# Project “Scarlett”

*Developing a tool to generate pseudo random race tracks using Bezier and Catmull rom curves combined with perlin noise*

https://www.youtube.com/watch?v=Sxa0mwhvSQU

https://github.com/Maths-Assignment2/RaceTrack

Table of Contents

# Introduction

This is a tool to generate race tracks from randomly generated points and curves. We use Quadratic Bezier, Cubic Bezier and Catmull rom methods to achieve the desired result. This software will also output a .ply file that allows you to import it into 3D modelling software or an engine like Unreal.

# Leyla Mammadova

Integrated the third order of  Bezier  equation and Catmull-Rom spline formulas for calculating the position of points on the axis that the curve will cross.

Created a ply type files constructor feature that holds data to generate 3D object.

## Formulas

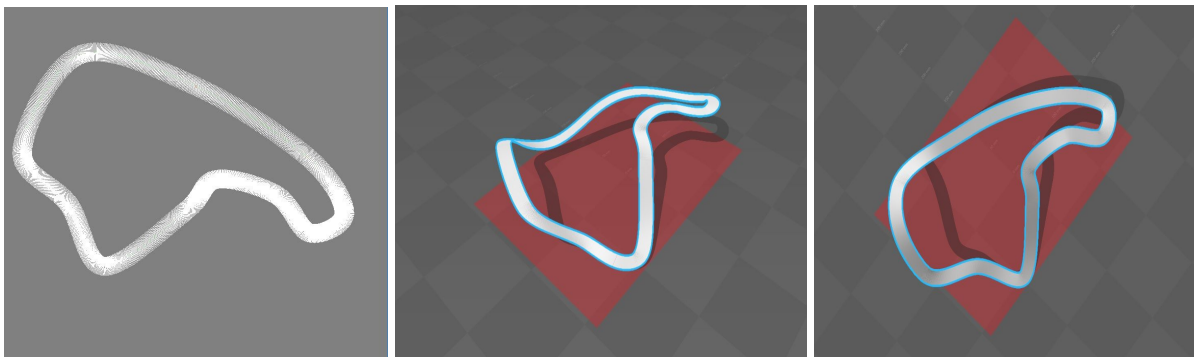
I needed to count cubic Bezier equation that use four control points, creates a curve between first and the fourth end-points, and use two points in the middle as a handle to set the position without crossing them.  The function takes parameter t which value changes between 0 and 1, use this value as a step for number of iteration from first to the last point. For example, at first point the t value equals 0, and on the fourth point it equals 1.

Unlike Bezier curve, Catmull-Rom spline cross all control points and creates smoother spline with less probability of overlapping and random curve angle. Catmull-Rom spline uses the same number of control points as cubic Bezier equation and has t parameter that also known as tension.

## Ply File

To increase scope of our tool that generates race track in OpenGL with C++ and allow it use in other environments like game engine or 3D editor, I added a ply file generating function that stores the vertex buffer and face buffer data and creates a 3D object file from OpenGL drawing.

OpenGL rendered | Ply 3D object side view | Ply 3D object upper view



# Louis Bennette

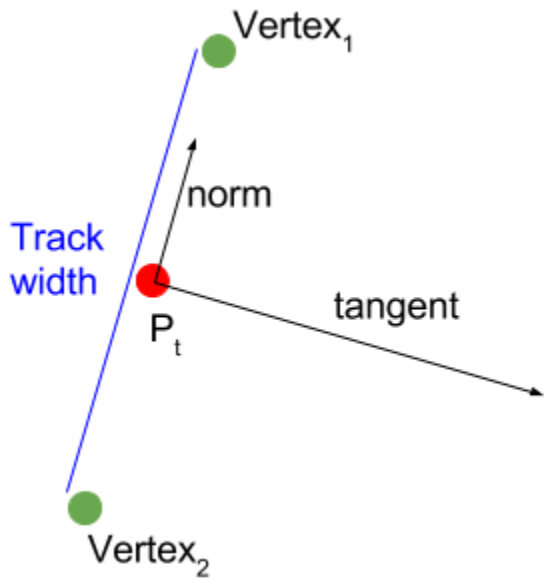
## Mesh generation & rendering:

To keep the tool simple we are rendering our track directly using opengl. The aim of the tool is to create raw mesh data. We can use the data to render a preview of the track in the software and also to output to a ".ply" file. The file io was done by Leyla, I was responsible for the data generation.

