Faculty of Science and Technology

BSc (Hons) Games Software Engineering

May 2024

Roller Coaster Modelling

by

Daniel Hance

Signed: Daniel Hance

Name: Daniel Hance

Date: 16/05/24

Program: BSc GSE

Permission for use of produced work

We ask you to sign this form because Bournemouth University Higher Education Corporation (**BU**) may wish to use your Individual Project material (in its original or an amended form) for marketing purposes in one or more of the following ways:

- prospectuses and other BU promotional materials, including promotional videos
- as part of an advert or advertisement feature (which includes possible use on outdoor media such as buses and billboards)
- on the BU websites or intranet
- at open days
- in social media

If we used it, you would receive exposure for your work. You do not have to agree, and not doing so will have no effect on marking your Individual Project material. For ease this is simply called Material below.

If you are kind enough to agree to BU having rights of use and editing, this will not affect your ownership rights in the Material or your right to use it for your own professional or personal reasons. Unless it is impractical to do so – for example a montage on social media where space limits may apply, BU will ensure that it credits you for BU's use of the Material.

By signing this form:

- I grant a non-exclusive, world-wide, royalty-free licence to BU to use and sub-licence the Material, for marketing purposes; and I waive any non-property rights (including moral rights) now or in the future existing. This licence includes the right to edit and create derivative works from the Material, for example, a multi-student showreel.

- I agree this licence shall subsist permanently unless I seek to end it in line with the procedure below; but that where BU has already used, or is committed to using, the Material for a particular marketing campaign - for example, in a prospectus already printed, any ending of the licence shall relate only to new uses of the Material.

- **I understand BU will not ask me for any further approval or permission for specific uses of the Material for marketing purposes; but that I can withdraw my permission at any time, with the consequences for continued use set out above.**

If you wish to withdraw your permission for use of your information, please contact the Programme Support Officer by e-mail scitechcreativepso@bournemouth.ac.uk. Otherwise, we will keep your Material and associated personal information for as long as we have any plan to, or actively, use the Material for marketing purposes.

The legal basis for BU processing your personal information is consent, that is by signing this form you are giving permission to use your personal information provided in, and in relation to, the Material for the purposes set out above.

In processing your personal data, BU complies with the Data Protection Act 2018 and General Data Protection Regulation (Regulation (EU) 2016/679). You can find further information about BU's data protection and privacy approach here: https://www.bournemouth.ac.uk/about/governance/access-information/data-protection-privacy.

Further information about your rights under the data protection legislation can be found from our Data Protection Officer on dpo@bournemouth.ac.uk.

I give permission for BU to use my Material on the basis set out above:

Signed: Daniel Hance

Name: Daniel Hance

Date: 16/05/24

Abstract

There are many games and CAD software that allows users to design their own three-dimensional roller coasters. As a fan of such games, I found the challenge of creating a roller coaster modeling system with a focus on visual appearance within my own game engine intriguing. The goal of the project was to develop a spline system that could represent a roller coaster track and give users the control to design their own layouts, ultimately generating a roller coaster model along this spline. The project successfully achieved these goals, resulting in a complex spline system with extensive user control and a fast model generation system capable of using multiple template models. Furthermore, the project exceeded initial expectations, as the program can also be used to create train tracks and roads.

Table of Contents

Introduction/Context

Roller coasters (referred to as RCs going forwards) have been thrilling riders since the inception of the Switchback Railway at Coney Island in 1884, Hunt, 2018. Over the years, the desire to create personalized RCs has led to the development of various video games and computer-aided design (CAD) software. These tools cater to enthusiasts seeking to design their own thrilling rides at different levels of realism and accessibility, from the beloved classics like the isometric Roller Coaster Tycoon series, to more advanced tools like No Limits 2.

Fig 1: Roller Coaster Tycoon 2 Gameplay

When examining the features that directly relate to RCs in these programs it becomes clear that they primarily focus on two distinct aspects of RCs: the visual representation of the ride's structure and the dynamic motion of its trains travelling around the track. This project aims to create a program that allows users to model their own three-dimensional RCs, the key focuses being on the visual aspect of RCs and less so on the dynamic simulation of them.

Background Research/Related Work

The initial stage of the research aimed to understand how to construct a 3D model of an RC, this can be achieved by creating a centreline of the RC track using a sequence of coordinates. These coordinates should be defined by a dimensional spline, as outlined by He in 2015.

With this in mind the research objectives for this project are:

- Research Bezier curves to gain an understanding of how splines work.
- Research and find an appropriate spline that can be used to represent the RC centreline.
- Research and find a method for creating models along this centreline.
- Research and find a method for modelling adaptive RC supports.
- Find a basic physics algorithm that can be used to animate a model along the centreline.

Bezier Curves

Before starting detailed research into splines, it is essential to understand the concept of Bezier curves. As they are closely linked with splines and serve as the foundation for how splines are calculated (many spline resources start by discussing Bezier curves e.g. An introduction to NURBS: with a historical perspective, Rogers 2001). Bezier curves use a weighted sum, where the weights are Bernstein polynomials, to generate smooth curves between a number of control points. When discussing both Bezier curves and splines the degree of the curve refers to the number of control points that were used in the generation of the curve as explained in Bezier curves (University of Cambridge 1999).

$$\begin{array}{rl} \text{linear} & \mathbf{P}(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \\ \text{quadratic} & \mathbf{P}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \\ \text{cubic} & \mathbf{P}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \end{array}$$

Fig 2: Second-, third- and fourth-degree Bezier curve definitions, Bezier curves (University of Cambridge 1999)

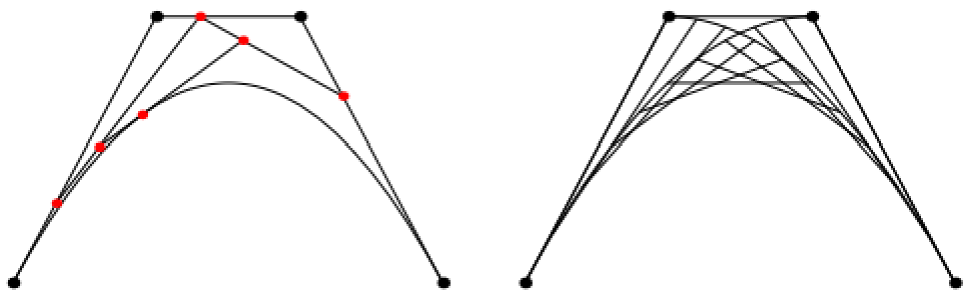


Fig 3: Fourth-degree Bezier curve (Lyche, 2008, p.16)

While initially appearing suitable for creating the centerline for a RC, Bezier curves encounter two significant challenges for this task. The first issue, as described by Holmér in “The Continuity of Splines” (2022), is a lack of local control. Bezier curves use a single weighted sum for the entire curve, meaning that adjusting one control point can alter the entire curve. This limitation makes working with high-degree Bezier curves challenging.

Although this problem can be mitigated by connecting multiple Bezier curves together instead of using a single large curve, it introduces a second problem: continuity. Continuity is a system used to describe how multiple curves connect, the levels of continuity as described in Spline Curves (University of Clemson, p.94) are:

- C^0 continuity, meaning that the two segments match values at the joint (position continuity).
- C^1 continuity, meaning that they match slopes at the joint (velocity continuity).
- C^2 continuity, meaning that they match curvatures at the joint (acceleration continuity).
- For a curve to meet a level of continuity it must also achieve all lower levels of continuity.

As a RC train will need the acceleration vector of the curve to be continuous at the joint to prevent sudden changes in speed the centerline of the track will need to be C^2 continuous. To achieve C^2 continuity with joined Bezier curves, control points in the second curve must be set to specific values to match the curvature between the Beziers, once again resulting in the loss of local control, Bezier curves (University of Cambridge 1999).

$$\mathbf{Q}_2 - 2\mathbf{Q}_1 + \mathbf{Q}_0 = \mathbf{P}_n - 2\mathbf{P}_{n-1} + \mathbf{P}_{n-2}$$

Fig 4: The requirements for C^2 continuity between Bezier curves P(t) defined by (P_0, P_1, …, P_n), and Q(t) defined by (Q_0, Q_1 …, Q_n). Bezier curves (University of Cambridge 1999)

This loss of local control renders Bezier curves inefficient for creating the RC centerline, this leads into the exploration of splines for a solution to this problem.

