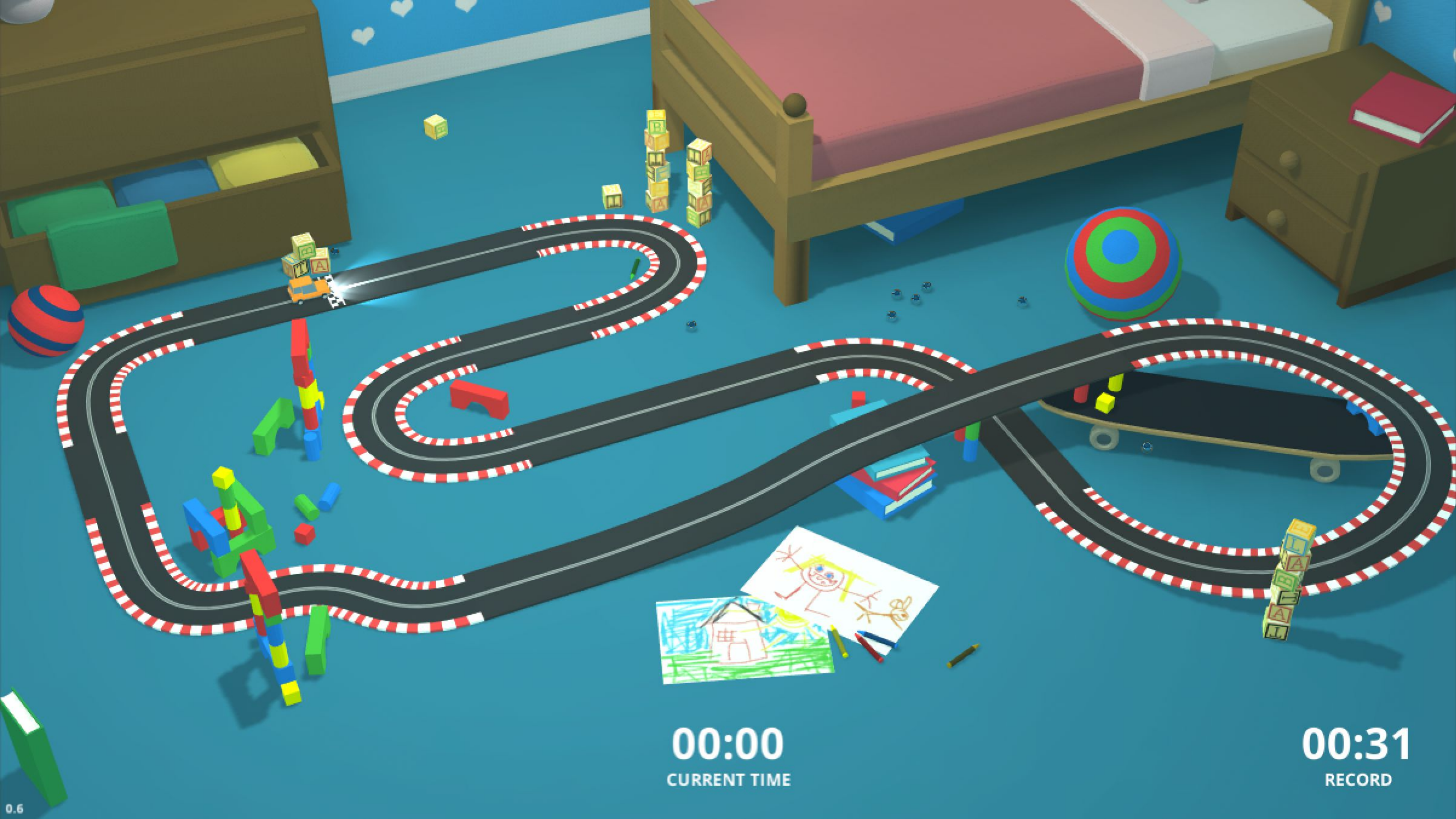
@ JoachimHolmér
Co-founder & indie developer at
Neat Corporation

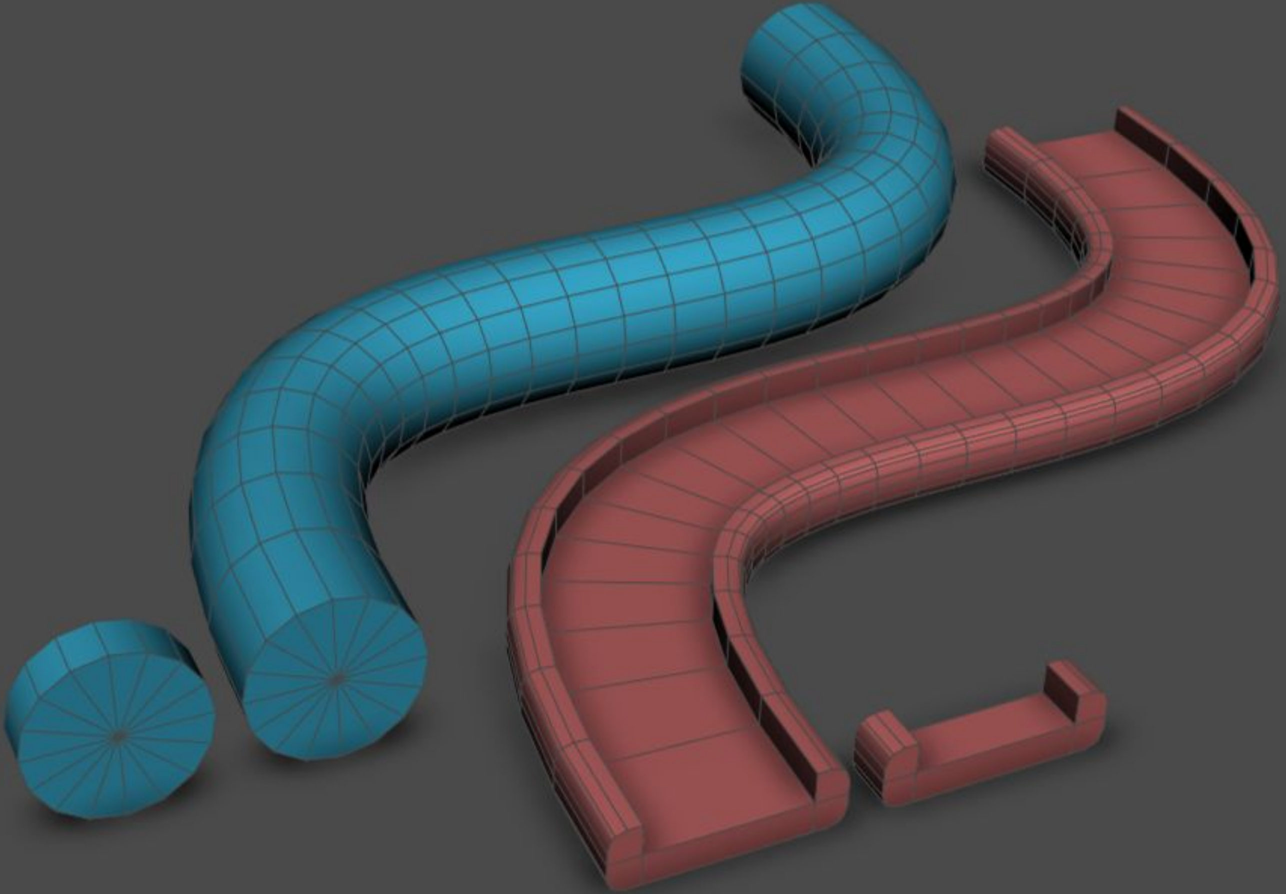
UNITE 2015 BOSTON

A coder's guide to spline-based procedural geometry

POP CORN
R1
LittleBIG Planet 3

74
38
111
115
115

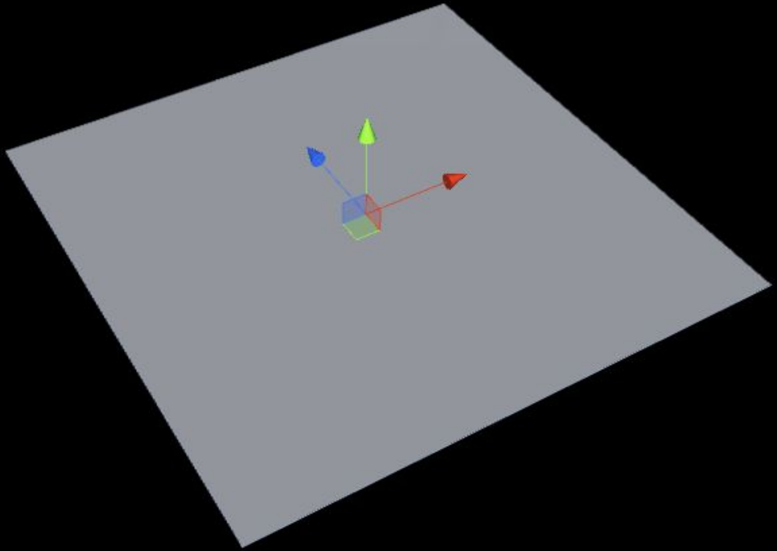

00:00
CURRENT TIME
00:31
RECORD
0.6

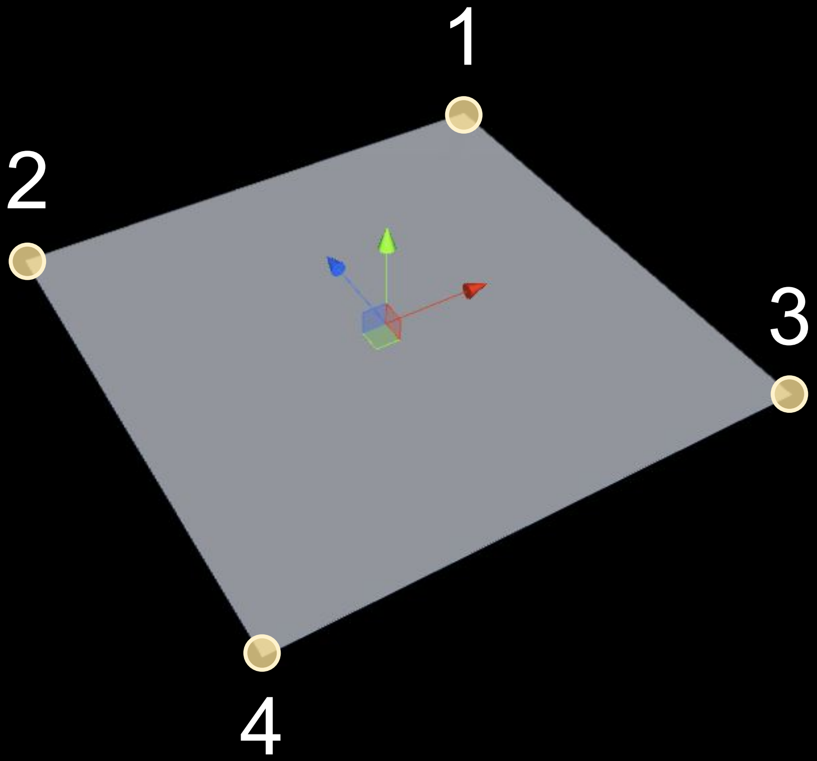


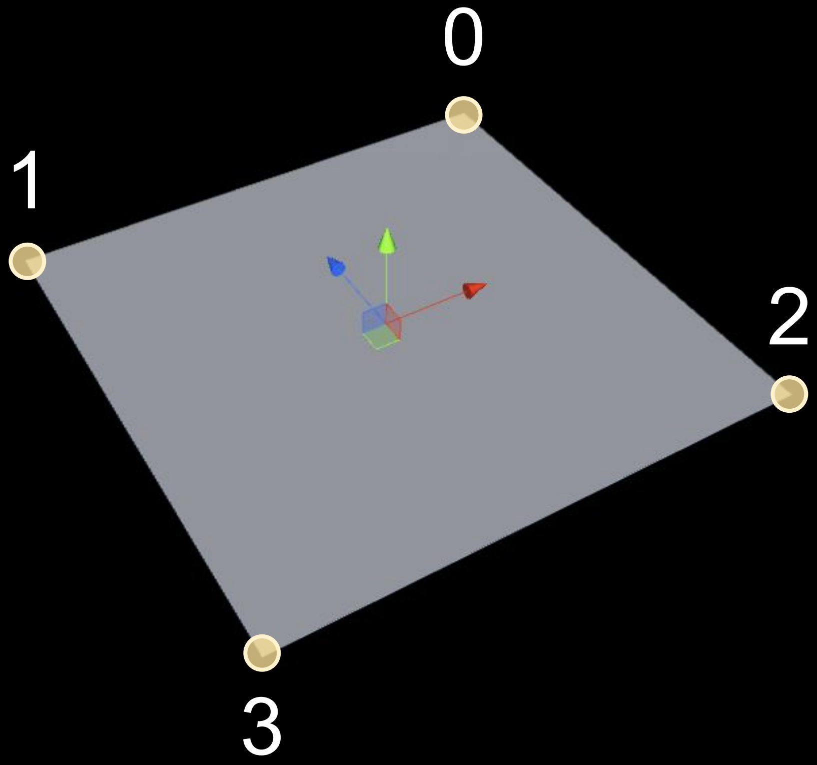
Before we get started…

What is a mesh?

- Vertices
- Triangles
- Normals
- UVs
- Vertex Colors
- Tangents
- Other

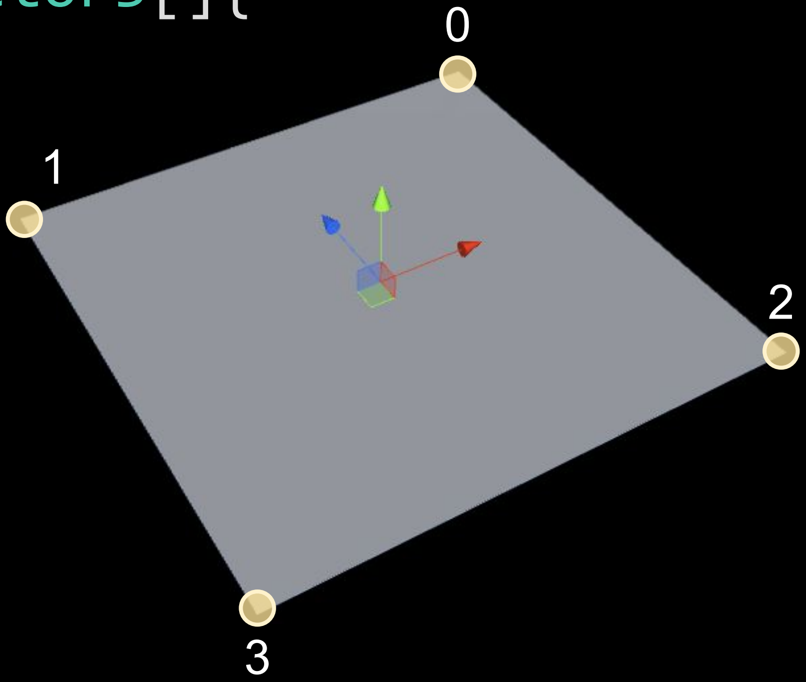



1
2
3
4


0
1
2
3

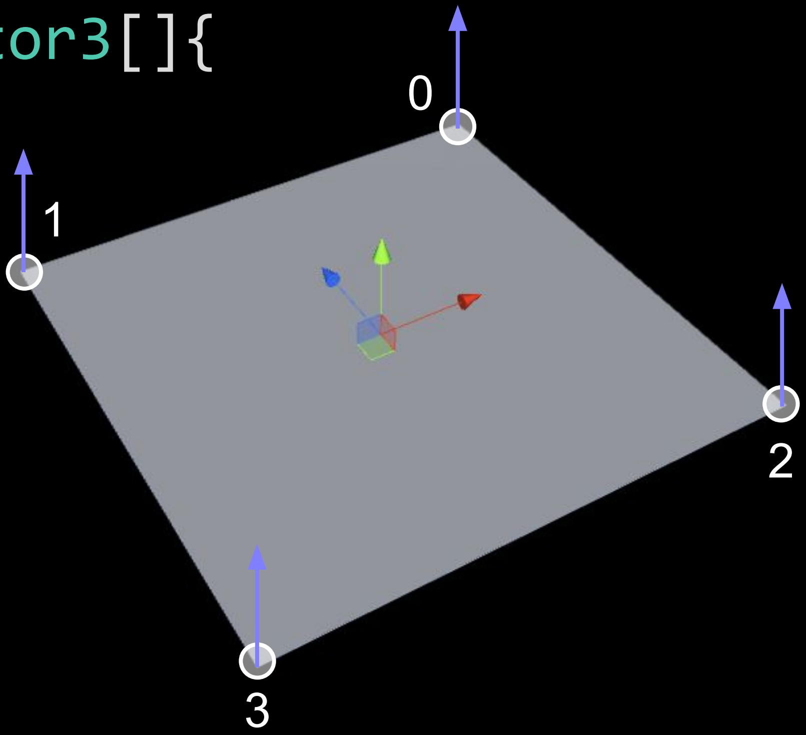
Vertices

```
Vector3[] vertices = new Vector3[]{
   new Vector3(  1, 0,  1 ),
   new Vector3( -1, 0,  1 ),
   new Vector3(  1, 0, -1 ),
   new Vector3( -1, 0, -1 )
};
```