We have an ordered list of waypoints that are used to trace a smooth curve. The code steps through the curve in small increments which finds a point in the centre of the track/road. To give the road width I need the tangent of the curve at that point.
There are two ways I've done this. The simplest way and the current implementation is to query a normal point at *t* along the curve and the query a second point at t + 0.01. The vector between these two points is often a very good approximation of the tangent. Then to create width, the normal is needed, the tangent is converted to a normal by performing a cross product with the world up vector. The normal is also normalised so we can control its magnitude later on.

$$\text{normal} = \text{tangent} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

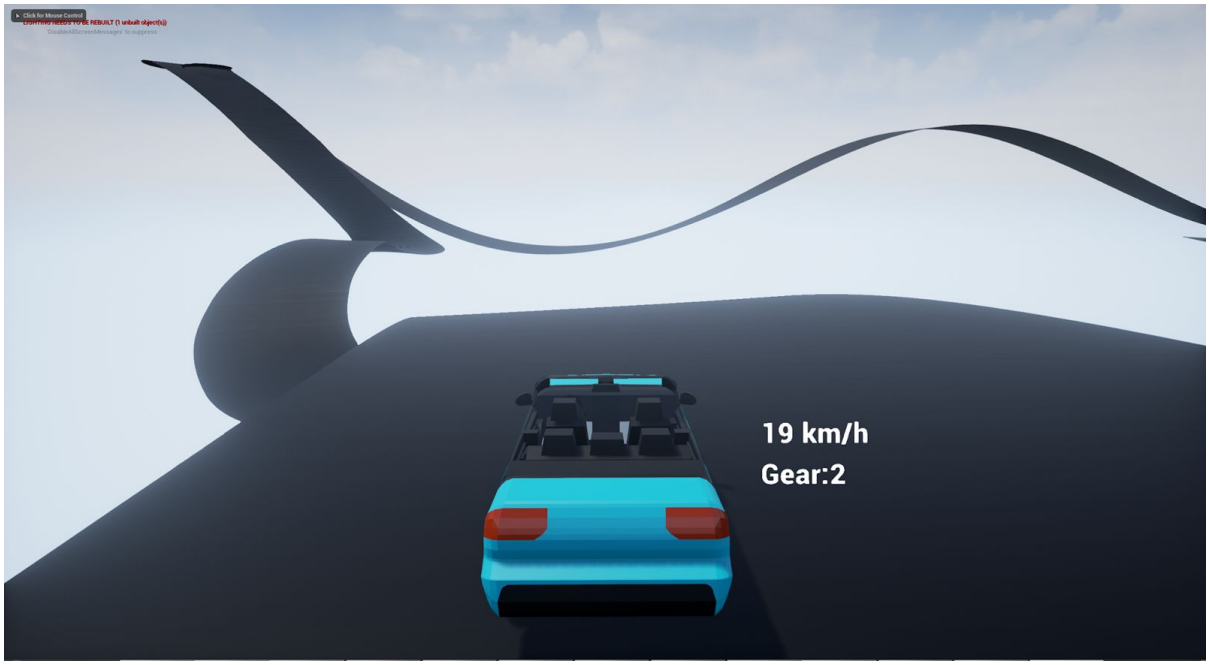
The normal is used to generate two vertices for opposite sides of the track. For every increment of the curve two points are created on either side and should always be the track's width apart.




$$Vertex_1 = P_t + norm \times \frac{1}{2} trackwidth$$
$$\text{Vertex}_2 = P_t - norm \times \tfrac{1}{2} trackwidth$$

As well as the vertex locations, the faces are needed for the .ply file output. The faces of the track are a simple triangle strip. This is generated as the points are generated.
The tool also outputs a top-down view of the track using opengl and supports a debug view where the waypoints and curvature line is generated, as well as a simple wireframe of the mesh for debugging purposes.