B-Splines

Splines represent a more sophisticated approach to curve generation compared to Bezier curves, however there are many types of splines which each has unique characteristics and use cases, Prenter 2008. In the context of modeling RC tracks, research in this area has concluded that cubic splines or shape-preserving splines are suitable for describing the centerline of a fully three-dimensional RC track (as proven by Pombo, 2007, p.498).

When this is combined with the continuity requirement outlined in the Bezier curve research, the spline that is selected to represent the RC's centerline must also achieve C^2 continuity. These two requirements lead into the cubic B-spline, this spline can guarantee C^2 continuity across the entire curve as long as multiple control points aren't overlapping as stated in B-splines (University of Cambridge 1999).

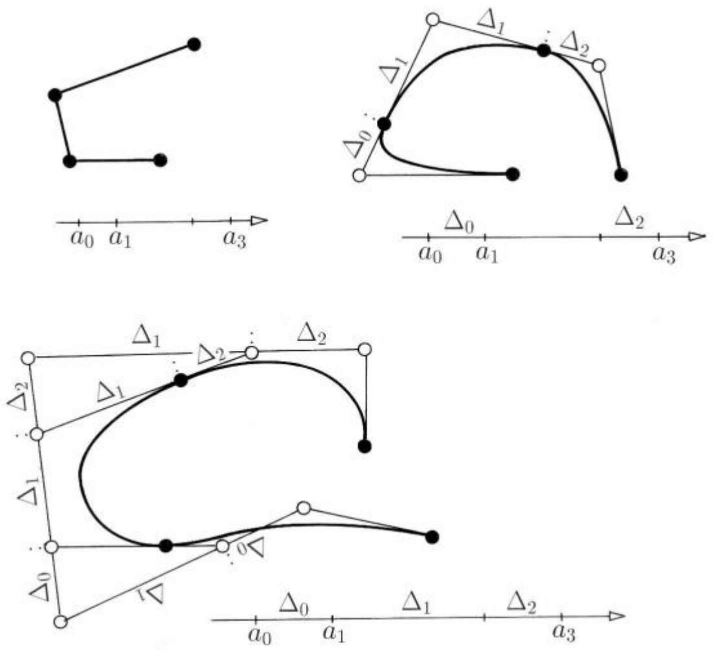


Fig 5: B-splines with a degree of 1, 2 and 3 (Prautzsch 2002, p.61)

When discussing B-splines, the term "order" (linear, quadratic, cubic) refers to their degree minus one. For example, a B-spline that uses four control points for each curve section (degree 4) would have an order of 3 (cubic). In a B-spline curve each control point is associated with a basis function (as seen in Fig.6). The significance of this is that the order of the generated curve is no longer fixed to the number of control points used to describe the curve, as it is for Bezier curves.

$$\mathbf{P}(t) = \sum_{i=0}^{n} \mathbf{P}_i N_{i,k}(t)$$

Fig 6: Definition of the B-spline curve P(t) piecewise continuous function (Joy 2008, p.1)

As depicted in Fig. 5, this characteristic simplifies working with high-order B-splines, as each control point projects control over a small amount of the total curve, keeping local control. To simplify this process, a B-spline creates many small curves, each with a unique function that are connected in a way that creates C^{k-1} continuity across the whole spline. A non-decreasing sequence of real numbers called a knot vector is used to determine the values of t where the pieces of curve join, this knot vector must be (t_1, t_2, …, $t_{k+(n+1)}$), B-splines (University of Cambridge 1999).

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } u \in [t_i, t_{i+1}) \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,k}(t) = \left(\frac{t - t_i}{t_{i+k-1} - t_i} \right) N_{i,k-1}(t) + \left(\frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \right) N_{i+1,k-1}(t)$$

Fig 7: $N_{i,k}$ defined recursively as used in Fig.6 (Joy 2008, p.2)

The value of $N_{i,k}$(t) is determined solely by the spline order and knot vector, this calculation not considering control point numbers is how the spline keeps local control. $N_{i,k}$(t) essentially preforms a series of modified linear interpolations, cascading down to a depth equivalent to the spline's order. As depicted in Fig.8, this leads to splines with a higher order preforming more layers of linear interpolation, leading to a smoother curve at the cost of single control point influence. This is why the B-spline implemented for the RC's centerline will be cubic as this provides the highest level of control point influence while still being C^2 continuous.

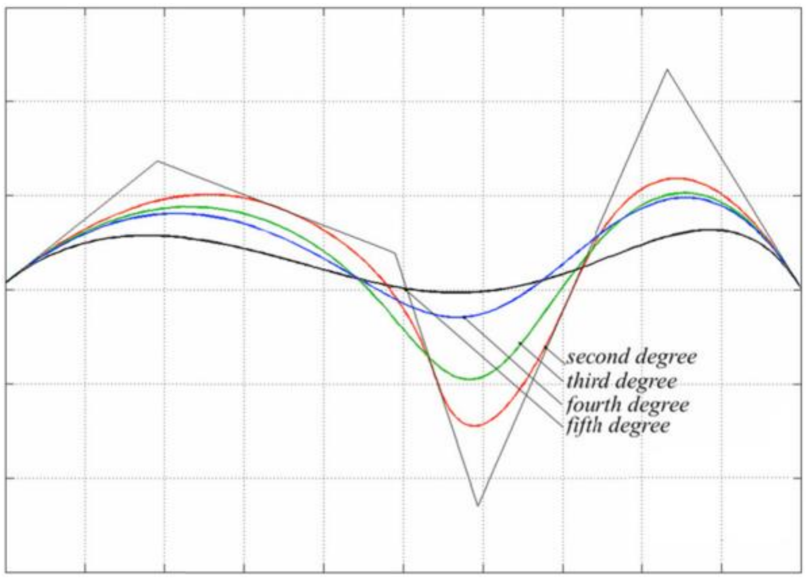


Fig.8: Curve smoothness using a range of clamped B-spline curve degrees (Elbanhawi 2015, p.8)

As depicted in Fig.8, the B-spline shown is clamped, this means that the spline curve directly intersects with the first and last control points. This clamping method is also referred to as open-uniform. To achieve a clamped B-spline, the formula in Fig.9 can be use when generating a knot vector, a clamped knot vector for a cubic B-spline could look like [0,0,0,0,1,2,3,4,4,4,4].

$$\begin{aligned} t_i &= t_1, & i &\le k \\ t_{i+1} - t_i &= \textbf{constant}, & k &\le i < n+2 \\ t_i &= t_{k+(n+1)}, & i &\ge n+2 \end{aligned}$$

*Fig.9: Open uniform knot vectors with k equal knot values at each end (*B-splines, University of Cambridge 1999)

Another configuration for B-splines is unclamped, this means the spline curve will not intersect with the first and last control point and they will be treated like the other control points. This unclamping method is also referred to as uniform. To achieve an unclamped B-spline, the formula in Fig.10 can be used when generating the knot vector, an unclamped knot vector for a cubic spline could look like [1,2,3,4,5,6,7,8,9,10,11].

$$t_{i+1} - t_i = \textbf{constant}, \forall i$$

*Fig.10: Uniform knot (*B-splines, University of Cambridge 1999)

There is a third-type of B-spline known as non-uniform, where the values in the knot vector do not increase by a constant value (e.g. [0,0.25,1,1.5,3]). According to B-splines by University of Cambridge (1999), non-uniform B-spline shouldn't be used unless there is a specific reason for doing so, this is because they can be challenging to work with. Since uniform and open-uniform splines already possess all the necessary properties of an RC's centerline, there is no justification for further exploration of non-uniform splines for this project.