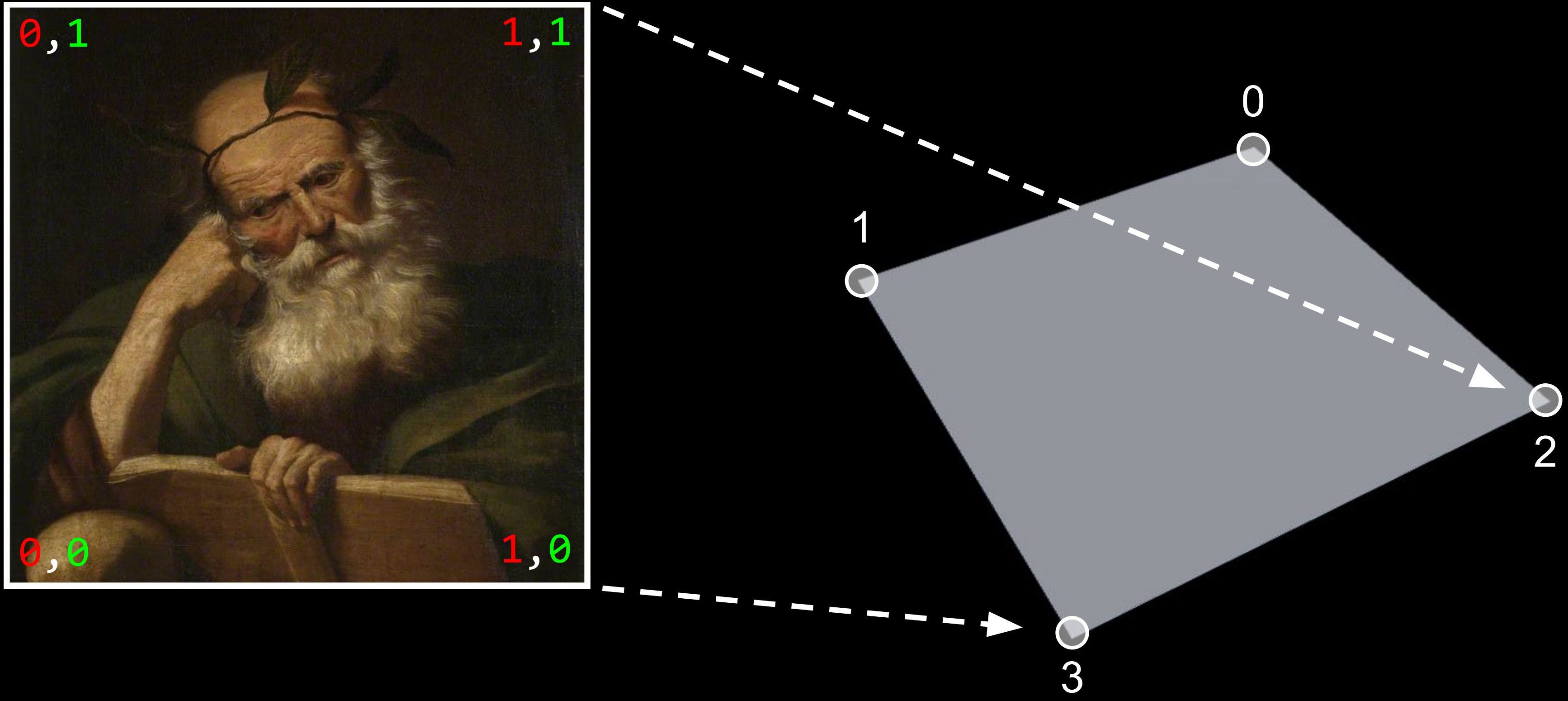


Normals

```
Vector3[] normals = new Vector3[]{
    new Vector3( 0, 1, 0 ),
    new Vector3( 0, 1, 0 ),
    new Vector3( 0, 1, 0 ),
    new Vector3( 0, 1, 0 )
};
```



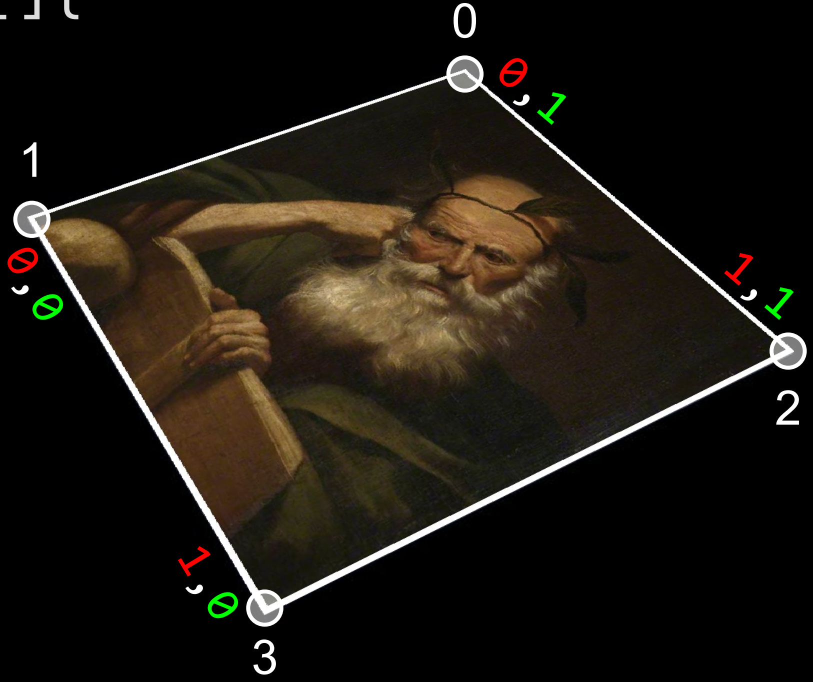
UVs



UVs

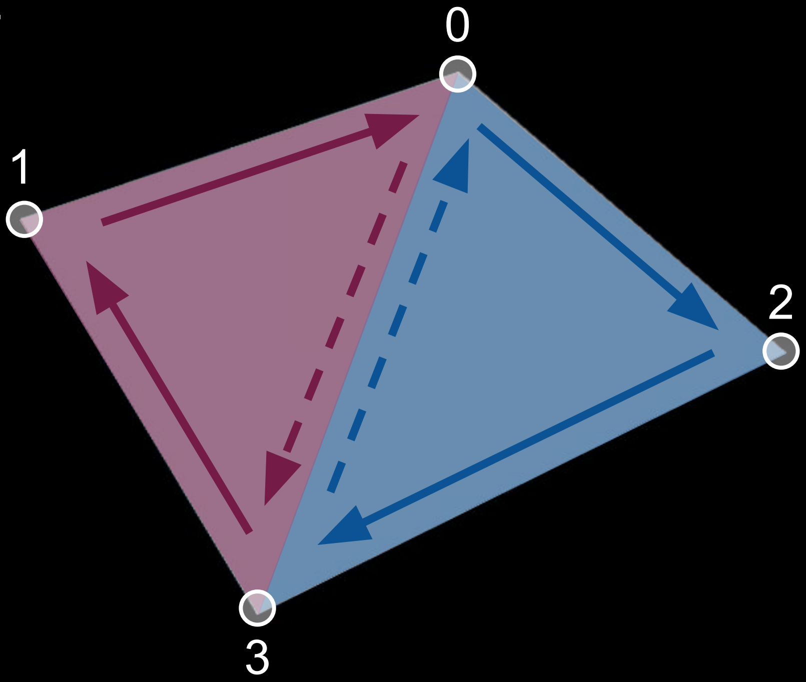
```
Vector2[] uvs = new Vector2[]{
    new Vector2( 0, 1 ),
    new Vector2( 0, 0 ),
    new Vector2( 1, 1 ),
    new Vector2( 1, 0 )
};
```

U V



Triangles

```
int[] triangles = new int[]{
    0, 2, 3,    First triangle
    3, 1, 0     Second triangle
};
```



Applying it to your objects

```
MeshFilter mf = GetComponent<MeshFilter>();
if( mf.sharedMesh == null )
     mf.sharedMesh = new Mesh();
Mesh mesh = mf.sharedMesh;


Vector3[] vertices = new Vector3[]{ … };
Vector3[] normals = new Vector3[]{ … };
Vector2[] uvs = new Vector2[]{ … };

mesh.Clear();
mesh.vertices = vertices;
mesh.normals = normals;
mesh.uv = uvs;
mesh.triangles = triangleIndices;
```

Applying it to your objects

- Make sure each object has their own unique mesh
- `GetComponent<MeshFilter>().sharedMesh = new Mesh();`
- Then access it using meshFilter.sharedMesh
- Or, use properties and inheritance to make this easier

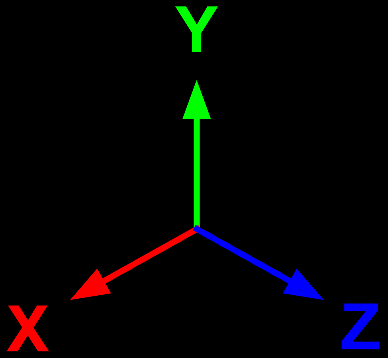
```
public class UniqueMesh : MonoBehaviour {
     [HideInInspector] int ownerID; // To ensure they have a unique mesh
     MeshFilter _mf;
     MeshFilter mf { // Tries to find a mesh filter, adds one if it doesn't exist yet
          get{
               _mf = _mf == null ? GetComponent<MeshFilter>() : _mf;
               _mf = _mf == null ? gameObject.AddComponent<MeshFilter>() : _mf;
               return _mf;
          }
     }
     Mesh _mesh;
     protected Mesh mesh { // The mesh to edit
          get{
               bool isOwner = ownerID == gameObject.GetInstanceID();
               if( mf.sharedMesh == null || !isOwner ){
                    mf.sharedMesh = _mesh = new Mesh();
                    ownerID = gameObject.GetInstanceID();
                    _mesh.name = "Mesh [" + ownerID + "]";
               }
               return _mesh;
          }
     }
}
```

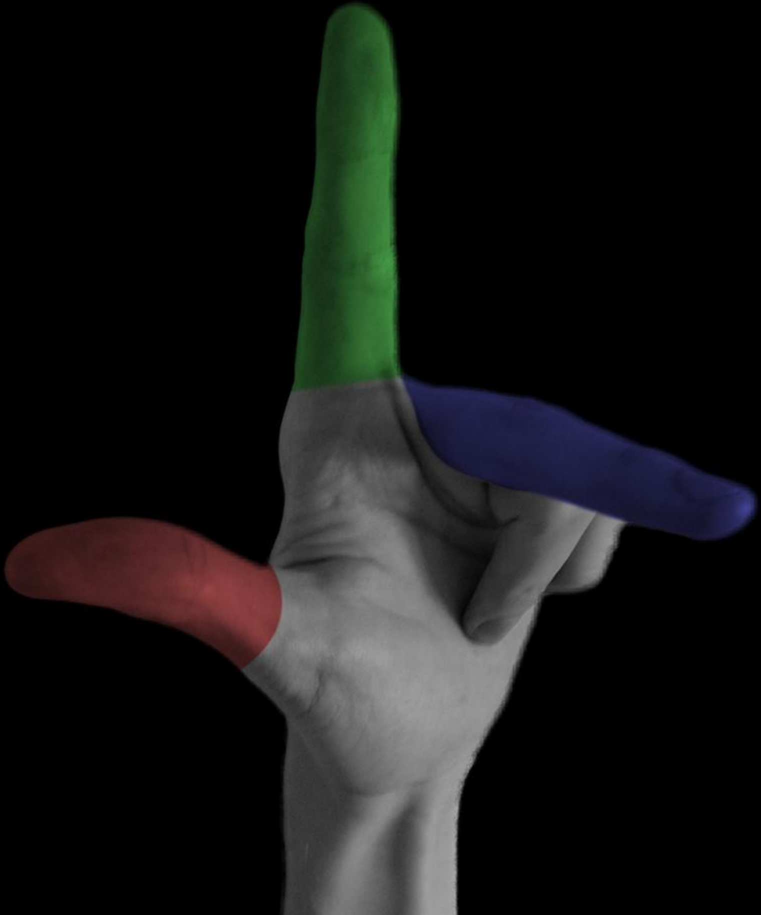
Before we move on…

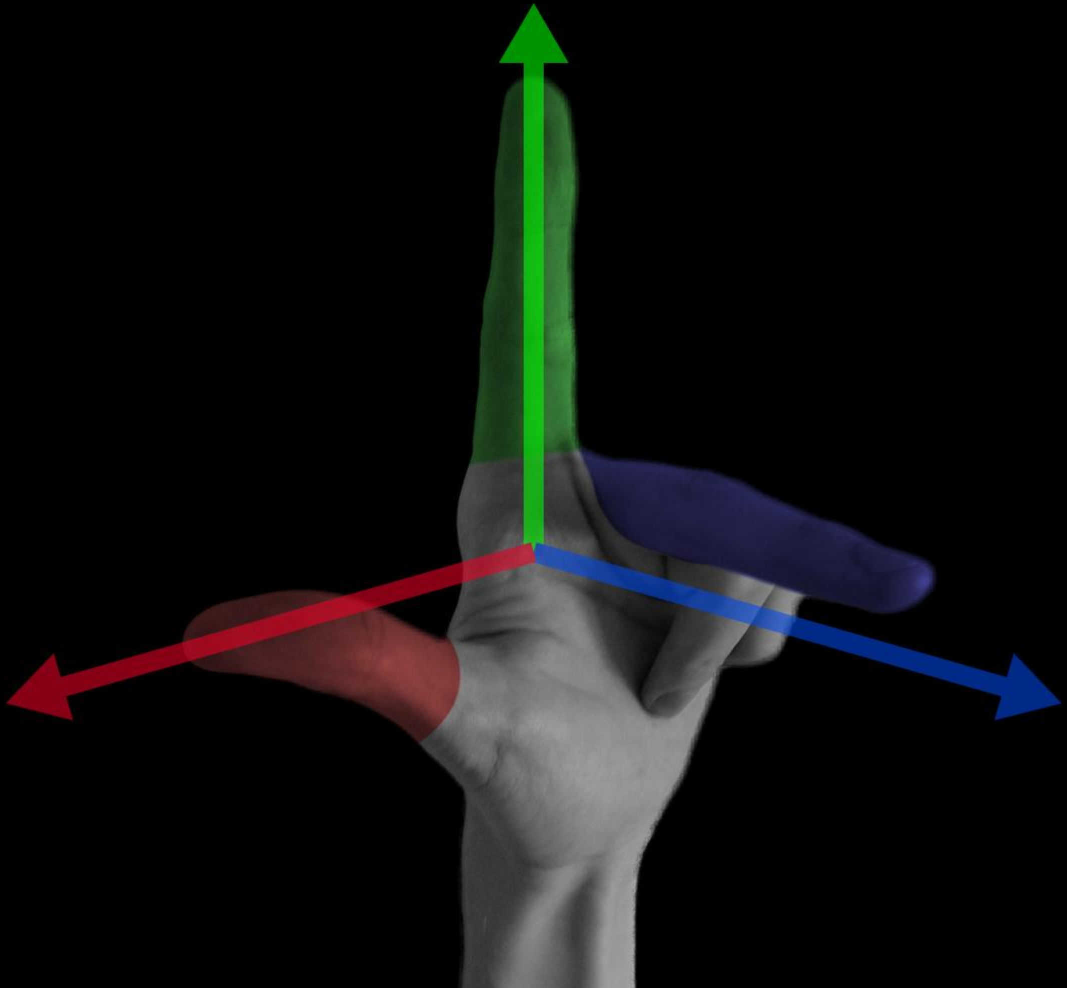
What is space?

