius = 10f;
selectionIndicator.pathArray = pathArray;
//Add a path indicator Normal broken or madeofshapes
path =
   ScriptableObject.CreateInstance<NormalSelectionIndicatorPath>();
selectionIndicator.selectionIndicatorPath = path;
path.outerWidthFunction =  AnimationCurve.Linear(0, 0.2f,
   1, 0.85f);
path.innerWidthFunction = AnimationCurve.Linear(0, 0.5f, 1,
   5f);
path.innerWidthFunction = AnimationCurve.Linear(0, 0.5f, 1,
   5f);
//Always include a layer for obstacle check
pathArray.obstacleCheck =
   PathArray.ObstacleCheckMode.Parellel;
pathArray.obstacleLayer =
   LayerMask.NameToLayer("Everything");
}

void Update () {
selectionIndicator.updateIndicatorMesh();
//Change over time to animate
pathArray.translation = new Vector3(3 *
   Mathf.Sin(Time.fixedTime), 3, 0);
path.heightFunction = AnimationCurve.Linear(0, 0.5f, 1,
   6+5*Mathf.Sin(Time.fixedTime));
}
}
```

8 Procedural Grid Indicators

Procedural grid indicators are used to draw 3d grids. Grid indicator draws a 3D wire grid and Grid Mesh Indicator slices a given mesh by a 3d grid.

8.1 Grid Indicator

Grid Indicator draws a 3D wire grid. It is defined by three float arrays for each dimension. Each float array contains the sizes of each cell. Grid can be made of uneven sizes. Whole grid is colored by the grid color parameter. Grid is drawn from point (0,0,0) to the size of the grid. This can be offset using the grid offset vector. Thickness size determines the thickness of the grid wire.

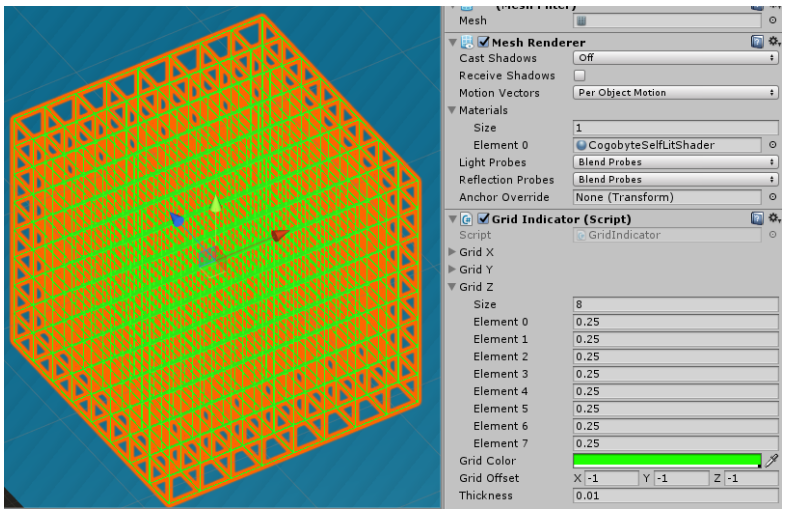

Figure 22: Grid Indicator

8.2 Grid Mesh Indicator

Grid Mesh indicator slices the given mesh into cells by a 3D grid. Grid cell size is determined by three float arrays. Grid can be made of uneven sizes. Indicator will create sliced triangles and assign them to each cell. It is then possible to change the colors of the triangles in each cell by calling the setColor(x,y,z,color) function. All triangles that are outside of the grid are assigned to a special outside cell. Color of these cells is determined by the outside color parameter. It is also possible to change the default position, rotation and scale of the input mesh. Grid is calculated from point (0,0,0) to the size of the grid. This can be offset using the grid offset vector.

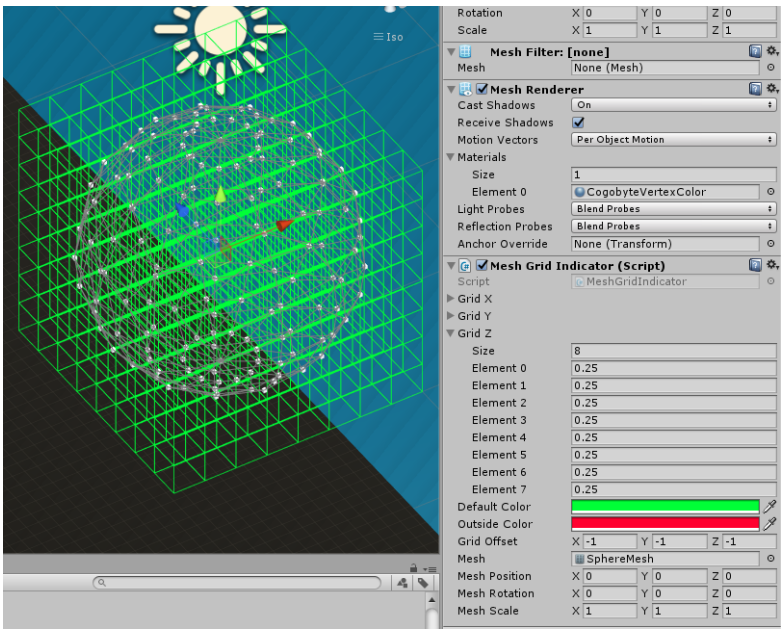

Figure 23: Mesh Grid Indicator