The final aspect of exploring the B-spline involves finding a derivative basis function. While the basis function shown in Fig.6/7 is effective for finding a point on a B-spline curve, the first derivative of this basis function will return the tangent vector of this point, as depicted in Fig.11. The tangent vector is essential when it comes to creating the RC mesh along the spline curve (this is expanded upon in the procedural geometry research). The same knot vector used in the original basis function should also be used in the derivative basis function to produce the correct results Piegl 1997.

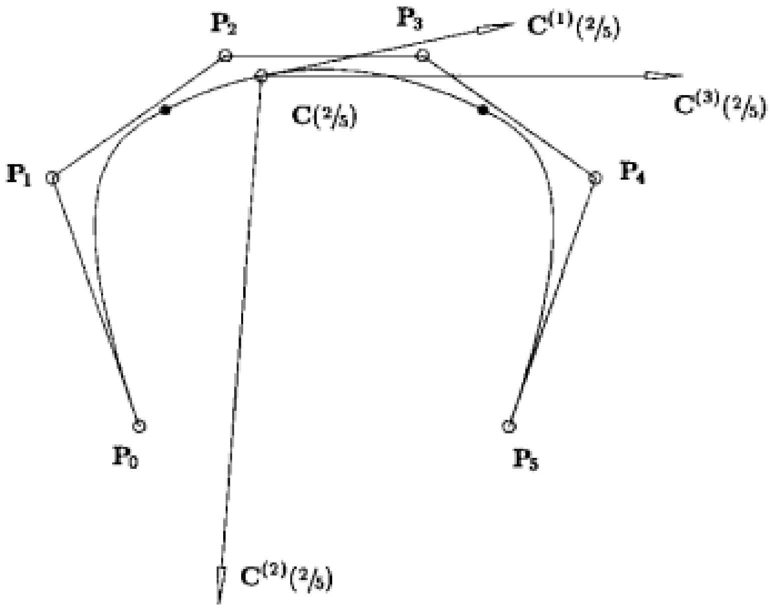


Fig.11: cubic B-spline curve on T = {0,0,0,0, 1/4,3/4,1,1,1,1} with the first, second and third derivatives computed at t = 2/5 (Piegl 1997, p.101)

The derivative basis function depicted in Fig.12 can compute derivative values along the spline curve using $N_{i,k}$ values form the original basis function shown in Fig.7. By utilizing this function, the tangent at any point along the spline curve can be determined. With this all the necessary information needed from the spline curve can be collected, this leaves only a few additional calculations needed for getting the RC mesh's positions which will be discussed in the procedural geometry research.

$$\mathbf{C}^{(k)}(u) = \sum_{i=0}^{n-k} N_{i,p-k}(u)\mathbf{P}_i^{(k)}$$

$$\mathbf{P}_i^{(k)} = \begin{cases} \mathbf{P}_i & k = 0 \\ \dfrac{p-k+1}{u_{i+p+1} - u_{i+k}} \left(\mathbf{P}_{i+1}^{(k-1)} - \mathbf{P}_i^{(k-1)}\right) & k > 0 \end{cases}$$

Fig.12: Basis derivative function for a B-spline curve, $N_{i,p-k}(u)$ is defined recursively as $N_{i,k}(t)$ in Fig.7 (Piegl 1997, p.97)

When considering all of the above research on B-splines, it can be concluded that a cubic B-spline will be appropriate for defining the RC tracks center line within the project. A cubic B-spline will provide the highest level of control point influence on the curve while remaining C^2 continuous. The spline should be able to switch between uniform and open-uniforms as having the option to change if the spline intersects the first and last control point will be useful to have in the implementation.

Procedural Geometry

The procedural geometry section of this projects needs to be able to modify a pre-existing model mesh and move it so the model is mapped along the spline acting as the RC centerline. As discussed by Holmér in Unite 2015 Boston, the first challenge to achieve this goal is to find the three local axis along the spline path, as shown in Fig.13. The z axis is already calculated using the spline points first derivative as discussed in the B-spline research so only the x and y need to be calculated.

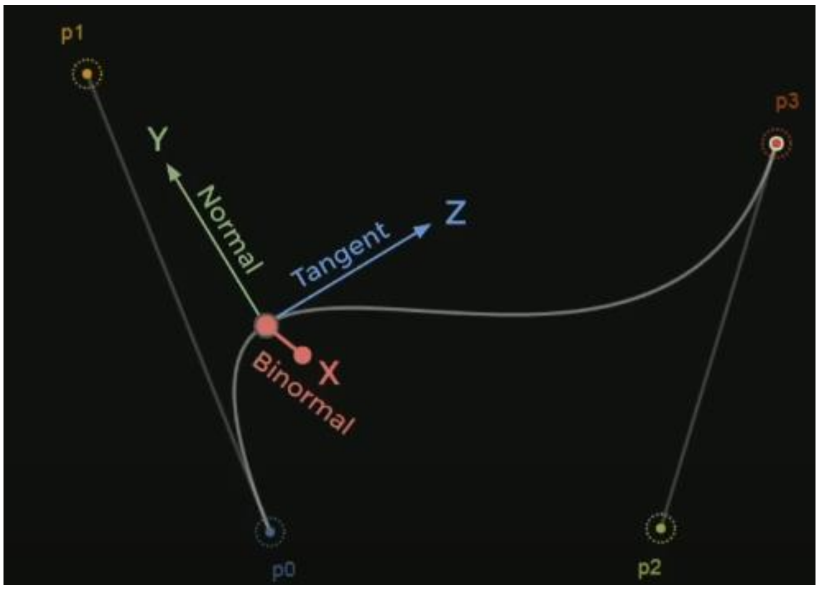
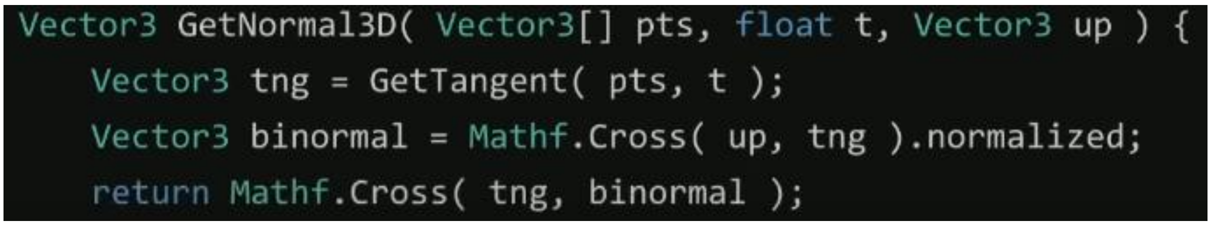


Fig.13: Spline point local axis (Holmer 2015)

As further discussed by Holmer 2015, to find the y axis (normal) of the spline point you need to use a reference up vector and find the cross product between this and the tangent vector. This formula means that on a flat spline with no change in height the roller coaster model will always face up, while if the height of the spline is changed the y axis direction will also change. Although the example code in Fig.14, is written in C# within the unity engine what matters is the theory as this can be re-written in C++ in Red Engine for this project's implementation. The x axis (binormal) can then be calculated by finding the cross product between the tangent and the normal vectors as explained in finding the normal to a plane (University of Texis 2016).