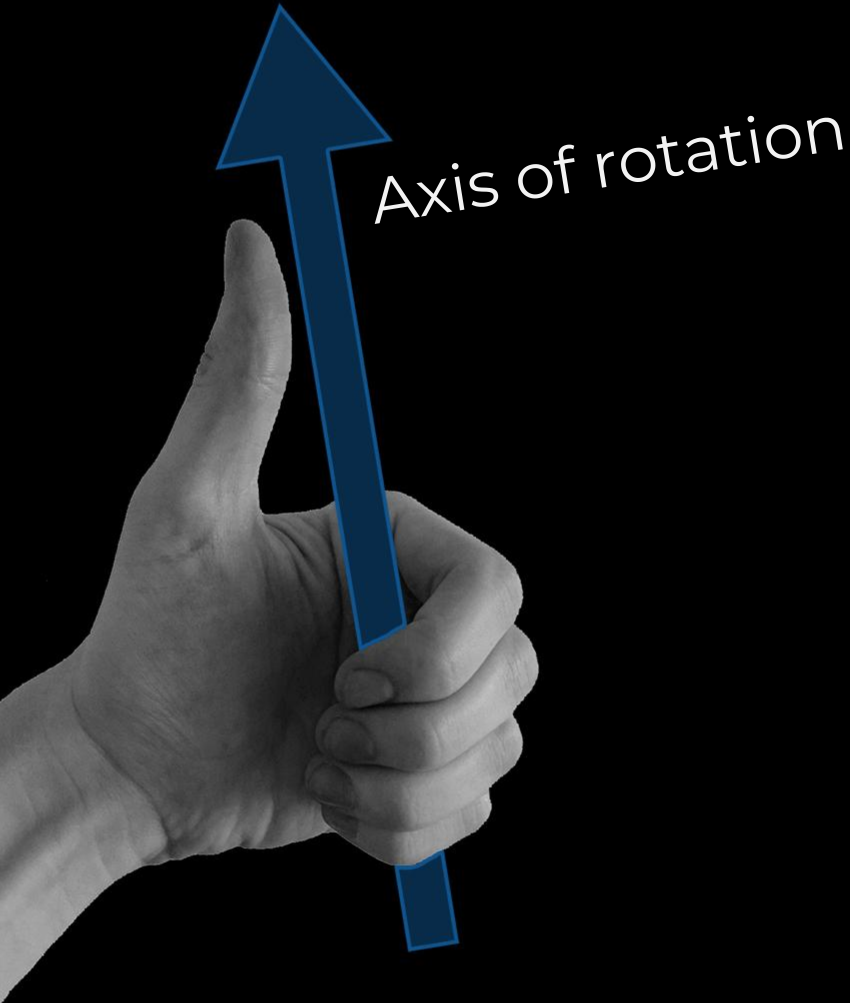
XYZ

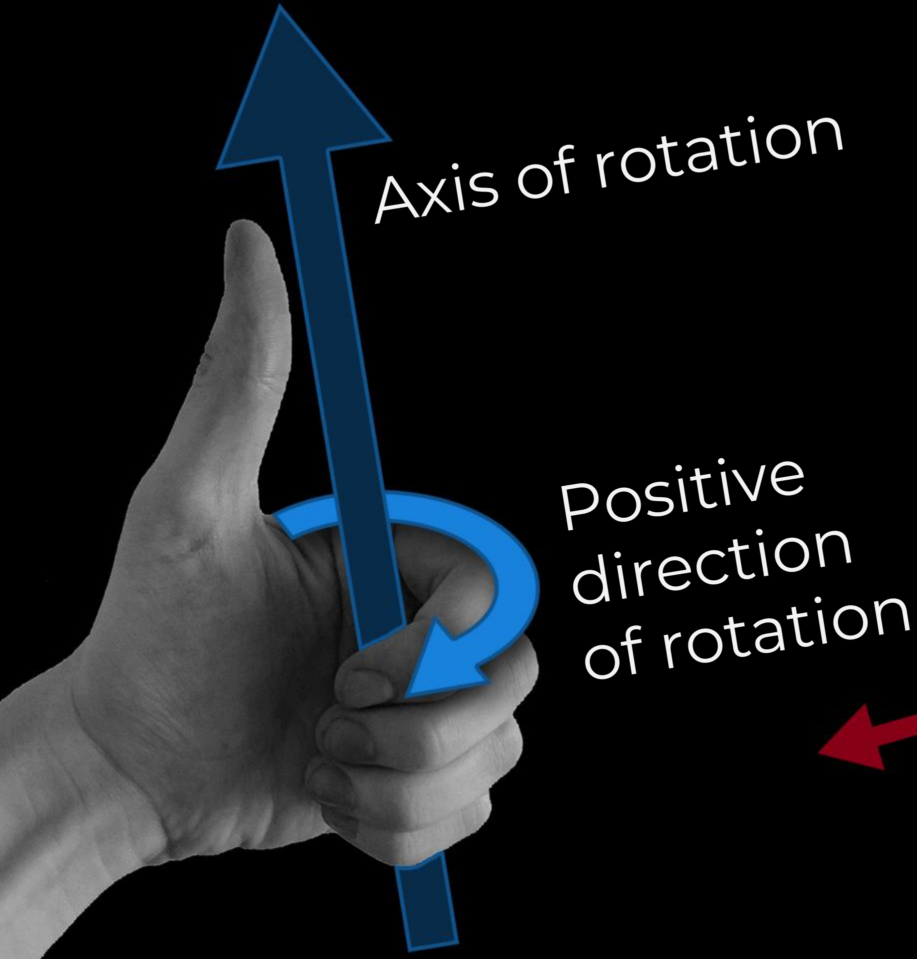
RGB

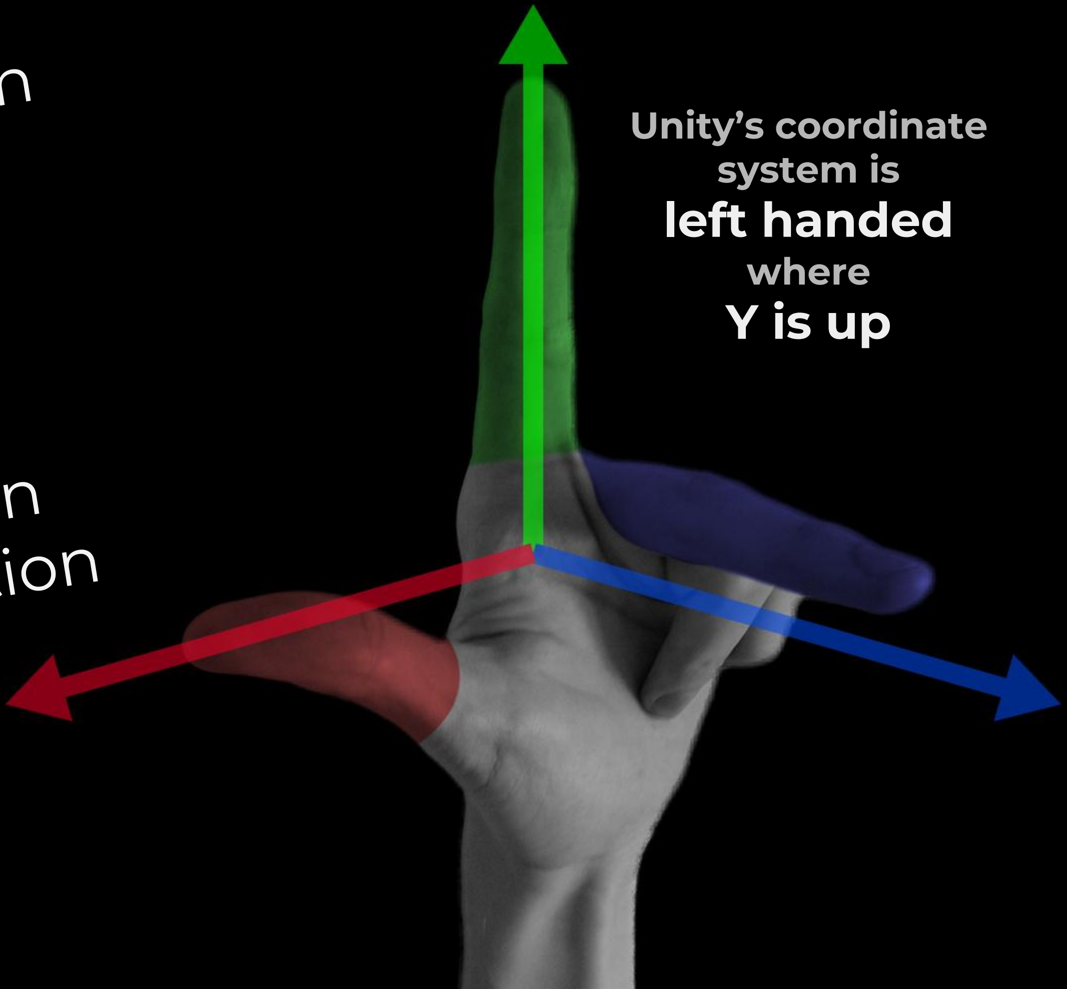




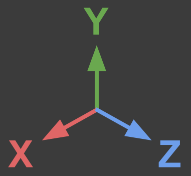
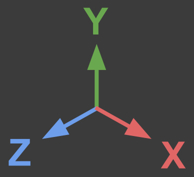
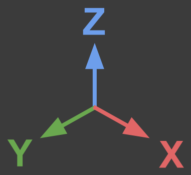
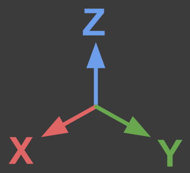



Axis of rotation





Unity's coordinate system is **left handed** where **Y is up**

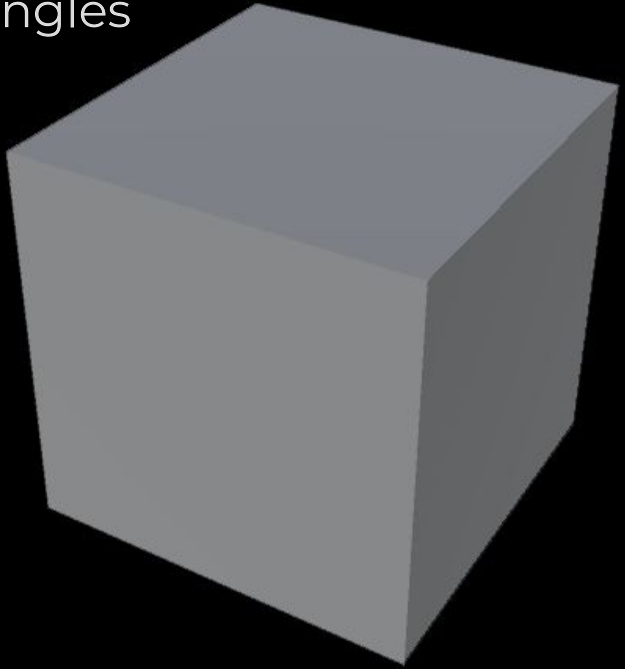
	Left-handed	Right-handed
Y is up		
Z is up		

	Left-handed	Right-handed
Y is up	unity DirectX®	MAYA OpenGL™
Z is up	UNREAL ENGINE	MAX source CRYENGINE® blender

Mini-quiz!

How many triangles does this cube have?

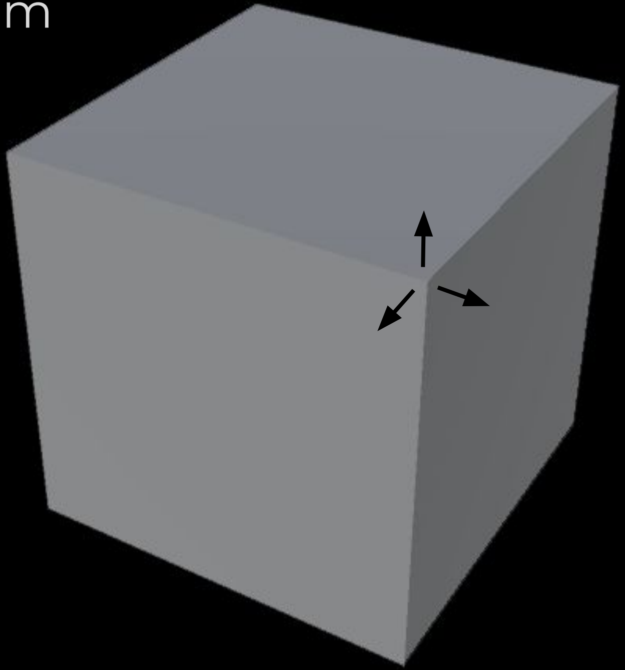
- A cube has six faces
- Each face has two triangles
- 6 x 2 = 12
- 12 triangles



How many vertices does this cube have?

- 4 vertices at the top
- 4 vertices at the bottom
- 8 vertices in total
- (Mathematically)

- 8 corners
- 3 normals per corner
- 8 x 3 = 24
- 24 vertices
- (Technically)



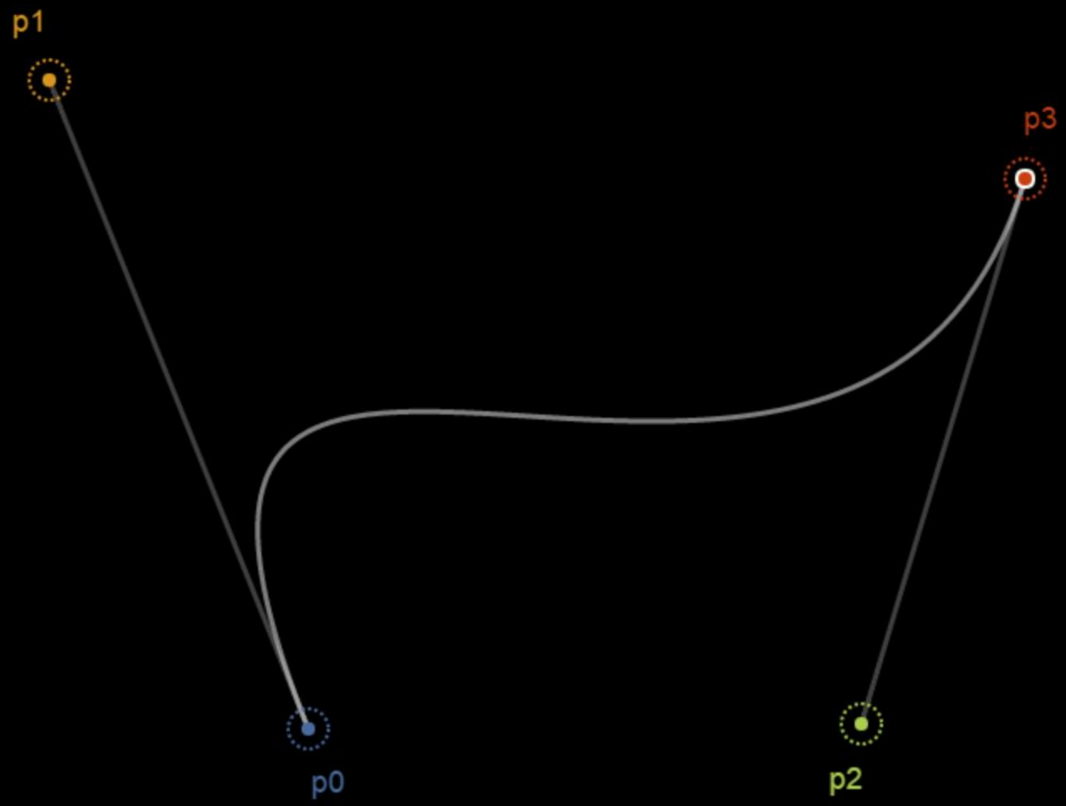
Thinking about vertices

- Think of a vertex as a container of data
- A vertex can only have a single position/normal/tangent, etc.

Vertex (Position, UV, Normal, Tangent, Color, …)

- A vertex has to split into multiple ones, if any of the data above needs to be unique for any of the connected triangles
- Hard-edges create additional vertices (Multiple normals)
- UV seams create additional vertices (Multiple UVs)

Bézier splines (Cubic)



Getting a point in the spline

```
Vector3 GetPoint( Vector3[] pts, float t ) {
    Vector3 a = Vector3.Lerp( pts[0], pts[1], t );
    Vector3 b = Vector3.Lerp( pts[1], pts[2], t );
    Vector3 c = Vector3.Lerp( pts[2], pts[3], t );
    Vector3 d = Vector3.Lerp( a, b, t );
    Vector3 e = Vector3.Lerp( b, c, t );
    return Vector3.Lerp( d, e, t );
}
```

Can we optimize this?

```
a = lerp( p0, p1, t )
b = lerp( p1, p2, t )
c = lerp( p2, p3, t )
d = lerp( a, b, t )
e = lerp( b, c, t )
pt = lerp( d, e, t )
```

Before we get started...

