# Nicole Munro – Geometry Shader paper

## Introduction:

What is a geometry shader? The geometry shader stage in the rendering pipeline is an optional stage that takes primitives as an input and outputs 0 or more primitives. Typical use cases of a geometry shader are text generation, hair/fur generation, water and shadow volume extrusion.

We were given pretty much free reign over what we could do, so I decided to create a roller coaster shader. This shader would take in 4 points and create a cubic Bezier curve out of it, then generate the geometry of a roller coaster. It would look very crude, but the learning process would be in the math I use to a) create a cubic Bezier and therefore the points from which the geometry would ‘spawn’ and b) create an orthogonal co-ordinate system from these points.

## General overview:

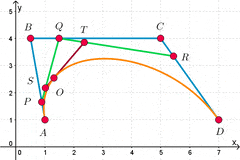
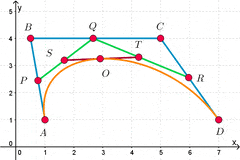
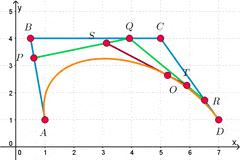
The first thing I needed to do was send 4 points in world space to the geometry shader. I sent these 4 points in as an array of float3’s. In the geometry shader I had to create a cubic Bezier curve from these 4 points. Once these points were created in space I needed to determine the orthogonal co-ordinate system for each point, to create geometry that followed the curve of the Bezier.

Once the co-ordinate system has been created for each point on the curve we could start creating geometry. I started by implementing a simple tube that followed the curve. Once that tube was working I could then try to create two tubes and the support beams between the tubes.  
  
I quickly hit my maximum number of verts. I decided to create an enumeration defining what part of the track the geometry shader would have to create.

## The math:

1.3.2 Cubic Bezier curve:  
Using Wikipedia I found the formula that defines a cubic Bezier curve, which is:

  
To create the Bezier curve, we must define the number of segments that the curve would have. We then loop over the number of segments and in each iteration, we create variable called t – t is the time component in the Bezier equation and it is defined by:

We then calculate the point on the Bezier curve using t and all 4 points sent to the geometry shader with the formula defined above.  


### Orthogonal co-ordinate system:

Now that we have our points of our Bezier curve we need to determine a forward, right and up vector for each point. This is important as we need this relative coordinate system to create our geometry around our point.   
I create this relative co-ordinate system by first creating another point on our Bezier curve just a small amount further away than our first point. With this point we can get the forward vector of the original point by doing a vector subtraction:

Now that we have the forward vector we can determine the Up vector and the Right vector by doing the following operations:  
We define an up vector with y as 1. Then we create our right vector by taking the cross product of the up vector and the forward vector. The final step is to finalize your up vector (which is currently 0,1,0 – which isn’t necessarily the up vector of that point) – we do this by taking the cross product of the forward vector and the right vector. Normalize both the up and right vector.

### Generating vertices

Now that we have everything we need to create vertices around our points, we need to think about what type of shapes we are making. For a tube-like shape we need to know a few things, a radius, the number of circle sides, and the angle you want to increment by in each loop. To calculate the angle we increment by, we use the following formula:

Then we need to create the point on our circle with the following simple formulas:

From there you can create your vertices as you please. Creating boxes is similar, with a width and depth instead of a radius.

## Overlord:

The Overlord Engine is the engine we are going to use to make our shader come to life.   
A few changes we must make to the engine itself would be,

1. Bezier Prefab
2. Bezier Material
3. Material Component Bezier

In our Bezier Scene we create a Bezier Prefab – this prefab holds the Material Component Bezier objects, and the Material Component has the Bezier Material object.   
  
The material component handles all the rendering for the

## References