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

Fig.14: Code for finding the normal of a point on a spline curve within the unity game engine (Holmer 2015)

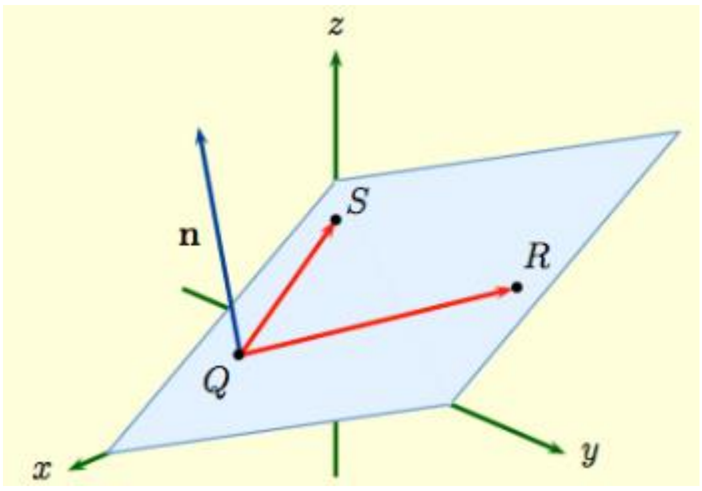
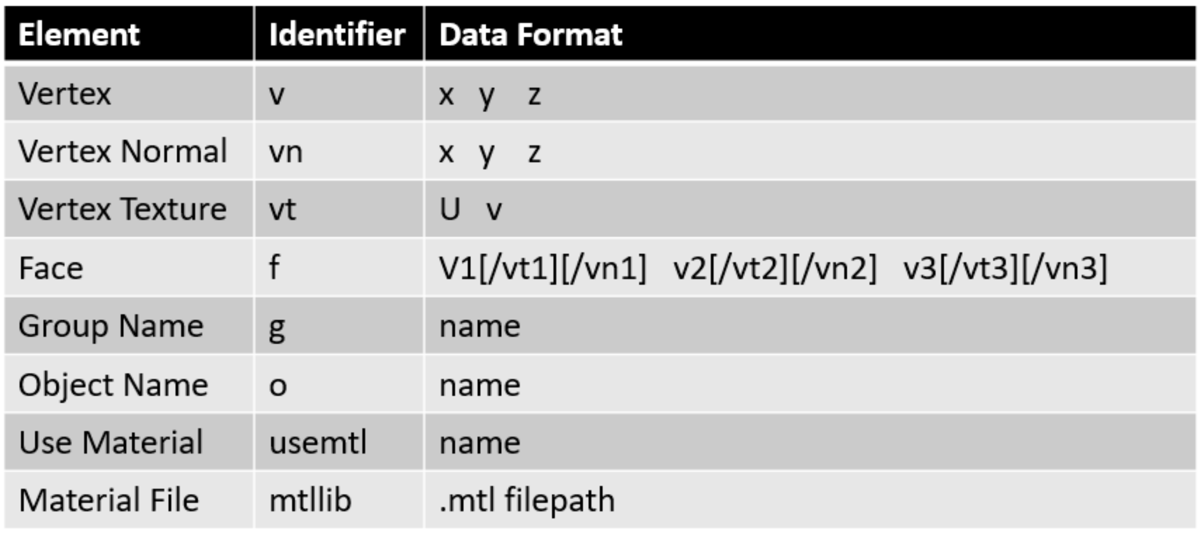


Fig.15: Diagram showing how to use the cross product to find the normal of a plane, this method can be used to find the final spline point axis (Finding a normal to a plane, University of Texas 2016).

Models are created by listing vertices and connecting them to form faces as shown in the Blender 2.80 manual (2024). To map a model to a spline curve, the model's vertices can be moved around the spline, this will involve assigning each model's axis to a spline axis and performing some basic vector math. This will use formulas found in "Algebra and trigonometry" Zill (2011) to correctly position the vertices.

There are few resources that explain how to do this in detail, as most guide/books use existing software like unity or unreal without explaining the underlying concepts. Therefore, the research for this part of the procedural geometry will occur during the implementation phase. This approach was chosen because it will be easier to develop an algorithm to move the vertices accurately once a spline system is in place, allowing for first-hand experimentation with mapping models.

The object file type that will be used for the procedural geometry will be the wavefront OBJ format. This was selected as while there are many different file types for 3D objects the .obj format is widely adoptive by most major software and the format is easy to work with (sustainability of Digital Formats: Planning for Library of Congress Collections 2020). The .obj uses keywords at the start of lines to say what that line stores, the keywords that are relevant to this project are shown in fig.16.



Element	Identifier	Data Format
Vertex	v	x y z
Vertex Normal	vn	x y z
Vertex Texture	vt	U v
Face	f	V1[/vt1][/vn1] v2[/vt2][/vn2] v3[/vt3][/vn3]
Group Name	g	name
Object Name	o	name
Use Material	usemtl	name
Material File	mtllib	.mtl filepath

Fig.16: Wavefront file format keywords and data format (Bournemouth University, no date)

Roller Coaster Dynamics

As can be seen in papers like "Modeling track for roller coaster dynamics" by Pombo 2007, a full dynamic simulation of a RC train is a large amount of work, adding this to the project would double the total work needed and drastically add to the amount of research that would need to be performed. It is unrealistic to complete all of this within the planned timeline for this project so a much simpler physics algorithm will be used to provide movement to the trains. This is acceptable as, as stated at the start of the research, the main focus of this project is the modeling and appearance of RCs and not the dynamic aspect. As the project is being created using object-oriented programming the option for a full dynamic simulation implementation to be swapped in later will be available.

As the initial physics implementation is going to act as more of a stand-in, an algorithm such as Fig.17 should be sufficient, the main factor in selecting an algorithm is the predicted implementation time and not how accurately it simulates RC physics.

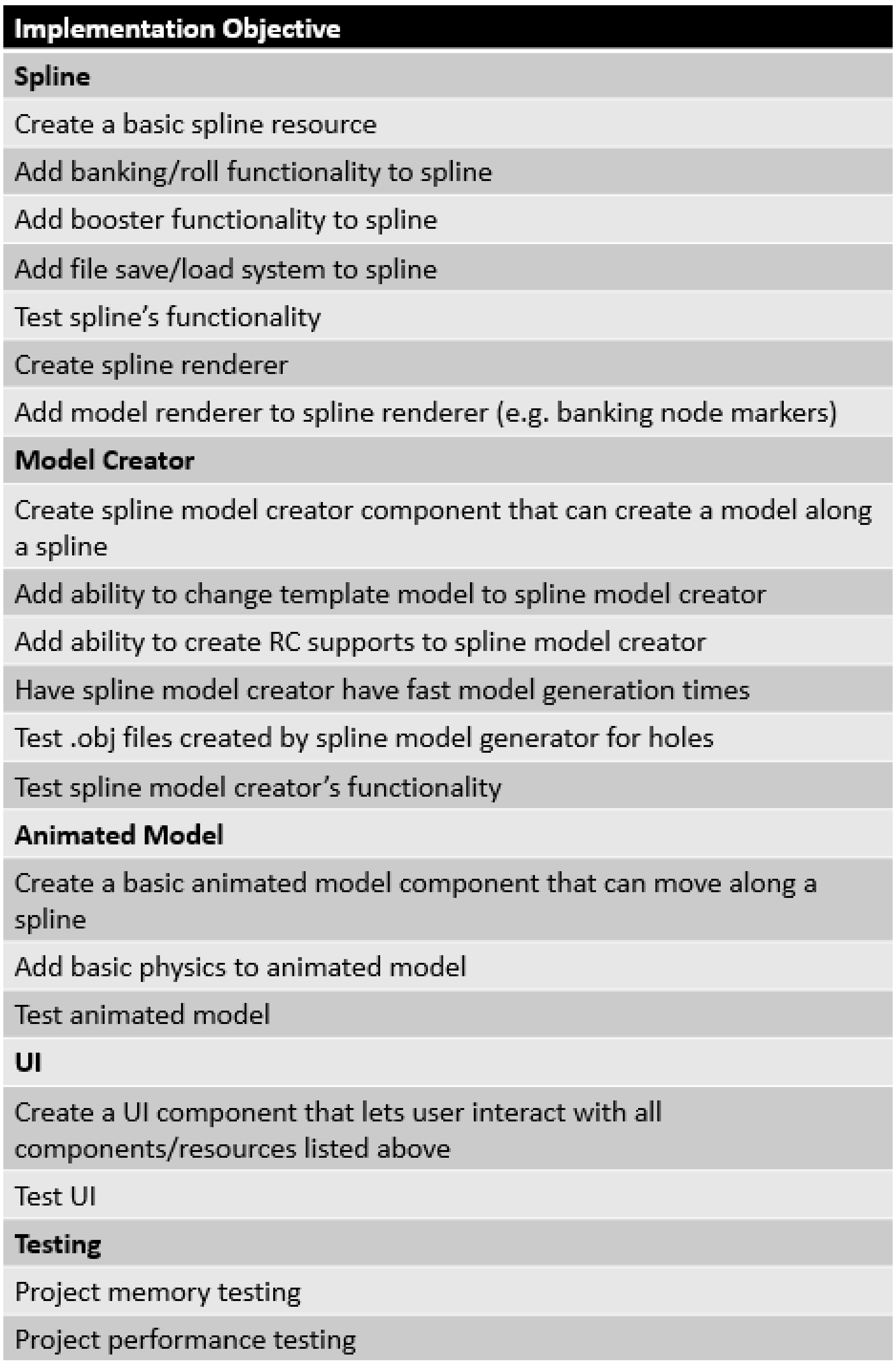
$$a = \frac{-g\,k}{\sqrt{(1+k^2)}} - {}^{b}\!/_{m}\, v$$