What is a Lerp?

- a, b, t
- start, end, percentage
- if t = 0, return a
- if t = 1, return b
- if t = 0.5, return avg.
- Weighted sum
- return (1-t)a + tb

```
v = lerp( a, b, t )
v = (1-t)a + tb
v = a - ta + tb
v = a + t(b-a)
```

```
a = lerp( p0, p1, t )
b = lerp( p1, p2, t )
c = lerp( p2, p3, t )
d = lerp( a, b, t )
e = lerp( b, c, t )
pt = lerp( d, e, t )
```

```
a = (1-t)p0 + tp1
b = (1-t)p1 + tp2
c = (1-t)p2 + tp3
d = (1-t)a + tb
e = (1-t)b + tc
pt = (1-t)d + te
```

Simplifying the math

$$\begin{tikzcd}
\boxed{(1-t)p0 + tp1} \arrow[dr] & \boxed{(1-t)p1 + tp2} \arrow[d] \arrow[drr] & & \boxed{(1-t)p2 + tp3} \arrow[d] \\
& (1-t)a + tb & & (1-t)b + tc
\end{tikzcd}$$

$$pt = (1-t)d + te$$

$$pt = (1-t)\overset{\boxed{(1-t)((1-t)p0+tp1)+t((1-t)p1+tp2)}}{\overset{\big\downarrow}{d}} + t\overset{\boxed{(1-t)((1-t)p1+tp2)+t((1-t)p2+tp3)}}{\overset{\big\downarrow}{e}}$$

$$pt = (1-t)((1-t)((1-t)p0+tp1)+t((1-t)p1+tp2)) + t((1-t)((1-t)p1+tp2)+t((1-t)p2+tp3))$$

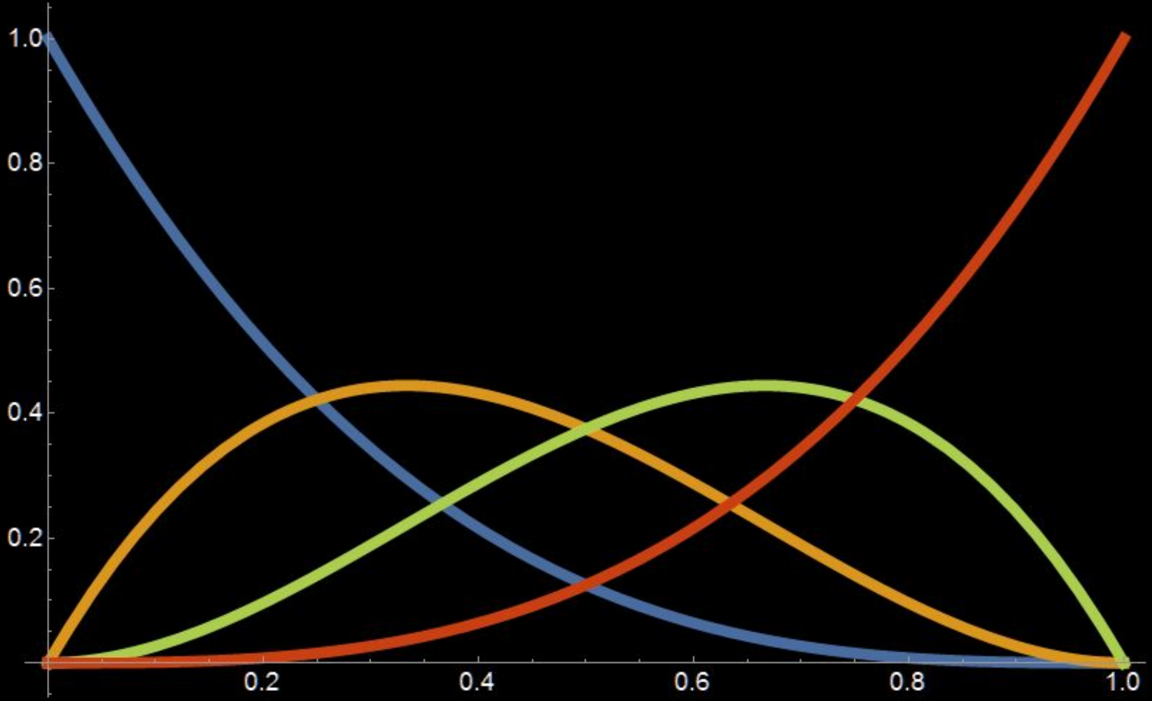
$$pt = -p0t^3 + 3p0t^2 - 3p0t + p0 + 3p1t^3 - 6p1t^2 + 3p1t - 3p2t^3 + 3p2t^2 + p3t^3$$

$$
\begin{aligned}
pt = {} & \\
& p0(-t^3+3t^2-3t+1) + \\
& p1(3t^3-6t^2+3t) + \\
& p2(-3t^3+3t^2) + \\
& p3t^3
\end{aligned}
$$

$$
\begin{aligned}
pt = {} & \\
& p0((1-t)^3) + \\
& p1(3(1-t)^2t) + \\
& p2(3(1-t)t^2) + \\
& p3(t^3)
\end{aligned}
$$

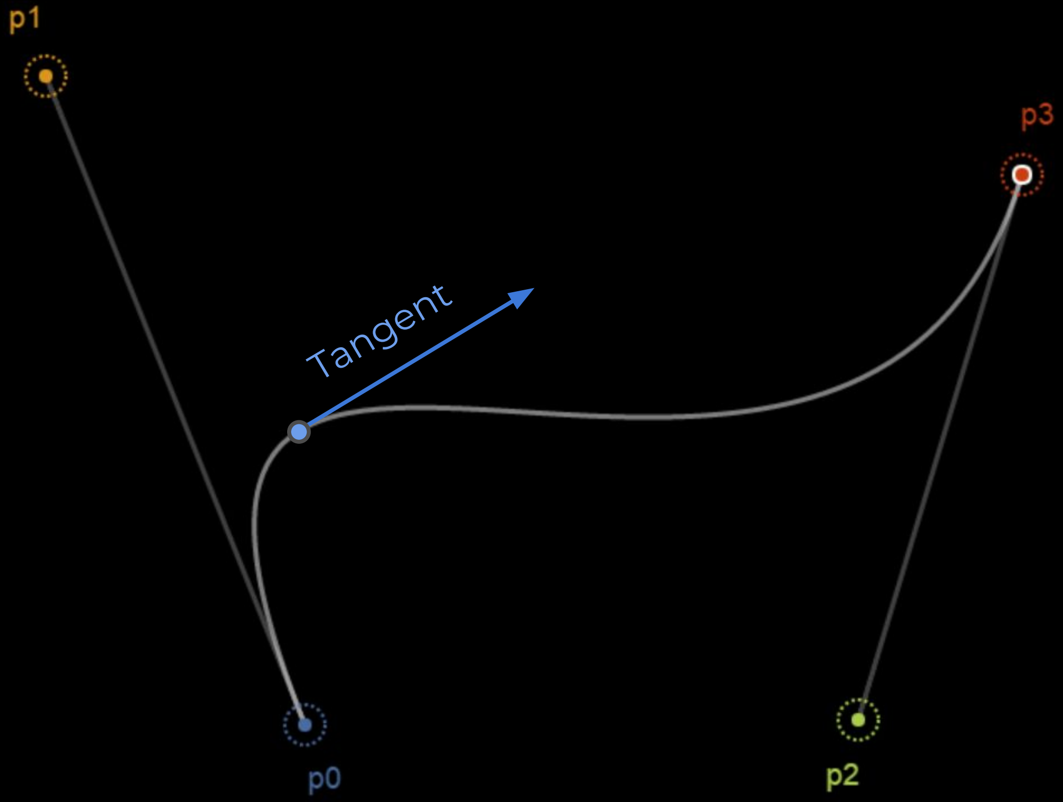
Point basis functions

```
pt =
— p0((1-t)³) +
— p1(3(1-t)²t) +
— p2(3(1-t)t²) +
— p3(t³)
```



Optimized GetPoint(...)

```
Vector3 GetPoint( Vector3[] pts, float t ) {
    float omt = 1f-t;
    float omt2 = omt * omt;
    float t2 = t * t;
    return pts[0] * ( omt2 * omt ) +
           pts[1] * ( 3f * omt2 * t ) +
           pts[2] * ( 3f * omt * t2 ) +
           pts[3] * ( t2 * t );
}
```


p1
p3
Tangent
p0
p2

$e-d = ((1-t)((1-t)p1 + tp2) + t((1-t)p2 + tp3))-((1-t)((1-t)p0 + tp1) + t((1-t)p1 + tp2))$

$e-d =$

$p0(-t^2+2t-1) +$

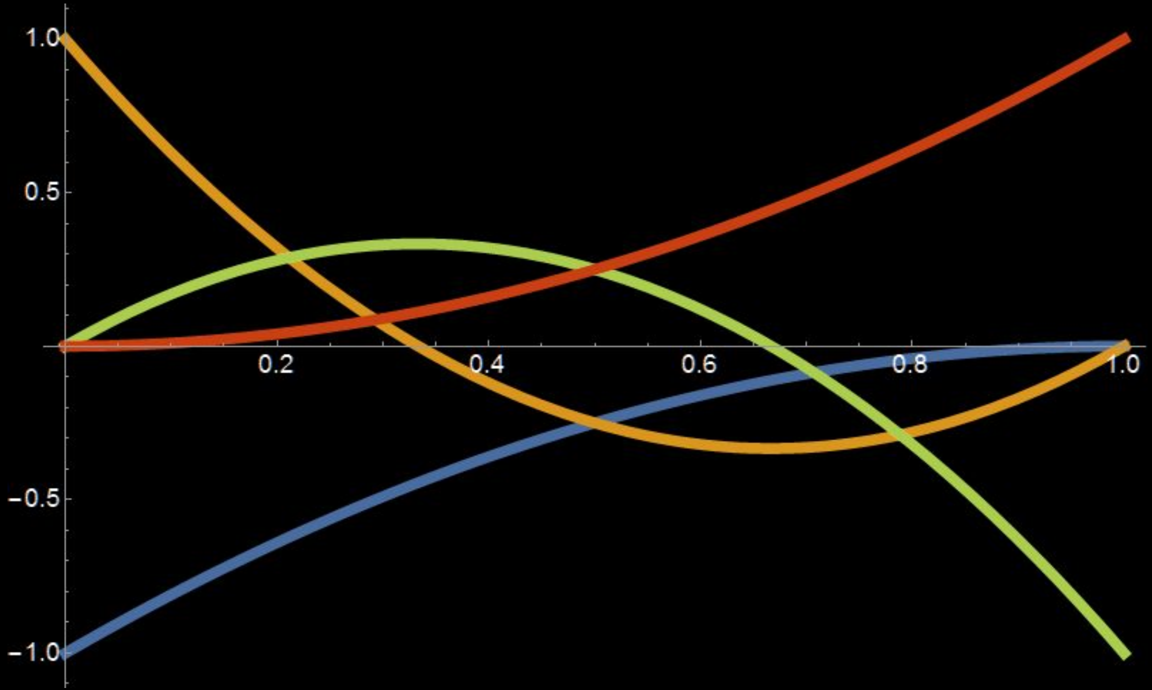
$p1(3t^2-4t+1) +$

$p2(-3t^2+2t) +$

$p3t^2$

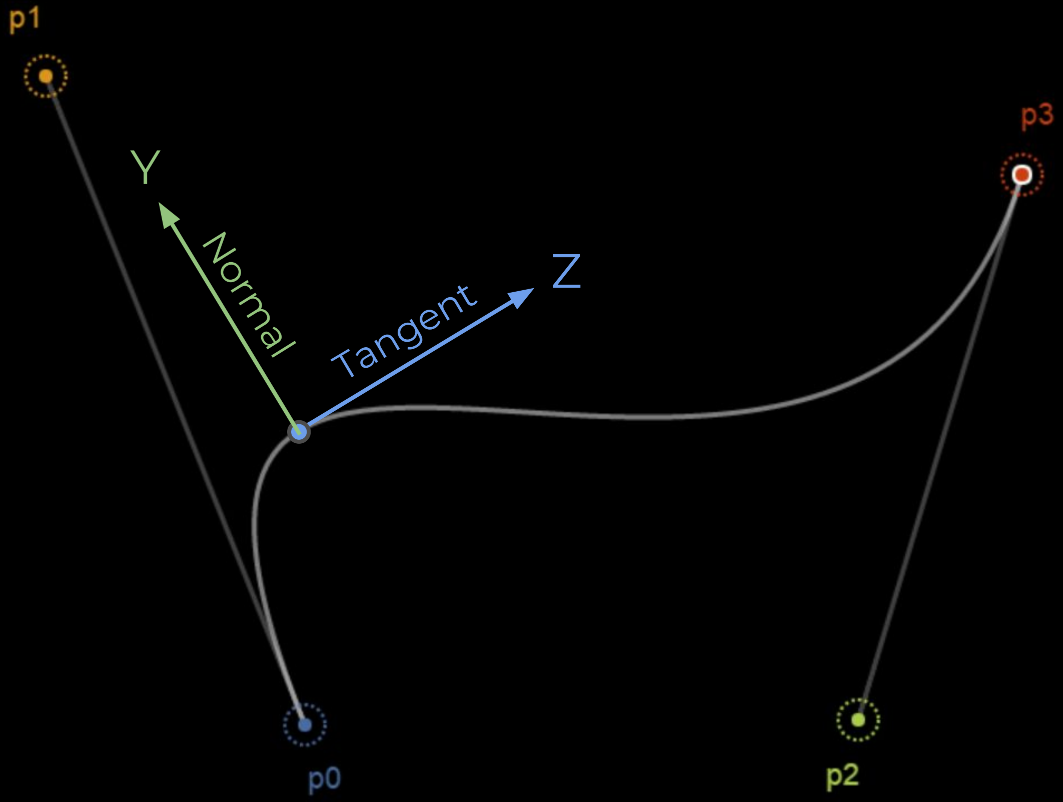
Tangent basis functions

```
pt =
— p0(-(1-t)²) +
— p1(t(3t-4)+1) +
— p2(-3t²+2t) +
— p3(t²)
```