Using this generic mesh data we can import our track into any game engine or 3D modeling tool. Previous screenshots show the track loaded into Autodesk Maya, additionally the output has been tested by loading it into the Unreal Engine 4, where a car was added to drive along it (see image bellow and project video).




# Perlin noise:

The perlin noise implemented in this project is a modified header only version of the perlin noise used in Solarian Programmer's C++ implementation of perlin.

Perlin noise allows us to create random heights from a seed, whilst ensuring that nearby points don't differ in height to drastically giving the track a more smooth vertical gradient.

The perlin noise controls the height of the track for every vertex generated. Whenever a curvature point is requested, the x and y coordinates of the curvature point are used to look up a perlin value. The perlin return value is scaled based on a user defined amplitude and used as the height value of the vertices for that curvature point.

# Abdullah Bin Abdullah

The racetrack is generated from a pseudo random points generator. It uses the vec3 data structure from octet and a rand() function that is constrained from -1 to 1 in the x and y axes. This is so that the race track is kept in view. After storing the points into a vector the points are then sorted by closest distance to each other and then averaged into a new vector. This helps create an order for the curves formula and space out the points to help alleviate generating sharp corners or overlapping tracks.

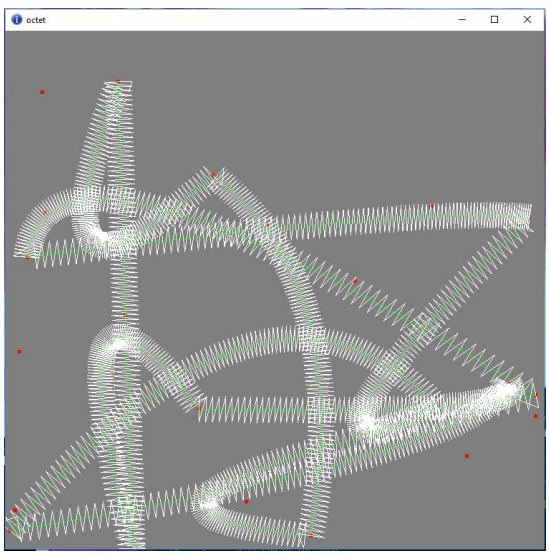
## Generating points

The random number generator returns a float and is constrained from -1 to 1. The random number generator creates a random float with a max random number and then multiplied by the difference between the second range and then adds it to the minimum.

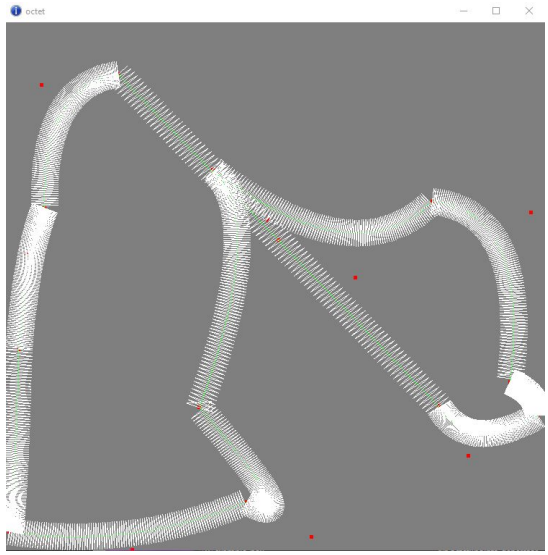
## Sorting the waypoints

The points are then sorted by closest distance to each other. Using two vectors, a and b, we take the first point from vector a and store it to vector b. This will be used to compare distances between the first point of vector b to points in vector a. The distance between points is calculated by finding the difference between the two points and then finding the magnitude (Dunn and Parberry, 2011). As the point with the shortest distance to the point in vector b is found, the point from vector a is then stored into vector b. This orders the list so that the race track will create curves in order. The following images shows how sorted and unsorted points will affect the race track:

Unsorted Points



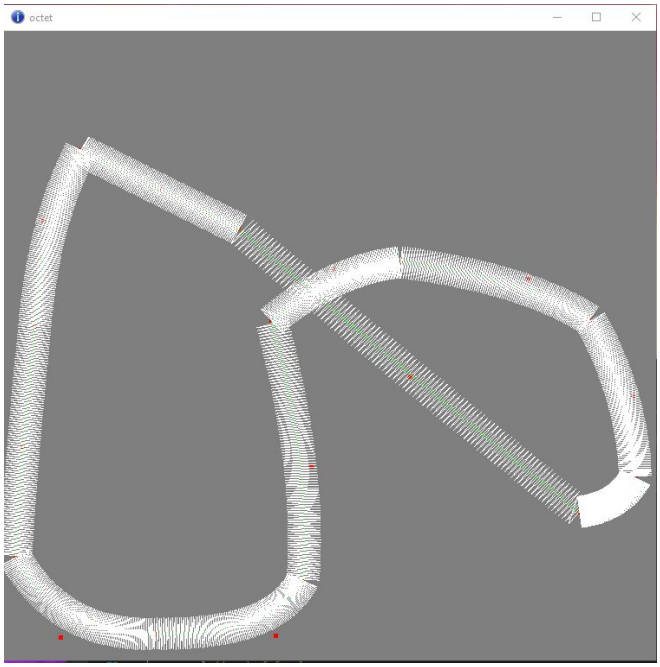

Sorted Points




## Averaging the waypoints

Finally, the waypoints are taken from the sorted vector and the average of three points are calculated and then stored into the original vector. This is to make sure that the distance between

points isn't too close. The following image shows the race track when the averaged points are used:




# Conclusion

More functionality can be added to this tool. Currently the tool is 100% randomised. This tool would benefit greatly from a user interface with fields buttons and sliders to craft aspects of the track. The user ideally would be able to place points and link them together before the curve generation.

Additionally, the random seed system needs to be developed further allowing the user to make width height amplitude and other operations without re randomising the waypoints.

This tool has helped us understand mathematical curvature but also it has helped us develop an understanding of creating tools that are more usable in industry.

# Bibliography

Dunn, F. and Parberry, I. (2011). *3D math primer for graphics and game development*. 2nd ed. Boca Raton, FL: A K Peters/CRC Press.

*OpenGL programming/GLStart/Tut3 - Wikibooks, open books for an open world* (no date) Available at: https://en.wikibooks.org/wiki/OpenGL_Programming/GLStart/Tut3 (Accessed: 27 January 2017).

Pomax (2015). *A Primer on Bézier Curves.* Available at: http://pomax.github.io/bezierinfo/#flattening. (Accessed 05 Feb 2017).

solarianprogrammer (2017) *Perlin noise in C++11*. Available at: http://solarianprogrammer.com/2012/07/18/perlin-noise-cpp-11/ (Accessed: 2 February 2017).

# Appendix A: Keybindings

| Command | Key |
|---|---|
| Switch to QUADRATIC_BEZIER | F1 |
| Switch to CUBIC_BEZIER | F2 |
| Switch to CATMULL_ROM | F3 |
| Randomise Track | F5 |
| Save Track | F6 |
| Toggle Debug mode | Space bar |
| Increase Track Length | Up Arrow |
| Decrease Track Length | Down Arrow |
| Increase Track Width | Right Arrow |
| Decrease Track Width | Left Arrow |
| Increase Height Amplitude | F8 |
| Decrease Height Amplitude | F7 |
| Increase Mesh Detail | F10 |
| Decrease Mesh Detail | F9 |