Fig 17: Basic physics formula for acceleration on a slope where k is the slope (Roller Coaster, myPhysicsLab 2023)

Design and Implementation

Objectives

The objectives for the implementation phase of this project are listed below in fig.18.

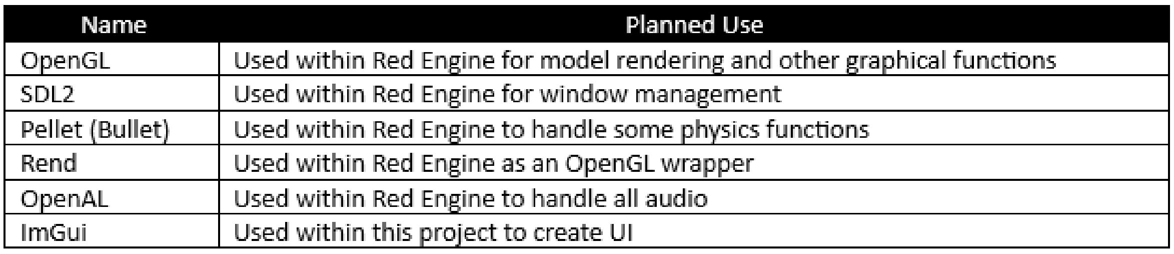


Implementation Objective
Spline
Create a basic spline resource
Add banking/roll functionality to spline
Add booster functionality to spline
Add file save/load system to spline
Test spline's functionality
Create spline renderer
Add model renderer to spline renderer (e.g. banking node markers)
Model Creator
Create spline model creator component that can create a model along a spline
Add ability to change template model to spline model creator
Add ability to create RC supports to spline model creator
Have spline model creator have fast model generation times
Test .obj files created by spline model generator for holes
Test spline model creator's functionality
Animated Model
Create a basic animated model component that can move along a spline
Add basic physics to animated model
Test animated model
UI
Create a UI component that lets user interact with all components/resources listed above
Test UI
Testing
Project memory testing
Project performance testing

Fig.18: Project's implementation objectives

Dependencies

All the dependencies for this project are listed below in fig.19, all the libraries used are c++ based which will lead to improved performance.



Name	Planned Use
OpenGL	Used within Red Engine for model rendering and other graphical functions
SDL2	Used within Red Engine for window management
Pellet (Bullet)	Used within Red Engine to handle some physics functions
Rend	Used within Red Engine as an OpenGL wrapper
OpenAL	Used within Red Engine to handle all audio
ImGui	Used within this project to create UI

Fig.19: Projects dependencies libraries

Target Platform

The projects target platform is windows 10 and 11, development of the project will be using a windows 10 laptop and a windows 11 desktop so this will be continuously tested throughout development.

Source Control

Git will be used as source control on this project.

Red Engine

This project was developed in a custom-made simple 3D game engine created for a previous assignment that uses an entity component system architecture (ECS) and an OpenGL wrapper library REND to handle rendering which was modified to suit the engine's needs. The engine already has feature such as: a resource management system (with caching), a collision detection and response system, 3D rendering and shader systems, input management systems.