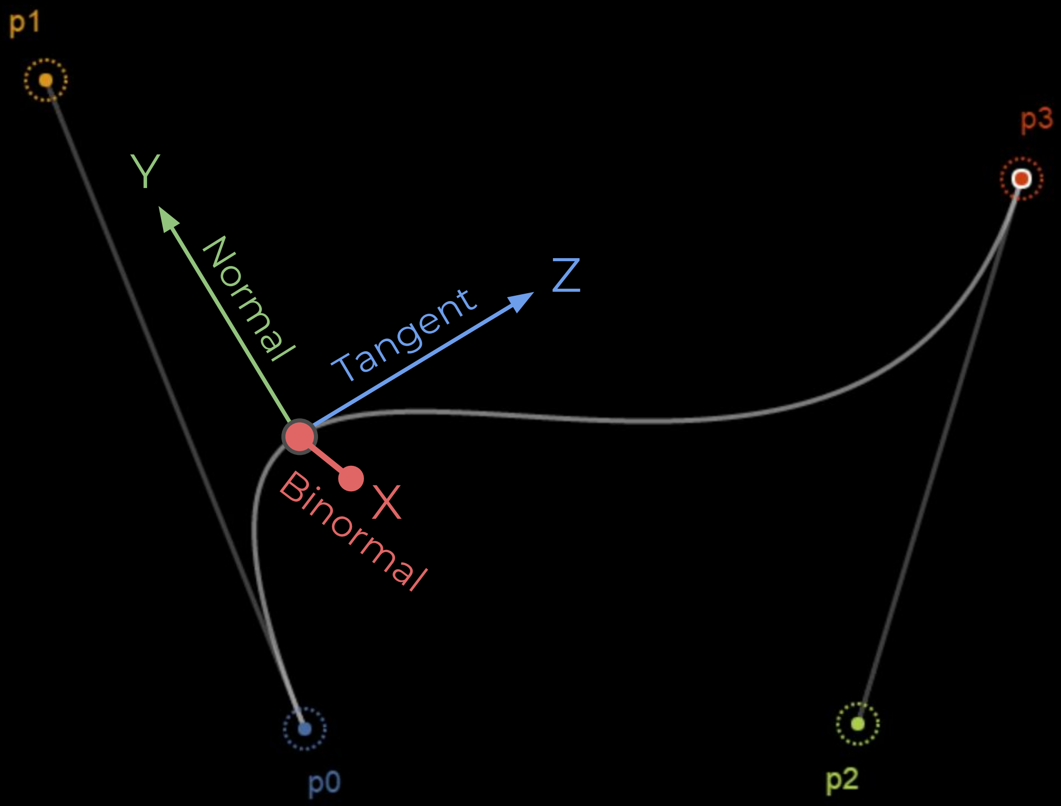
GetTangent(…)

```
Vector3 GetTangent( Vector3[] pts, float t ) {
    float omt = 1f-t;
    float omt2 = omt * omt;
    float t2 = t * t;
    Vector3 tangent =
            pts[0] * ( -omt2 ) +
            pts[1] * ( 3 * omt2 - 2 * omt ) +
            pts[2] * ( -3 * t2 + 2 * t ) +
            pts[3] * ( t2 );
    return tangent.normalized;
}
```


p1
p3
Y
Normal
Z
Tangent
p0
p2

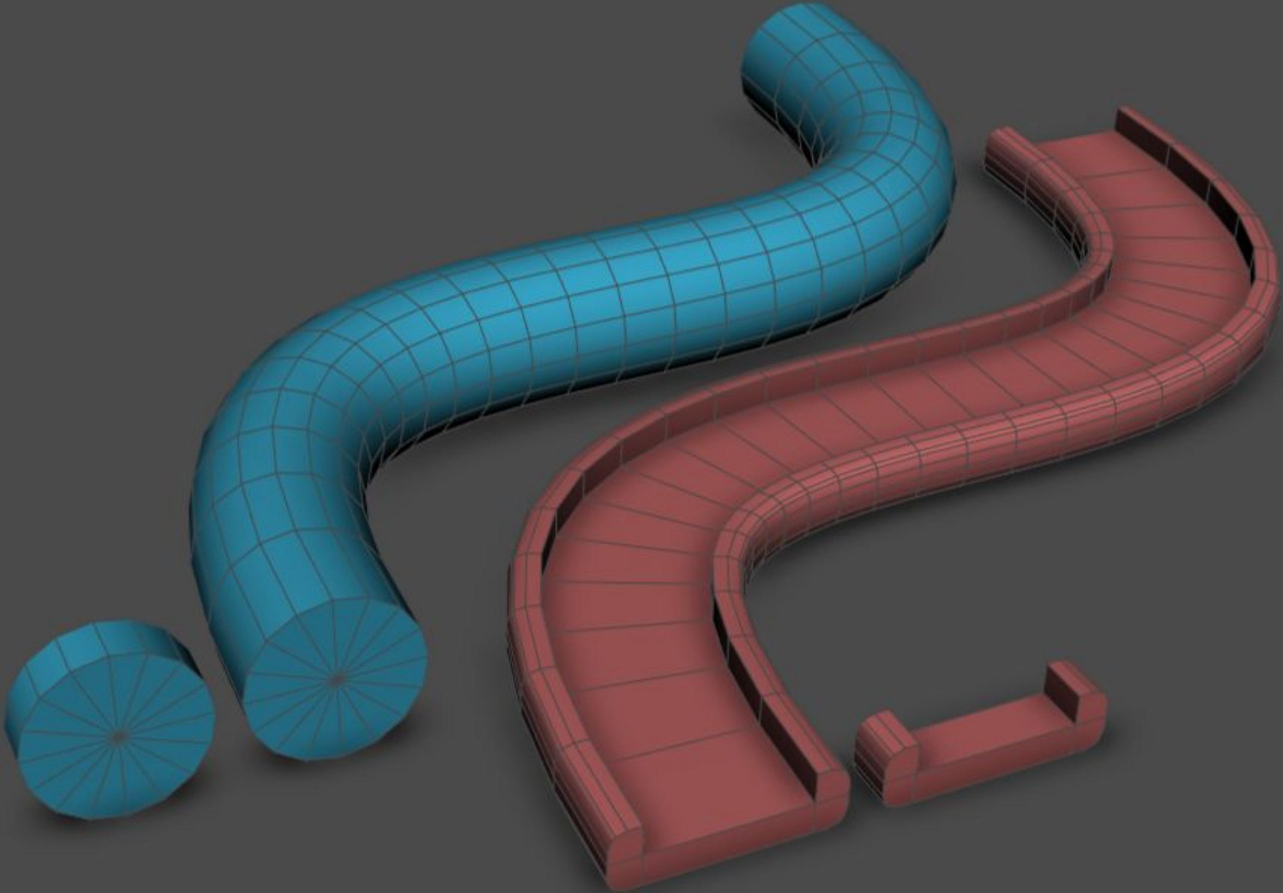
GetNormal(…)

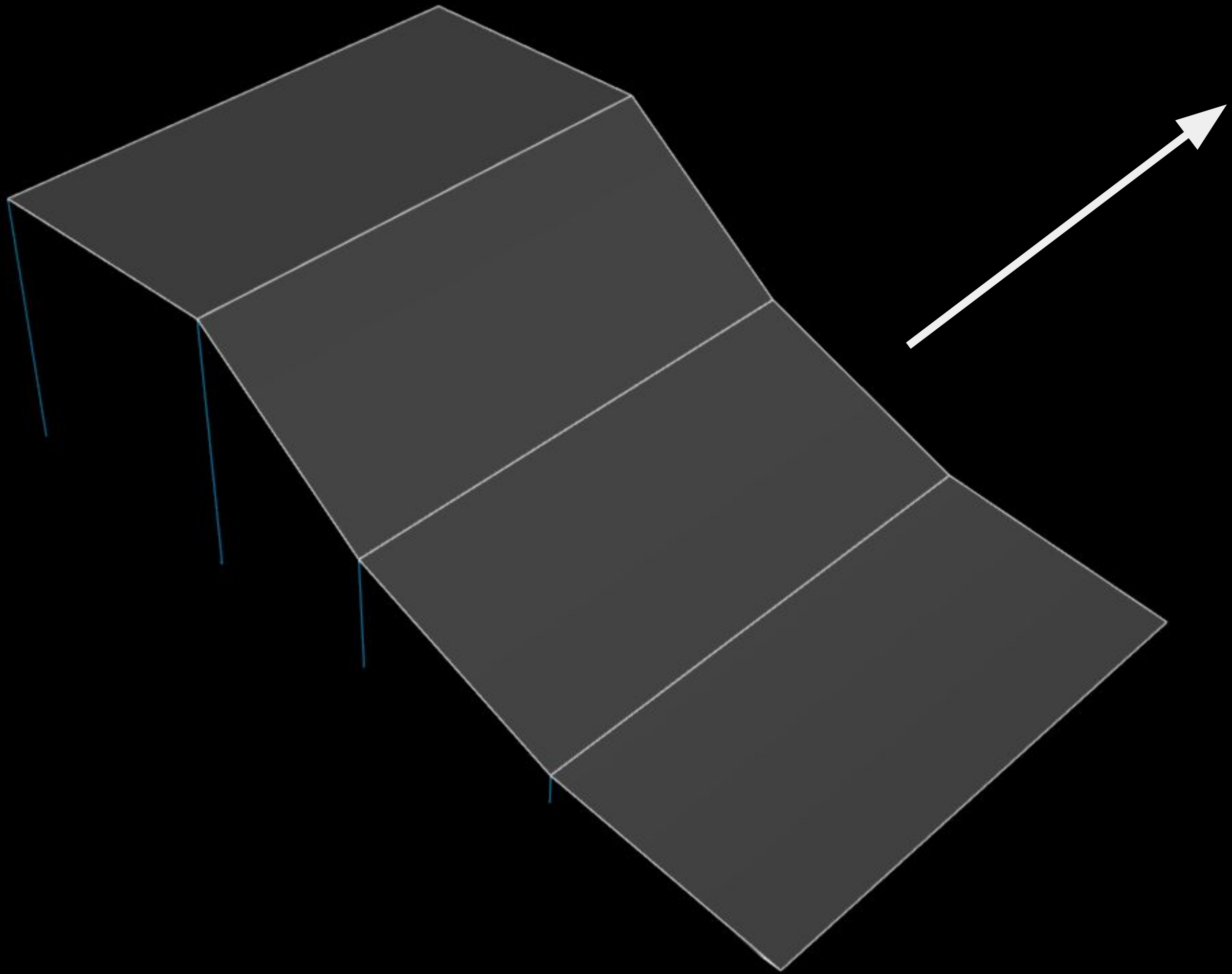
```
Vector3 GetNormal2D( Vector3[] pts, float t ) {
    Vector3 tng = GetTangent( pts, t );
    return new Vector3( -tng.y, tng.x, 0f );
}

Vector3 GetNormal3D( Vector3[] pts, float t, Vector3 up ) {
    Vector3 tng = GetTangent( pts, t );
    Vector3 binormal = Vector3.Cross( up, tng ).normalized;
    return Vector3.Cross( tng, binormal );
}
```


p1
Y
Normal
Z
Tangent
p3
X
Binormal
p0
p2

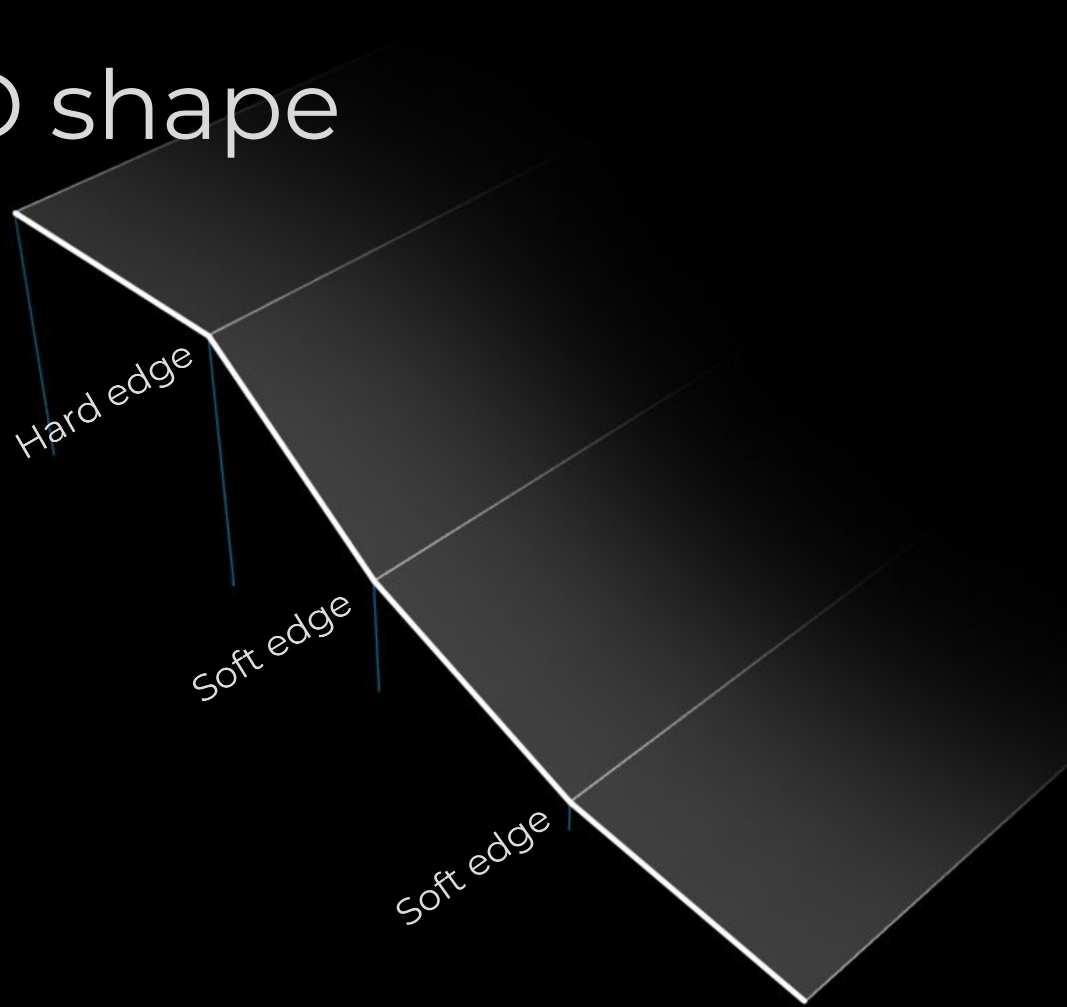
GetOrientation(…)

```
Quaternion GetOrientation2D( Vector3[] pts, float t ) {
    Vector3 tng = GetTangent( pts, t );
    Vector3 nrm = GetNormal2D( pts, t );
    return Quaternion.LookRotation( tng, nrm );
}

Quaternion GetOrientation3D( Vector3[] pts, float t, Vector3 up ) {
    Vector3 tng = GetTangent( pts, t );
    Vector3 nrm = GetNormal3D( pts, t, up );
    return Quaternion.LookRotation( tng, nrm );
}
```



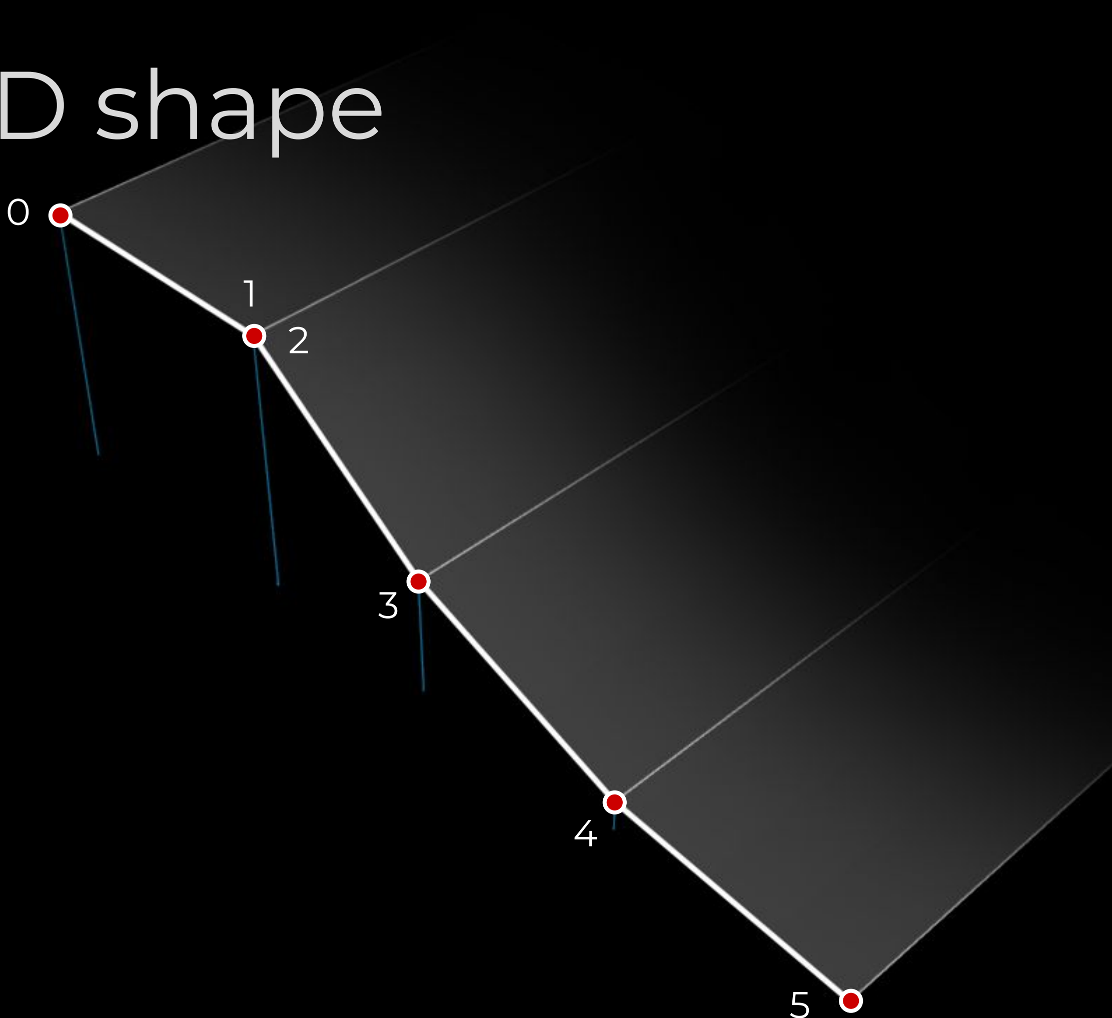


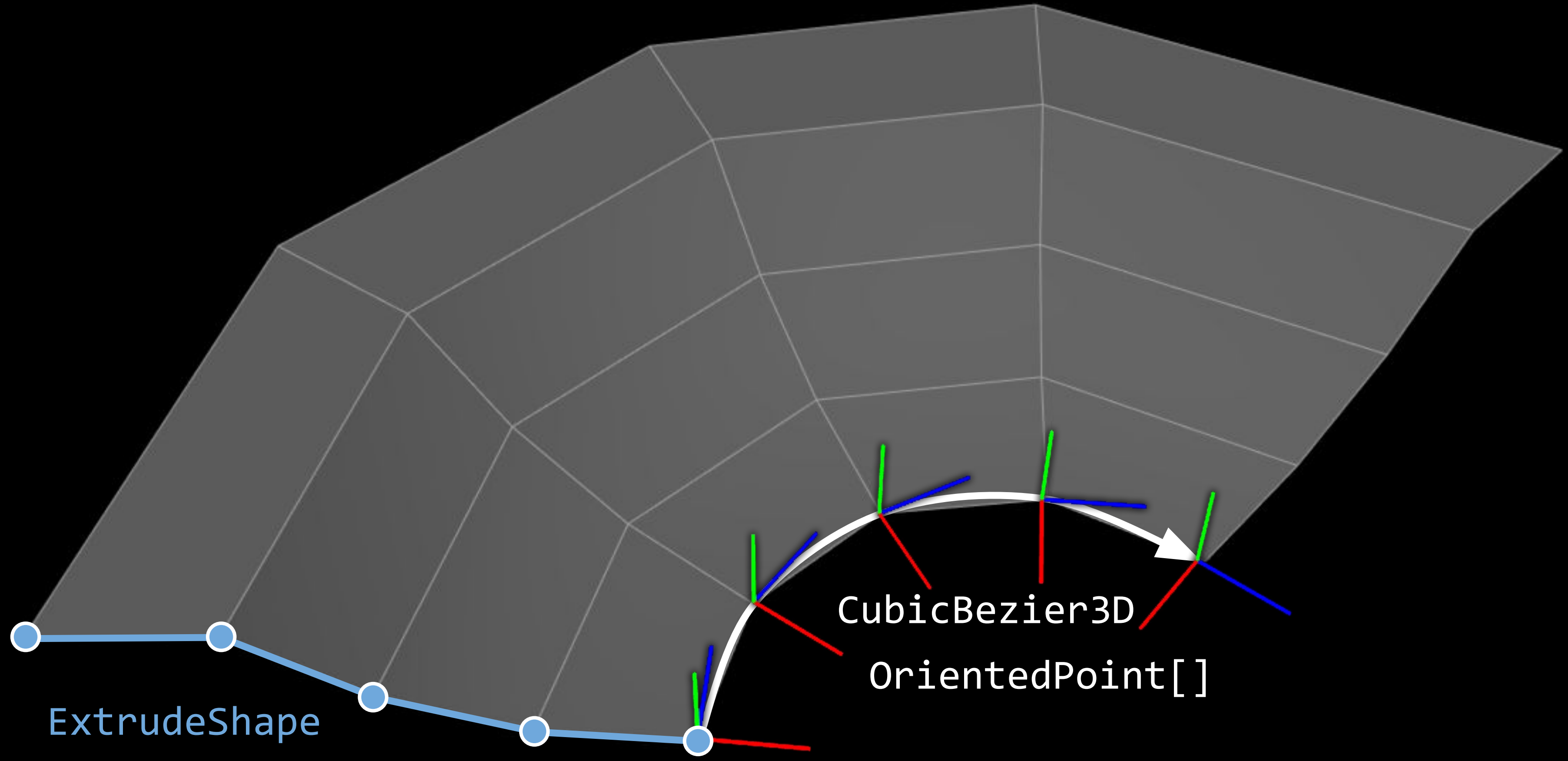
Defining the 2D shape



Defining the 2D shape

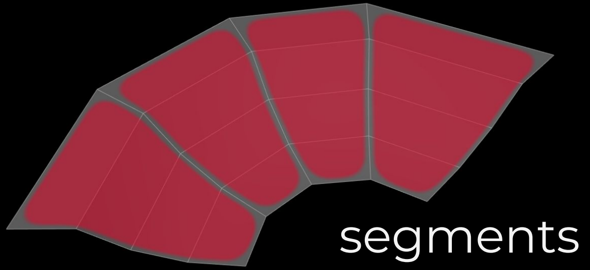
```
Vector2[] verts;
Vector2[] normals;
float[] us;
int[] lines = new int[]{
    0, 1,
    2, 3,
    3, 4,
    4, 5
};
```

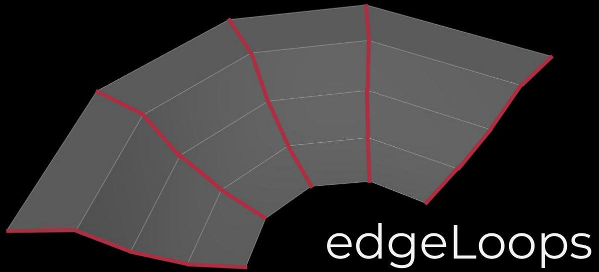


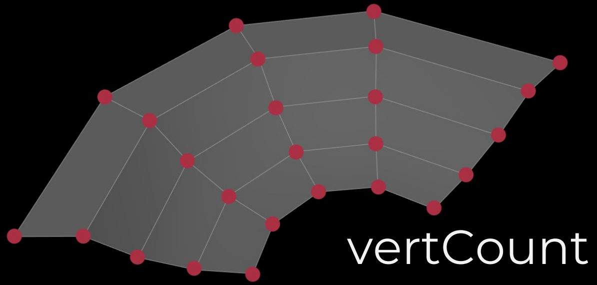

CubicBezier3D
OrientedPoint[]
ExtrudeShape

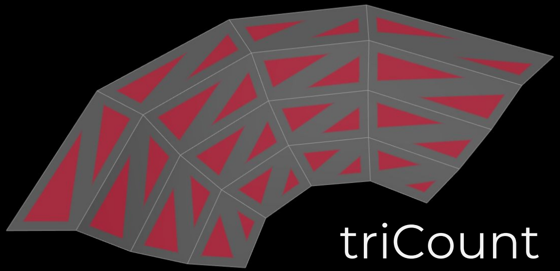
```
public struct OrientedPoint {

    public Vector3 position;
    public Quaternion rotation;

    public OrientedPoint( Vector3 position, Quaternion rotation ) {
        this.position = position;
        this.rotation = rotation;
    }

    public Vector3 LocalToWorld( Vector3 point ) {
        return position + rotation * point;
    }

    public Vector3 WorldToLocal( Vector3 point ) {
        return Quaternion.Inverse( rotation ) * ( point - position );
    }

    public Vector3 LocalToWorldDirection( Vector3 dir ) {
        return rotation * dir;
    }

}
```

```
public void Extrude( Mesh mesh, ExtrudeShape shape, OrientedPoint[] path ){

    int vertsInShape = shape.vert2Ds.Length;
    int segments = path.Length - 1;
    int edgeLoops = path.Length;
    int vertCount = vertsInShape * edgeLoops;
    int triCount = shape.lines.Length * segments;
    int triIndexCount = triCount * 3;
```





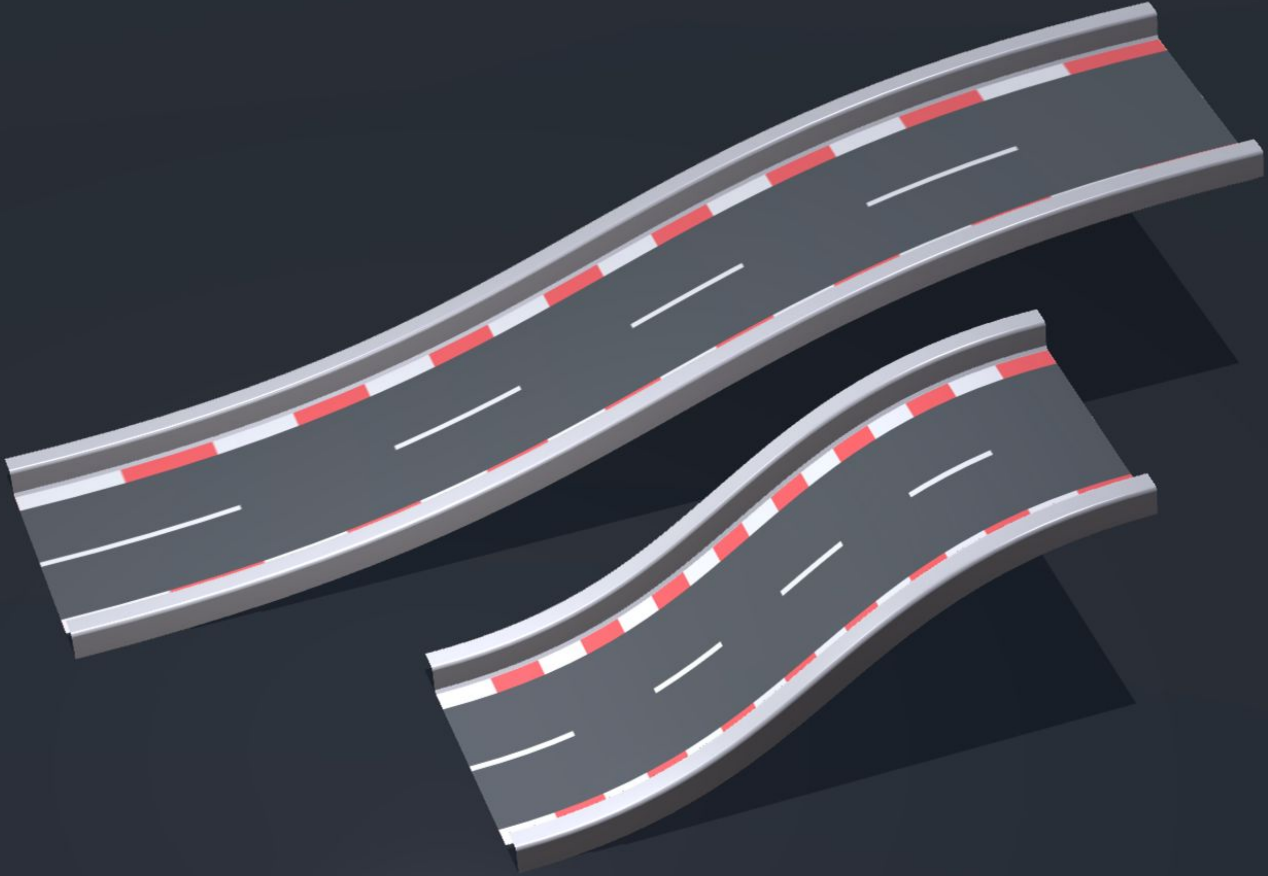




```csharp
int vertsInShape = shape.vert2Ds.Length;
int segments = path.Length - 1;
int edgeLoops = path.Length;
int vertCount = vertsInShape * edgeLoops;
int triCount = shape.lines.Length * segments;
int triIndexCount = triCount * 3;

int[] triangleIndices   = new int[ triIndexCount ];
Vector3[] vertices      = new Vector3[ vertCount ];
Vector3[] normals       = new Vector3[ vertCount ];
Vector2[] uvs           = new Vector2[ vertCount ];

/* Generation code goes here */

mesh.Clear();
mesh.vertices = vertices;
mesh.triangles = triangleIndices;
mesh.normals = normals;
mesh.uv = uvs;
```

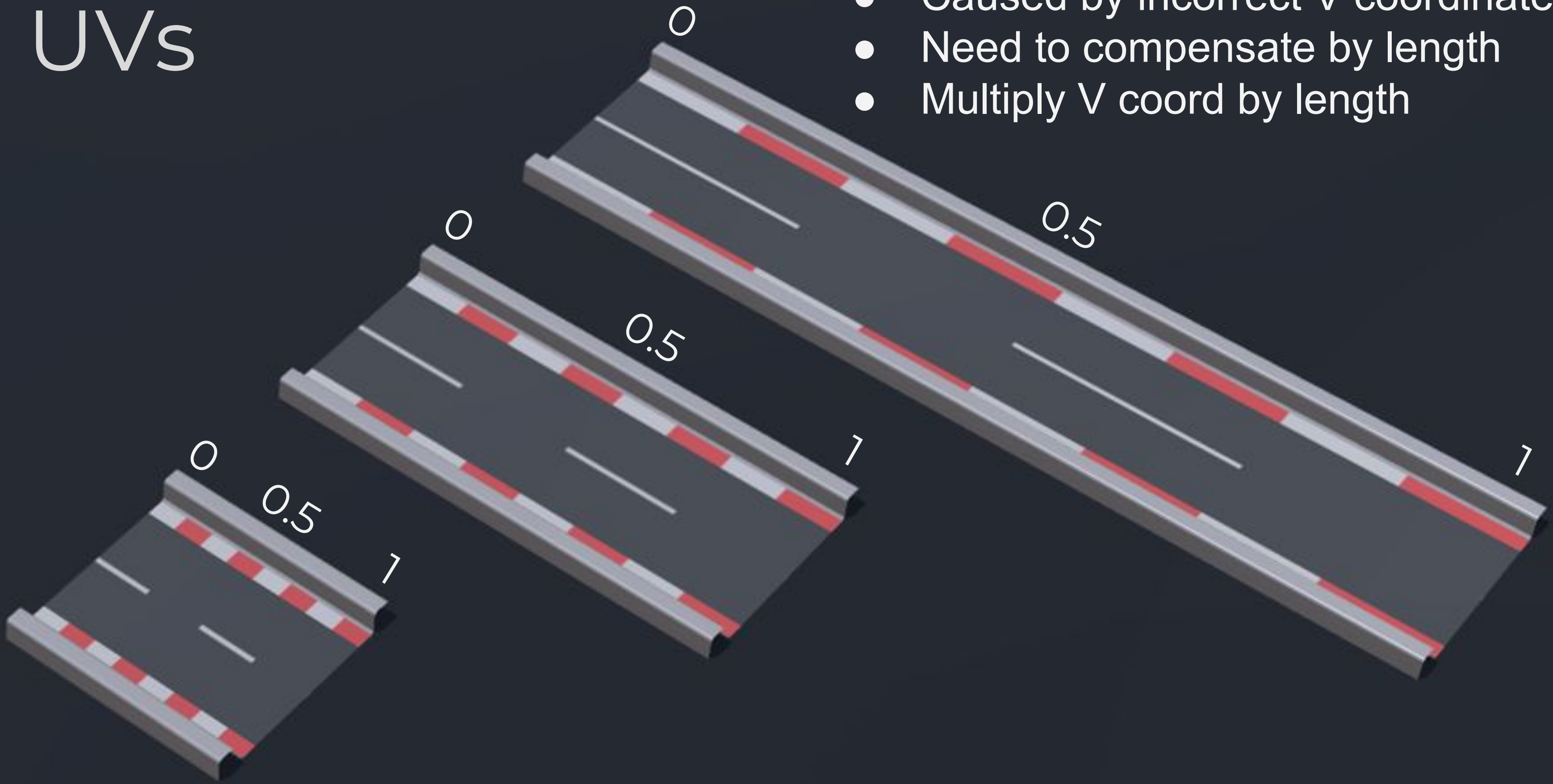
```
foreach oriented point in the path
    foreach vertex in the 2D shape
        Add the vertex position, based on the oriented point
        Add the normal direction, based on the oriented point
        Add the UV. U is based on the shape, V is based on distance along the path
    end
end

foreach segment
    foreach line in the 2D shape
        Add two triangles with vertex indices based on the line indices
    end
end
```

```
for( int i = 0; i < path.Length; i++ ) {
      int offset = i * vertsInShape;
      for( int j = 0; j < vertsInShape; j++ ) {
            int id = offset + j;
            vertices[id] = path[i].LocalToWorld( shape.vert2Ds[j].point );
            normals[id] = path[i].LocalToWorldDirection( shape.vert2Ds[j].normal );
            uvs[id] = new Vector2( shape.vert2Ds[j].uCoord, i / ((float)edgeLoops) );
      }
}
int ti = 0;
for( int i = 0; i < segments; i++ ) {
      int offset = i * vertsInShape;
      for ( int l = 0; l < lines.Length; l += 2 ) {
            int a = offset + lines[l] + vertsInShape;
            int b = offset + lines[l];
            int c = offset + lines[l+1];
            int d = offset + lines[l+1] + vertsInShape;
            triangleIndices[ti] = a;   ti++;
            triangleIndices[ti] = b;   ti++;
            triangleIndices[ti] = c;   ti++;
            triangleIndices[ti] = c;   ti++;
            triangleIndices[ti] = d;   ti++;
            triangleIndices[ti] = a;   ti++;
      }
}
```