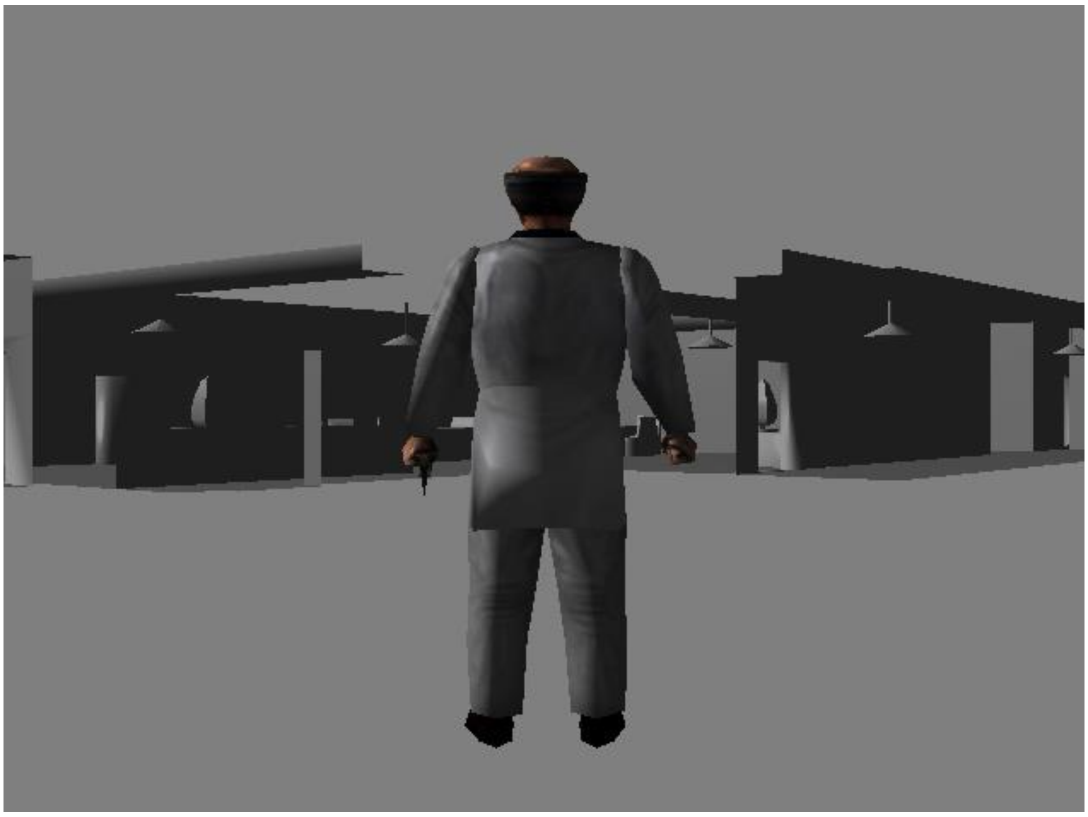


Fig.20: Output of original Red Engine before any upgrades/modifications

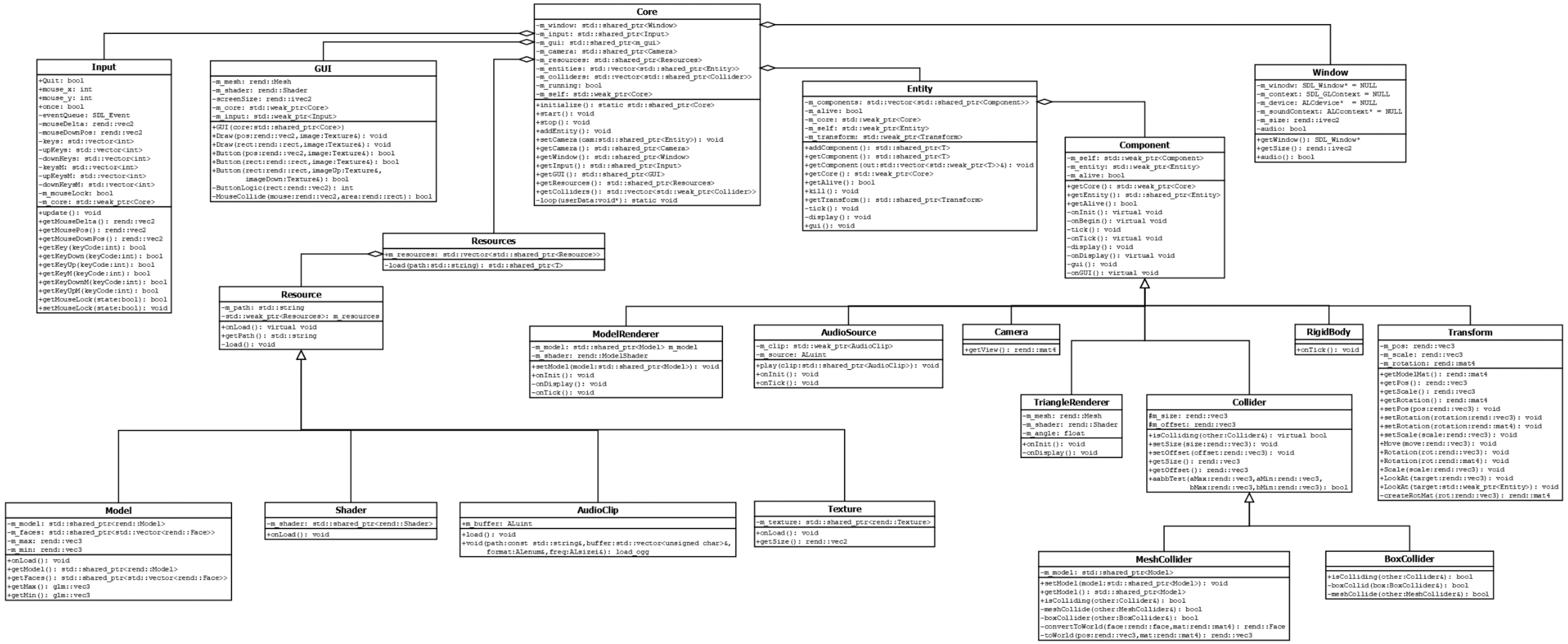
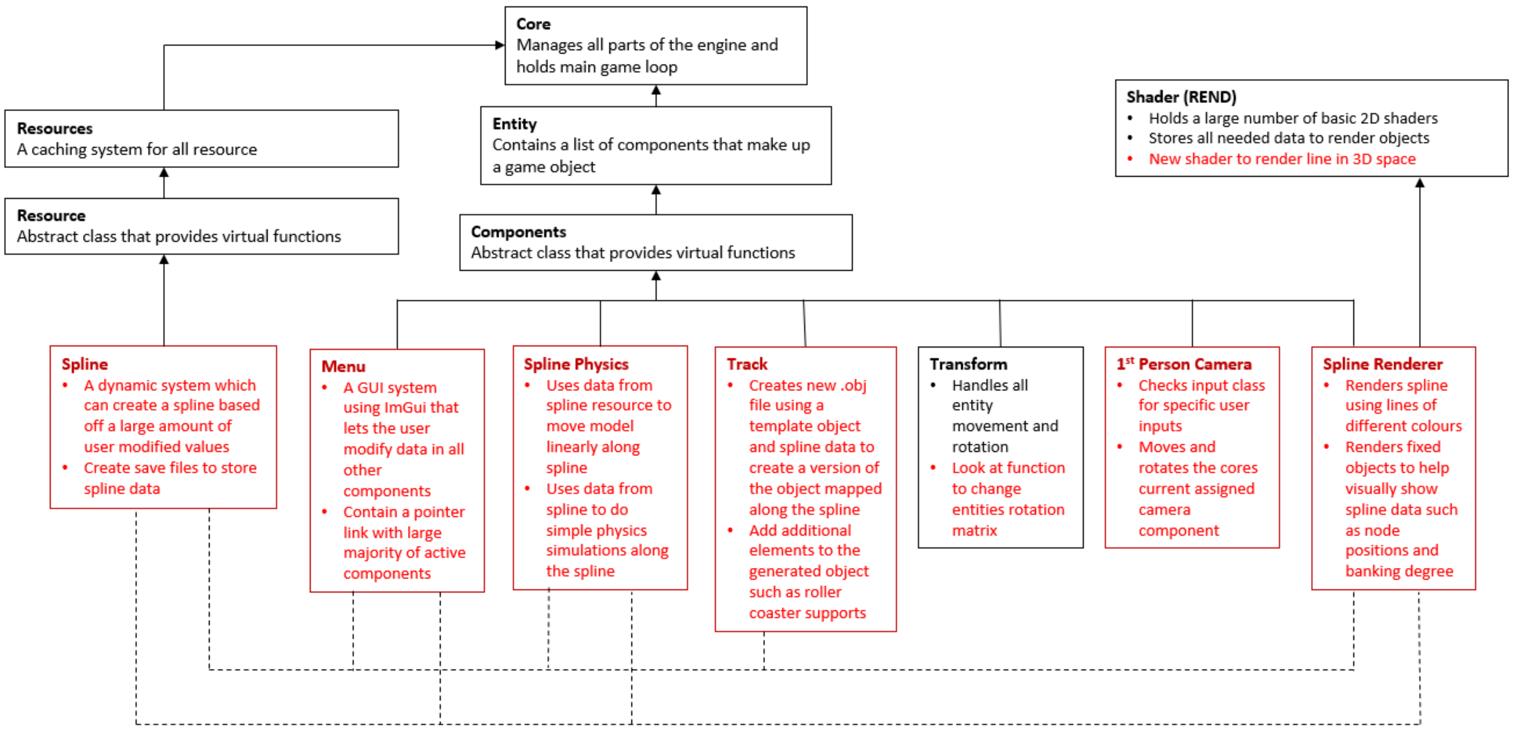

Core
-m_window: std::shared_ptr<Window>
-m_input: std::shared_ptr<Input>
-m_gui: std::shared_ptr<m_gui>
-m_camera: std::shared_ptr<Camera>
-m_resources: std::shared_ptr<Resources>
-m_entities: std::vector<std::shared_ptr<Entity>>
-m_colliders: std::vector<std::shared_ptr<Collider>>
-m_running: bool
-m_self: std::weak_ptr<Core>
+initialize(): static std::shared_ptr<Core>
+start(): void
+stop(): void
+addEntity(): void
+setCamera(cam:std::shared_ptr<Entity>): void
+getCamera(): std::shared_ptr<Camera>
+getWindow(): std::shared_ptr<Window>
+getInput(): std::shared_ptr<Input>
+getGUI(): std::shared_ptr<GUI>
+getResources(): std::shared_ptr<Resources>
+getColliders(): std::vector<std::weak_ptr<Collider>>
-loop(userData:void*): static void
Input
+Quit: bool
+mouse_x: int
+mouse_y: int
+once: bool
-eventQueue: SDL_Event
-mouseDelta: rend::vec2
-mouseDownPos: rend::vec2
-keys: std::vector<int>
-upKeys: std::vector<int>
-downKeys: std::vector<int>
-keysM: std::vector<int>
-upKeysM: std::vector<int>
-downKeysM: std::vector<int>
-m_mouseLock: bool
-m_core: std::weak_ptr<Core>
+update(): void
+getMouseDelta(): rend::vec2
+getMousePos(): rend::vec2
+getMouseDownPos(): rend::vec2
+getKey(keyCode:int): bool
+getKeyDown(keyCode:int): bool
+getKeyUp(keyCode:int): bool
+getKeyM(keyCode:int): bool
+getKeyDownM(keyCode:int): bool
+getKeyUpM(keyCode:int): bool
+getMouseLock(state:bool): bool
+setMouseLock(state:bool): void
GUI
-m_mesh: rend::Mesh
-m_shader: rend::Shader
-screenSize: rend::ivec2
-m_core: std::weak_ptr<Core>
-m_input: std::weak_ptr<Input>
+GUI(core:std::shared_ptr<Core>)
+Draw(pos:rend::vec2,image:Texture&): void
+Draw(rect:rend::rect,image:Texture&): void
+Button(pos:rend::vec2,image:Texture&): bool
+Button(rect:rend::rect,image:Texture&): bool
+Button(rect:rend::rect,imageUp:Texture&,
imageDown:Texture&): bool
-ButtonLogic(rect:rend::vec2): int
-MouseCollide(mouse:rend::vec2,area:rend::rect): bool
Entity
-m_components: std::vector<std::shared_ptr<Component>>
-m_alive: bool
-m_core: std::weak_ptr<Core>
-m_self: std::weak_ptr<Entity>
-m_transform: std::weak_ptr<Transform>
+addComponent(): std::shared_ptr<T>
+getComponent(): std::shared_ptr<T>
+getComponent(out:std::vector<std::weak_ptr<T>>&): void
+getCore(): std::weak_ptr<Core>
+getAlive(): bool
+kill(): void
+getTransform(): std::shared_ptr<Transform>
-tick(): void
-display(): void
+gui(): void
Window
-m_winodw: SDL_Window* = NULL
-m_context: SDL_GLContext = NULL
-m_device: ALCdevice* = NULL
-m_soundContext: ALCcontext* = NULL
-m_size: rend::ivec2
-audio: bool
+getWindow(): SDL_Window*
+getSize(): rend::ivec2
+audio(): bool
Component
-m_self: std::weak_ptr<Component>
-m_entity: std::weak_ptr<Entity>
-m_alive: bool
+getCore(): std::weak_ptr<Core>
+getEntity(): std::shared_ptr<Entity>
+getAlive(): bool
-onInit(): virtual void
-onBegin(): virtual void
-tick(): void
-onTick(): virtual void
-display(): void
-onDisplay(): virtual void
-gui(): void
-onGUI(): virtual void
Resources
+m_resources: std::vector<std::shared_ptr<Resource>>
-load(path:std::string): std::shared_ptr<T>
Resource
-m_path: std::string
-std::weak_ptr<Resources>: m_resources
+onLoad(): virtual void
+getPath(): std::string
-load(): void
ModelRenderer
-m_model: std::shared_ptr<Model> m_model
-m_shader: rend::ModelShader
+setModel(model:std::shared_ptr<Model>): void
+onInit(): void
-onDisplay(): void
-onTick(): void
AudioSource
-m_clip: std::weak_ptr<AudioClip>
-m_source: ALuint
+play(clip:std::shared_ptr<AudioClip>): void
+onInit(): void
+onTick(): void
Camera
+getView(): rend::mat4
RigidBody
+onTick(): void
Transform
-m_pos: rend::vec3
-m_scale: rend::vec3
-m_rotation: rend::mat4
+getModelMat(): rend::mat4
+getPos(): rend::vec3
+getScale(): rend::vec3
+getRotation(): rend::mat4
+setPos(pos:rend::vec3): void
+setRotation(rotation:rend::vec3): void
+setRotation(rotation:rend::mat4): void
+setScale(scale:rend::vec3): void
+Move(move:rend::vec3): void
+Rotation(rot:rend::vec3): void
+Rotation(rot:rend::mat4): void
+Scale(scale:rend::vec3): void
+LookAt(target:rend::vec3): void
+LookAt(target:std::weak_ptr<Entity>): void
-createRotMat(rot:rend::vec3): rend::mat4
TriangleRenderer
-m_mesh: rend::Mesh
-m_shader: rend::Shader
-m_angle: float
+onInit(): void
-onDisplay(): void
Collider
#m_size: rend::vec3
#m_offset: rend::vec3
+isColliding(other:Collider&): virtual bool
+setSize(size:rend::vec3): void
+setOffset(offset:rend::vec3): void
+getSize(): rend::vec3
+getOffset(): rend::vec3
+aabbTest(aMax:rend::vec3,aMin:rend::vec3,
bMax:rend::vec3,bMin:rend::vec3): bool
Model
-m_model: std::shared_ptr<rend::Model>
-m_faces: std::shared_ptr<std::vector<rend::Face>>
-m_max: rend::vec3
-m_min: rend::vec3
+onLoad(): void
+getModel(): std::shared_ptr<rend::Model>
+getFaces(): std::shared_ptr<std::vector<rend::Face>>
+getMax(): glm::vec3
+getMin(): glm::vec3
Shader
-m_shader: std::shared_ptr<rend::Shader>
+onLoad(): void
AudioClip
+m_buffer: ALuint
+load(): void
+void(path:const std::string&,buffer:std::vector<unsigned char>&,
format:ALenum&,freq:ALsizei&): load_ogg
Texture
-m_texture: std::shared_ptr<rend::Texture>
+onLoad(): void
+getSize(): rend::vec2
MeshCollider
-m_model: std::shared_ptr<Model>
+setModel(model:std::shared_ptr<Model>): void
+getModel(): std::shared_ptr<Model>
+isColliding(other:Collider&): bool
-meshCollide(other:MeshCollider&): bool
-boxCollider(other:BoxCollider&): bool
-convertToWorld(face:rend::face,mat:rend::mat4): rend::Face
-toWorld(pos:rend::vec3,mat:rend::mat4): rend::vec3
BoxCollider
+isColliding(other:Collider&): bool
-boxCollid(box:BoxCollider&): bool
-meshCollide(other:MeshCollider&): bool

Fig.21: UML diagram of original Red Engine before any upgrades/modifications

Before work could be started on the main bulk of the project this engine needed some upgrades to support some of the new features that would be added, these upgrades include:

- Improvements to transform component (lookAt function).
- Further modifications to REND to allow line rendering.
- A re-write of the current camera system to convert it from a 3D follow camera to a 1st person camera.
- New functions in the resource caching system to re-load already cached resources.

Other than these upgrades the bulk of the project will be implemented by writing new component and resource classes without majorly changing the original engine architecture. *Fig.22* is a simplified UML diagram that shows the planned additional resources/components that will be added to the engine as part of the project and there intended features.



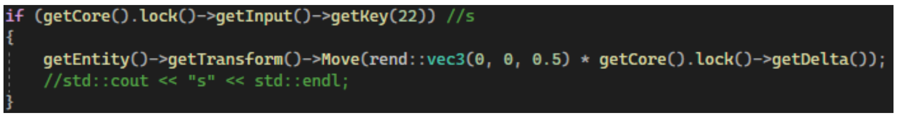
Core
Manages all parts of the engine and holds main game loop
Entity
Contains a list of components that make up a game object
Components
Abstract class that provides virtual functions
Resources
A caching system for all resource
Resource
Abstract class that provides virtual functions
Shader (REND)
• Holds a large number of basic 2D shaders
• Stores all needed data to render objects
• New shader to render line in 3D space
Spline
• A dynamic system which can create a spline based off a large amount of user modified values
• Create save files to store spline data
Menu
• A GUI system using ImGui that lets the user modify data in all other components
• Contain a pointer link with large majority of active components
Spline Physics
• Uses data from spline resource to move model linearly along spline
• Uses data from spline to do simple physics simulations along the spline
Track
• Creates new .obj file using a template object and spline data to create a version of the object mapped along the spline
• Add additional elements to the generated object such as roller coaster supports
Transform
• Handles all entity movement and rotation
• Look at function to change entities rotation matrix
1st Person Camera
• Checks input class for specific user inputs
• Moves and rotates the cores current assigned camera component
Spline Renderer
• Renders spline using lines of different colours
• Renders fixed objects to help visually show spline data such as node positions and banking degree

Fig.22: Planned additions/modifications to the engine (and planned data flow)