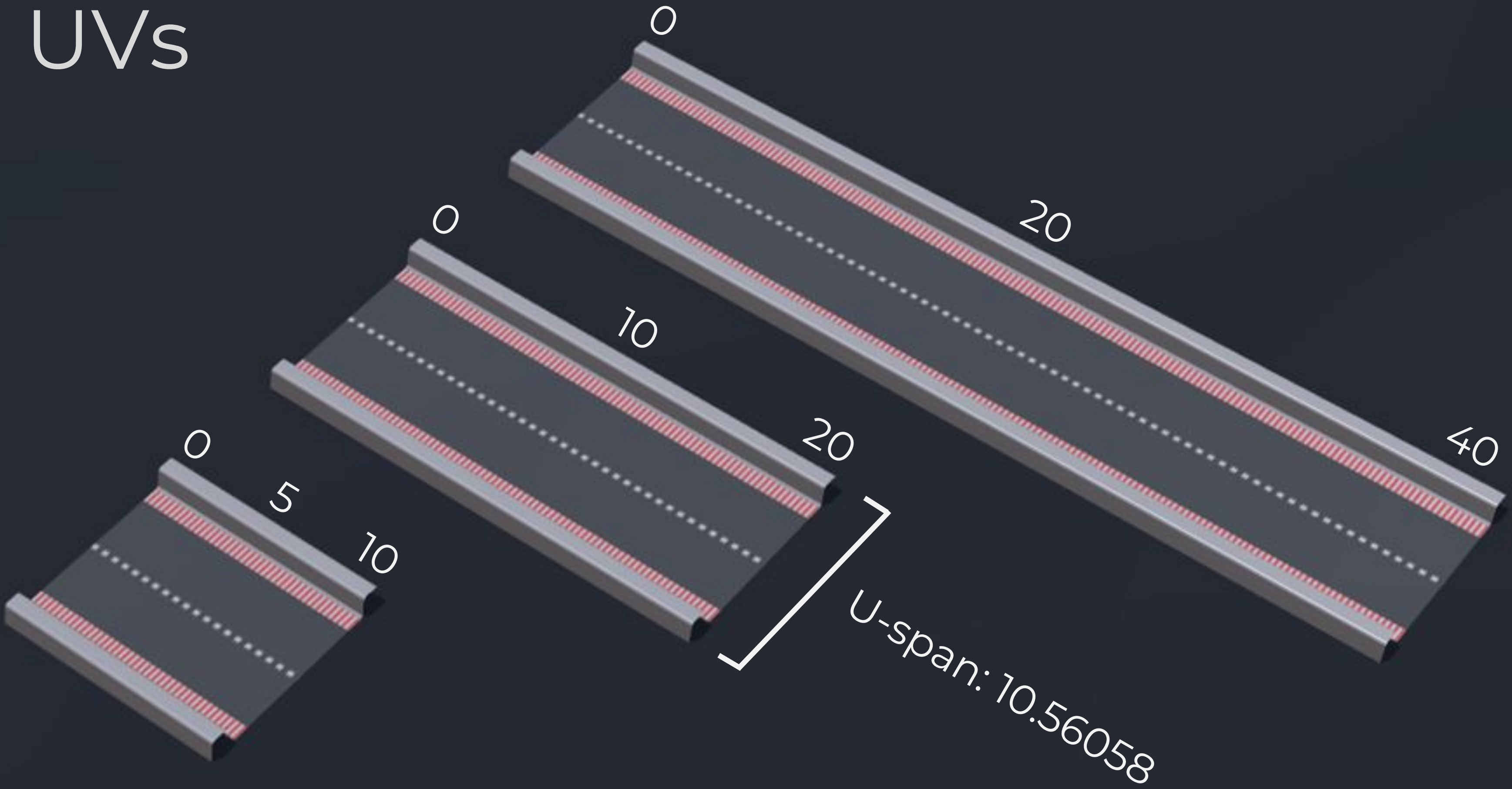


UVs

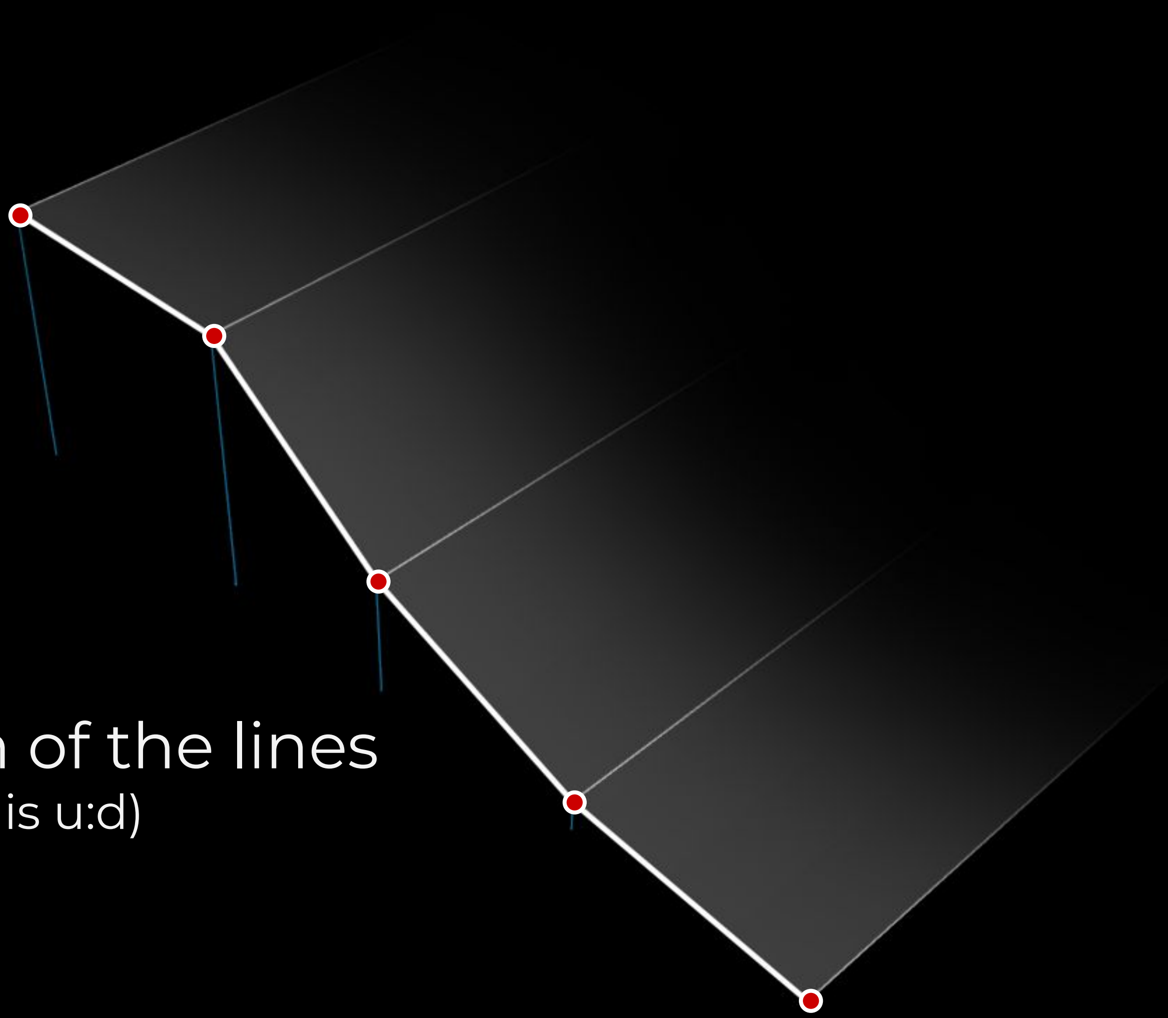
- Caused by incorrect V coordinates
- Need to compensate by length
- Multiply V coord by length



UVs

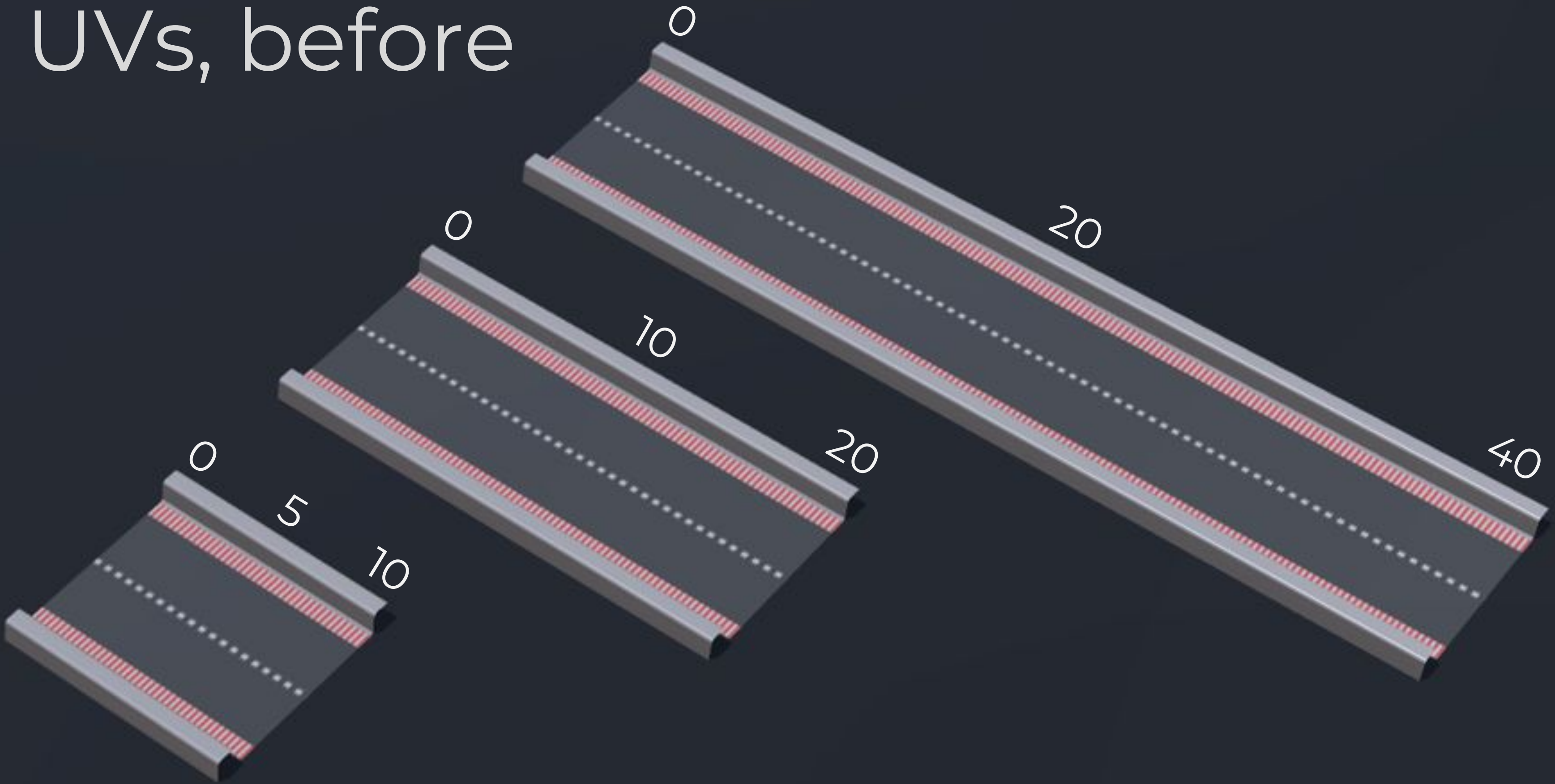


UVs

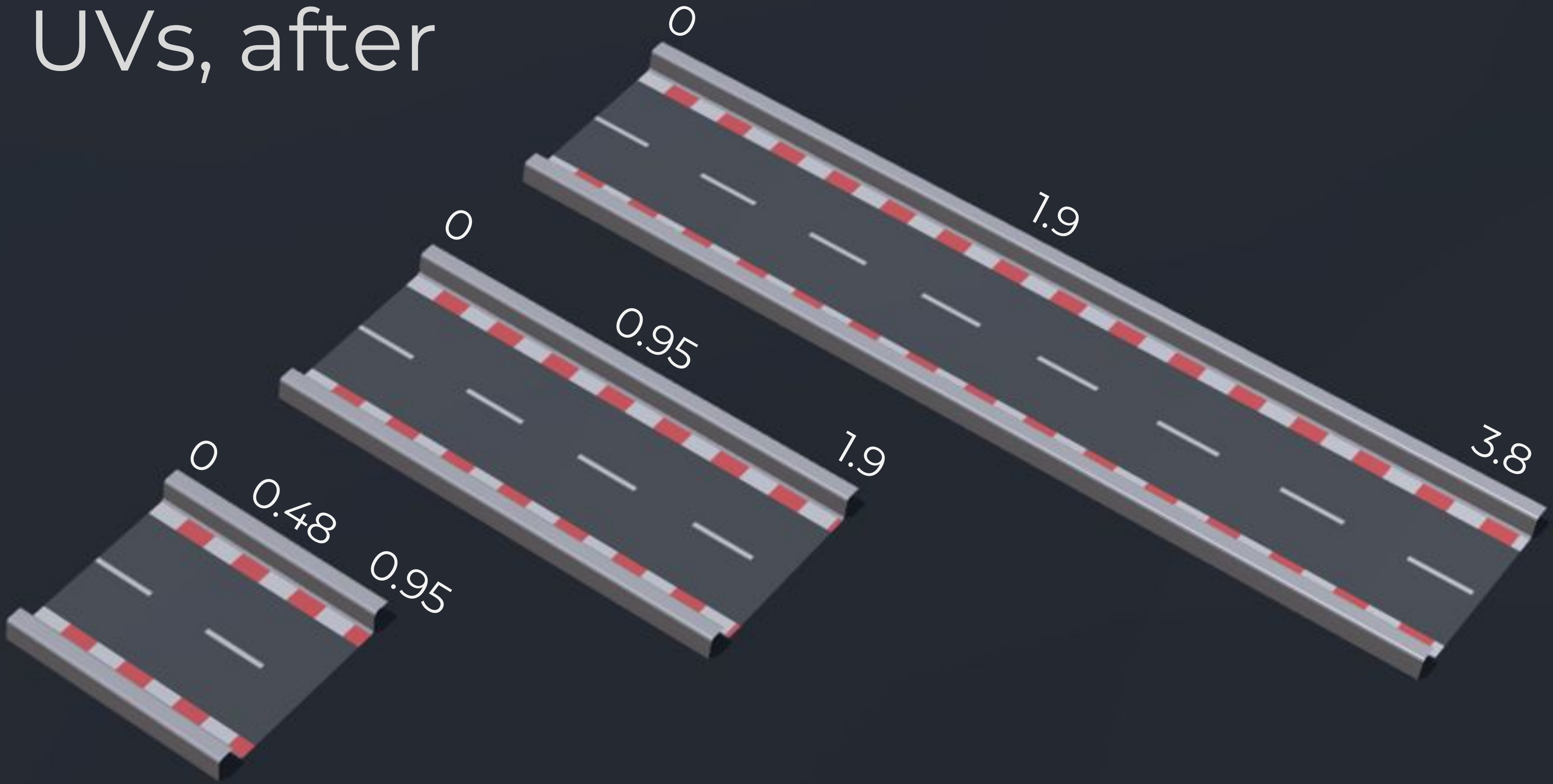


U-span ≈ Total length of the lines
(Depends on how the mesh is u:d)

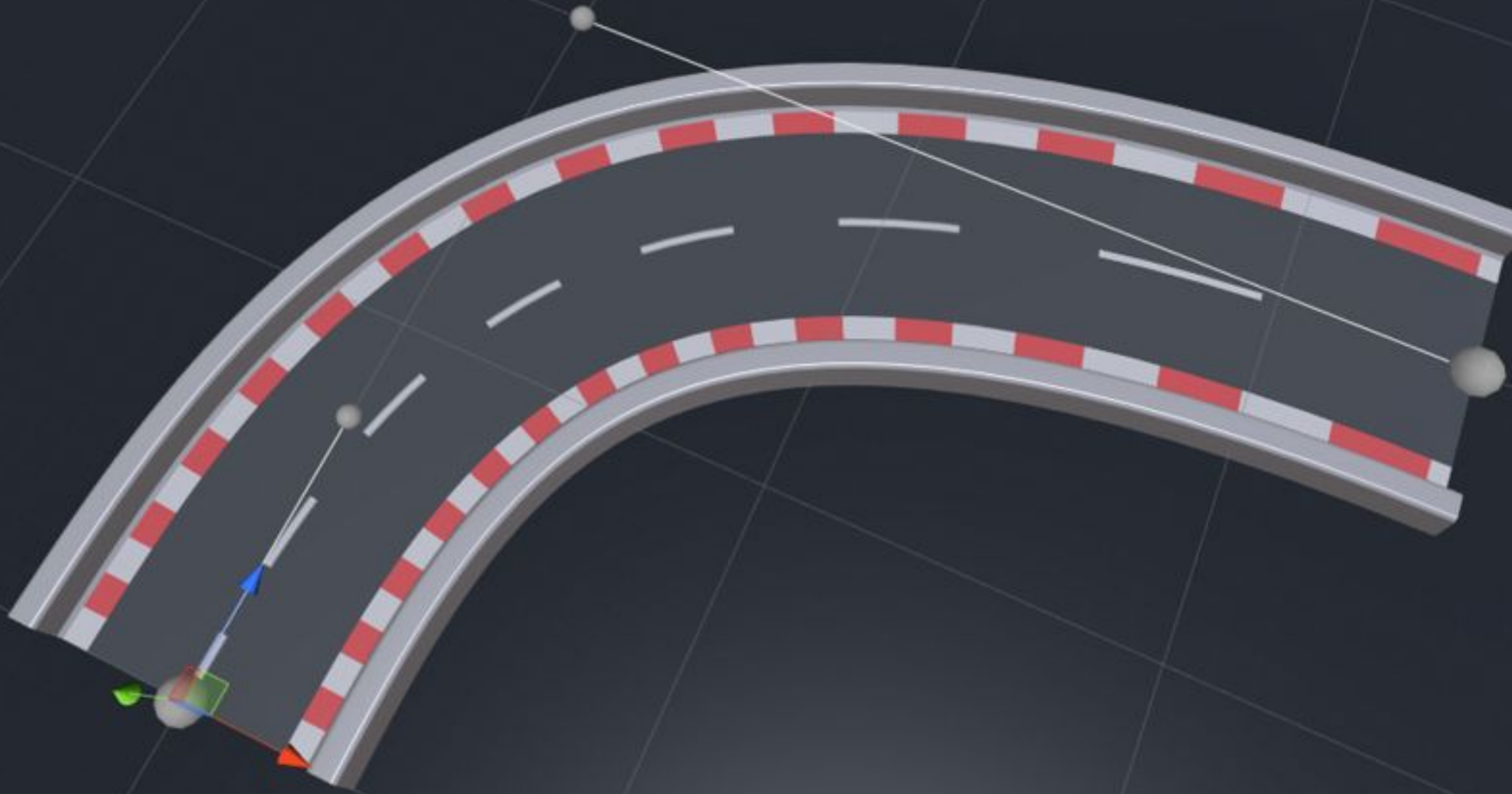
UVs, before



UVs, after



UVs



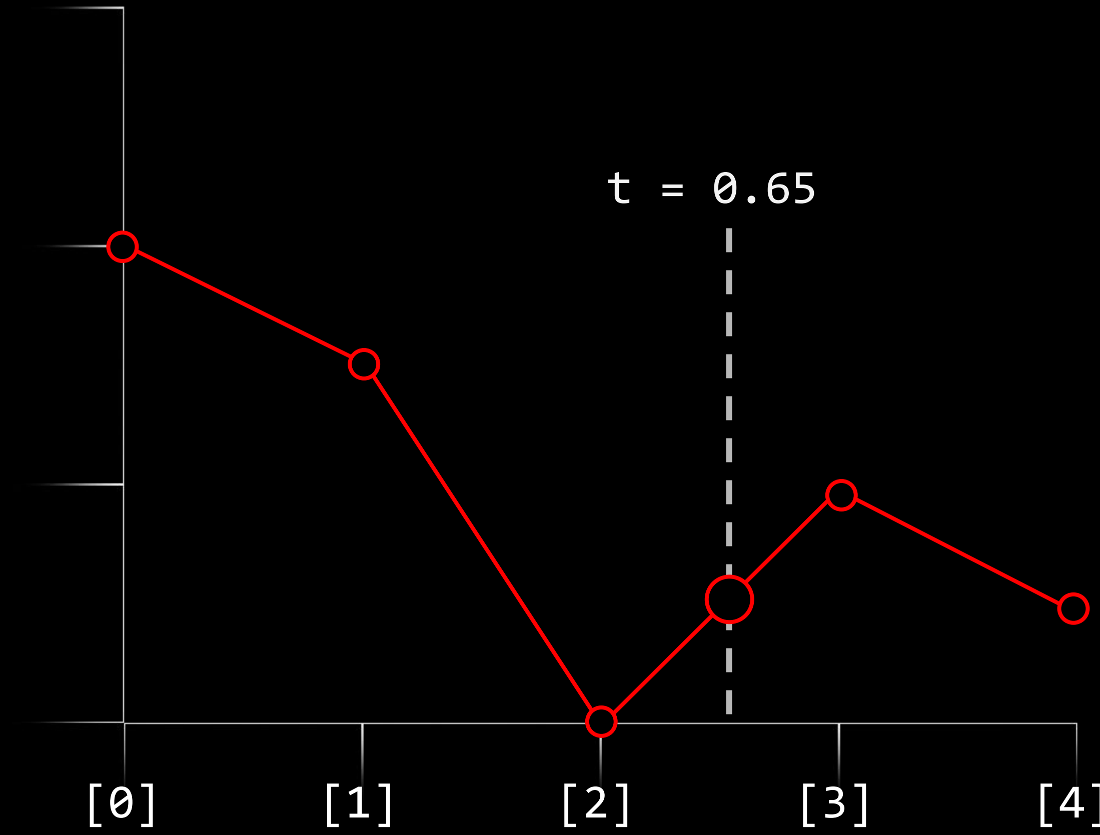
t != Percentage of distance along spline

- t "moves" faster the longer the distance between control points
- We need to compensate for this discrepancy
- Create a look-up-table containing cumulative point distances
- Ex:

```
void CalcLengthTableInto( float[] arr, CubicBezier3D bezier ) {
	arr[0] = 0f;
	float totalLength = 0f;
	Vector3 prev = bezier.p0;
	for( int i = 1; i < arr.Length; i++ ) {
		float t = ( (float)i ) / ( arr.Length - 1 );
		Vector3 pt = bezier.GetPoint( t );
		float diff = ( prev - pt ).magnitude;
		totalLength += diff;
		arr[i] = totalLength;
		prev = pt;
	}
}
```

t != Percentage of distance along spline

- Sample the length array at t
- Make a `Sample()` extension method for `float[]`



```
float[]{   2    ,   1.5    ,   0    ,   1    ,  0.5  }
     t = 0        0.25       0.5       0.75      1
```

t != Percentage of distance along spline

- Sample the length array at t
- Make a `Sample()` extension method for `float[]`

```
public static class FloatArrayExtensions {
      public static float Sample( this float[] fArr, float t){
            int count = fArr.Length;
            if(count == 0){
                  Debug.LogError("Unable to sample array - it has no elements" );
                  return 0;
            }
            if(count == 1)
                  return fArr[0];
            float iFloat = t * (count-1);
            int idLower = Mathf.FloorToInt(iFloat);
            int idUpper = Mathf.FloorToInt(iFloat + 1);
            if( idUpper >= count )
                  return fArr[count-1];
            if( idLower < 0 )
                  return fArr[0];
            return Mathf.Lerp( fArr[idLower], fArr[idUpper], iFloat - idLower);
      }
}
```

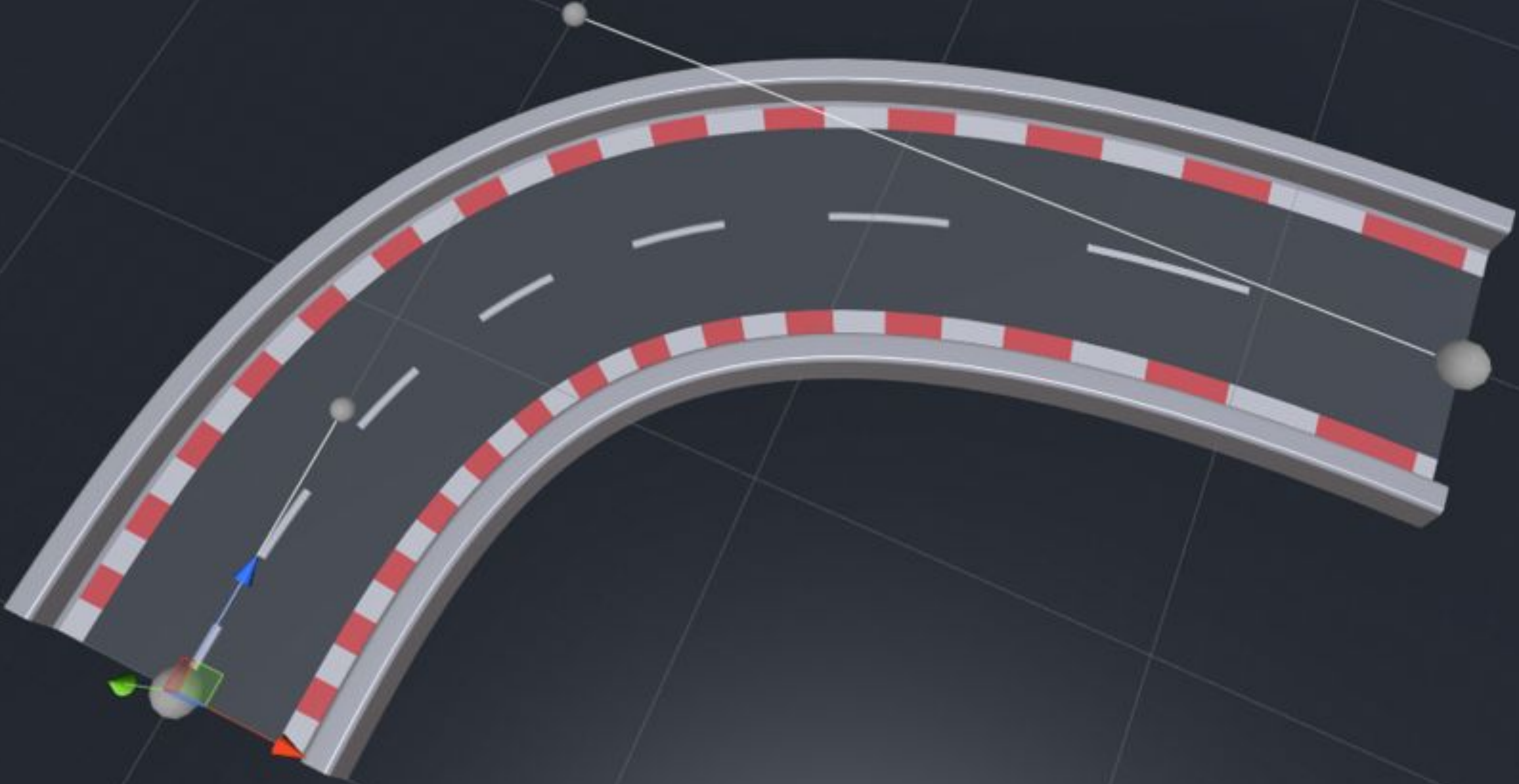
t != Percentage of distance along spline

- Sample the length array at t
- Make a `Sample()` extension method for `float[]`

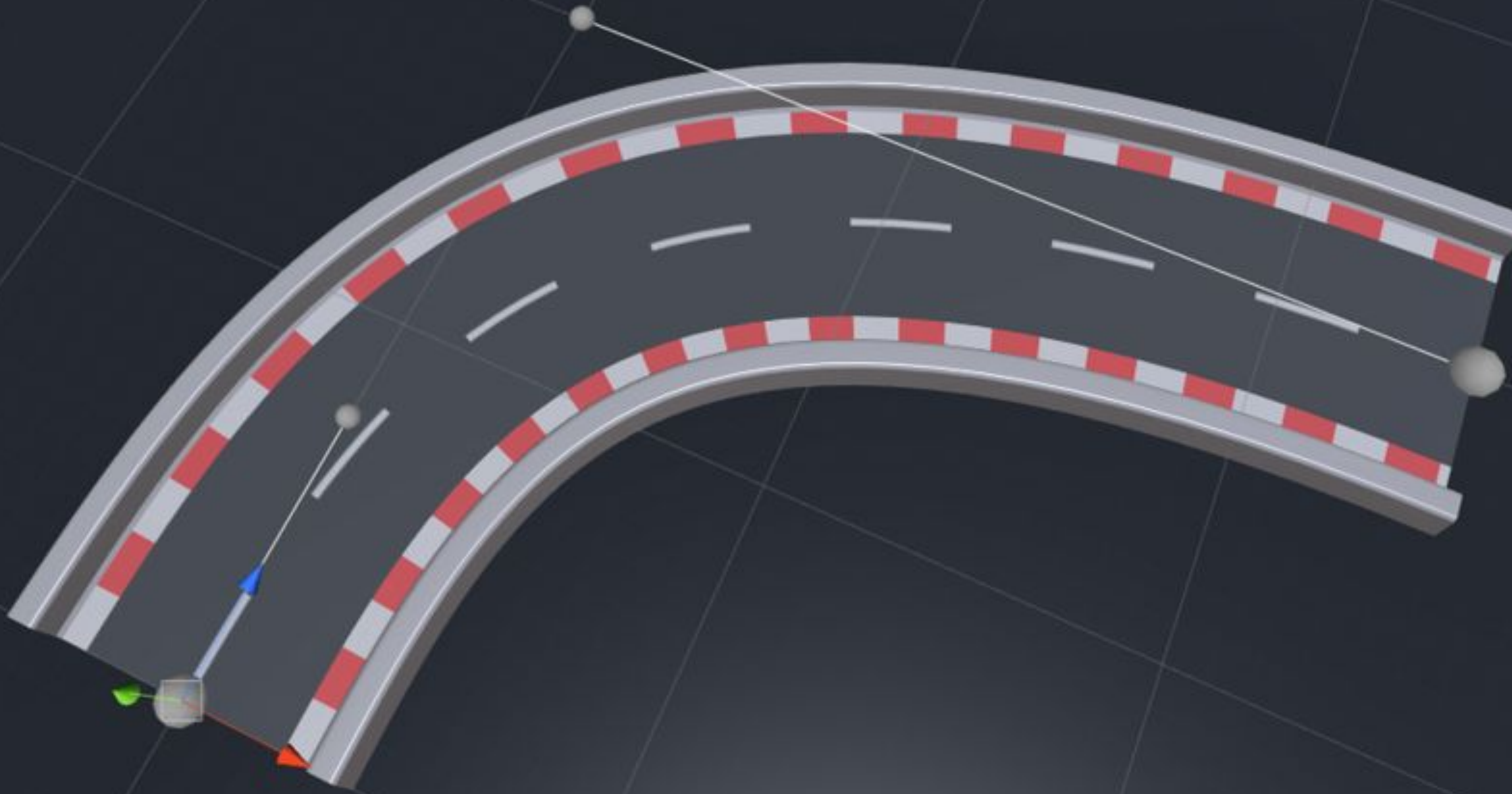
t != Percentage of distance along spline

- Sample the length array at t
- Make a `Sample()` extension method for `float[]`
- Instead of multiplying by length, sample the length array at t

UVs, before



UVs, after



Final notes

- Updating mesh colliders is expensive
- Use separate, simplified and smoothed collision geometry
- Reading mesh.vertices / mesh.triangles / etc. is expensive
- Can easily be modified to use the spline as a deformer, tiling a mesh for things like railroad, fences, rollercoaster tracks, etc.

@ JoachimHolmér
Co-founder & indie developer at
Neat Corporation

Questions?

Thank you!