Red Engine Upgrades

1st Person Camera

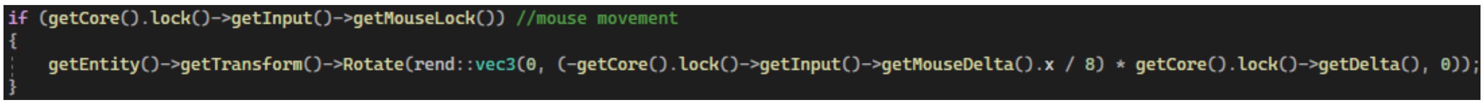
The first feature implemented as part of this project was the conversion of the existing 3rd person camera to a 1st person camera, this was needed to view all other features that would later be added. The camera uses the w,a,s,d,q,e keys to move along the x,y and z axis, the left and right arrow keys and the mouse are used to rotate the camera.



```
if (getCore().lock()->getInput()->getKey(22)) //s
{
    getEntity()->getTransform()->Move(rend::vec3(0, 0, 0.5) * getCore().lock()->getDelta());
    //std::cout << "s" << std::endl;
}
```

Fig.23: Statement to check if key is pressed and move camera

To use the mouse for rotation the curser needs to be continually wrapped to the center of the screen to measure its change in position each frame, the engine already had a function that calculated this in the input class so this was used. However, as the curser will later need to stop wrapping to use the GUI system that will be added the camera must first check if the curser is currently wrapping before calculating the rotation.

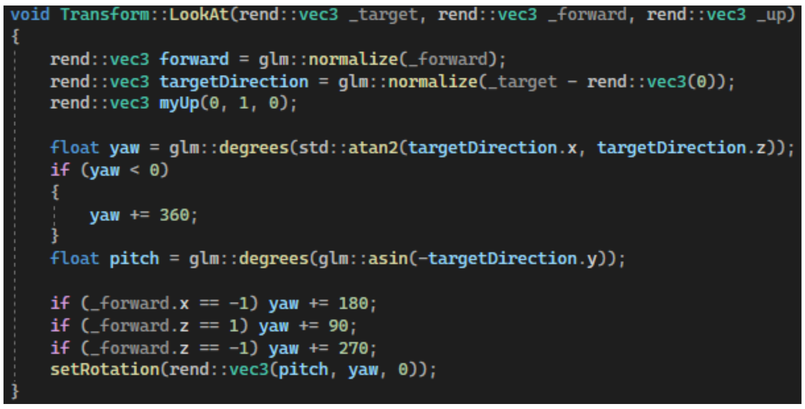


```
if (getCore().lock()->getInput()->getMouseLock()) //mouse movement
{
    getEntity()->getTransform()->Rotate(rend::vec3(0, (-getCore().lock()->getInput()->getMouseDelta().x / 8) * getCore().lock()->getDelta(), 0));
}
```

Fig.24: Statement to check if curser is currently wrapping and rotate the camera by change in mouse position

Transform LookAt() Function

In order to render objects moving along a spline, they will need to be able to face the spline curve's tangent vector, therefore before the spline itself is implemented a lookAt() function needed to be added to the transform component. This function uses some basic trigonometry to calculate the angle between a target vector and the default forward vector. As trigonometry only works on a 180-degree area some extra angle needs to be added for the yaw depending on the object's default forward vector to make this work across 360 degrees.



```
void Transform::LookAt(rend::vec3 _target, rend::vec3 _forward, rend::vec3 _up)
{
    rend::vec3 forward = glm::normalize(_forward);
    rend::vec3 targetDirection = glm::normalize(_target - rend::vec3(0));
    rend::vec3 myUp(0, 1, 0);

    float yaw = glm::degrees(std::atan2(targetDirection.x, targetDirection.z));
    if (yaw < 0)
    {
        yaw += 360;
    }
    float pitch = glm::degrees(glm::asin(-targetDirection.y));

    if (_forward.x == -1) yaw += 180;
    if (_forward.z == 1) yaw += 90;
    if (_forward.z == -1) yaw += 270;
    setRotation(rend::vec3(pitch, yaw, 0));
}
```

Fig.25: Transform component's LookAt() function

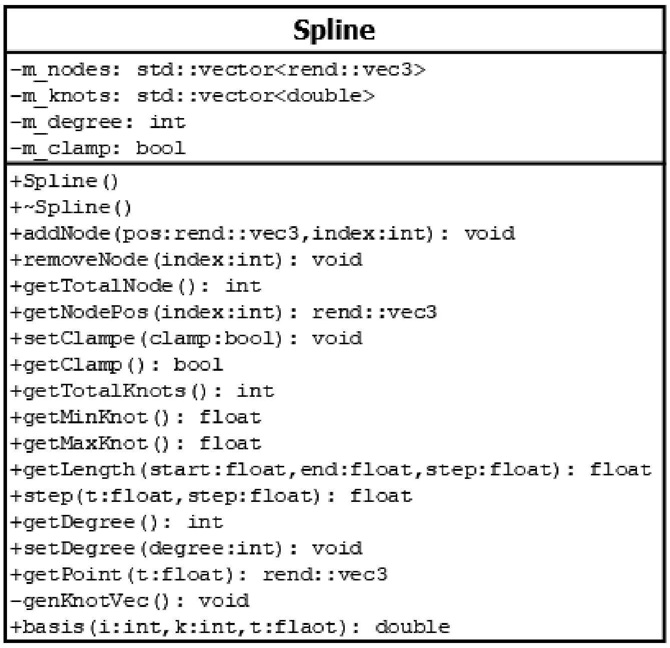
Line Shader

Well, the engine already had the capabilities needed to renderer a line it lacked a dedicated line shader for the rendering pipeline to use, this is obviously needed if a spline is to be rendered. The line shader itself is very simple as the vertex shader is just a model-view-projection matrix transformation and the fragment shader just colours the fragments with the inputted colour.

Fig.26: Demonstration of engine line rendering capabilities

Spline Basics

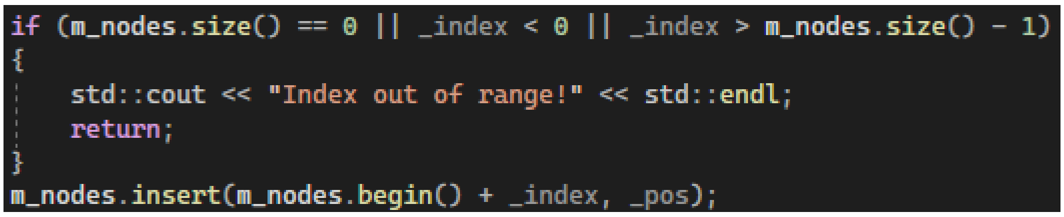
After Red Engine had been upgraded, work on the projects new components and resources began, this started with the spline resource. To begin with, the goal was to create a very basic spline resource with stripped down functionality that could create a basic spline which could then be built upon by adding things such as a roll and file system.



```
Spline
-m_nodes: std::vector<rend::vec3>
-m_knots: std::vector<double>
-m_degree: int
-m_clamp: bool
+Spline()
+~Spline()
+addNode(pos:rend::vec3,index:int): void
+removeNode(index:int): void
+getTotalNode(): int
+getNodePos(index:int): rend::vec3
+setClampe(clamp:bool): void
+getClamp(): bool
+getTotalKnots(): int
+getMinKnot(): float
+getMaxKnot(): float
+getLength(start:float,end:float,step:float): float
+step(t:float,step:float): float
+getDegree(): int
+setDegree(degree:int): void
+getPoint(t:float): rend::vec3
-genKnotVec(): void
+basis(i:int,k:int,t:flaot): double
```

Fig.27: The stripped-down spline resource planned to be implemented.

First some code was written that simply cratered an empty class and added the key spline attributes along with getter and setter functions where appropriate so the attributes could stay private. The spline node vector, m_nodes, needed more complex functions that took an index value to return/add/remove a specific node in the vector. These functions had additional checks added to them to check that the index was in range of the list to prevent the program from attempting to access invalid memory locations.



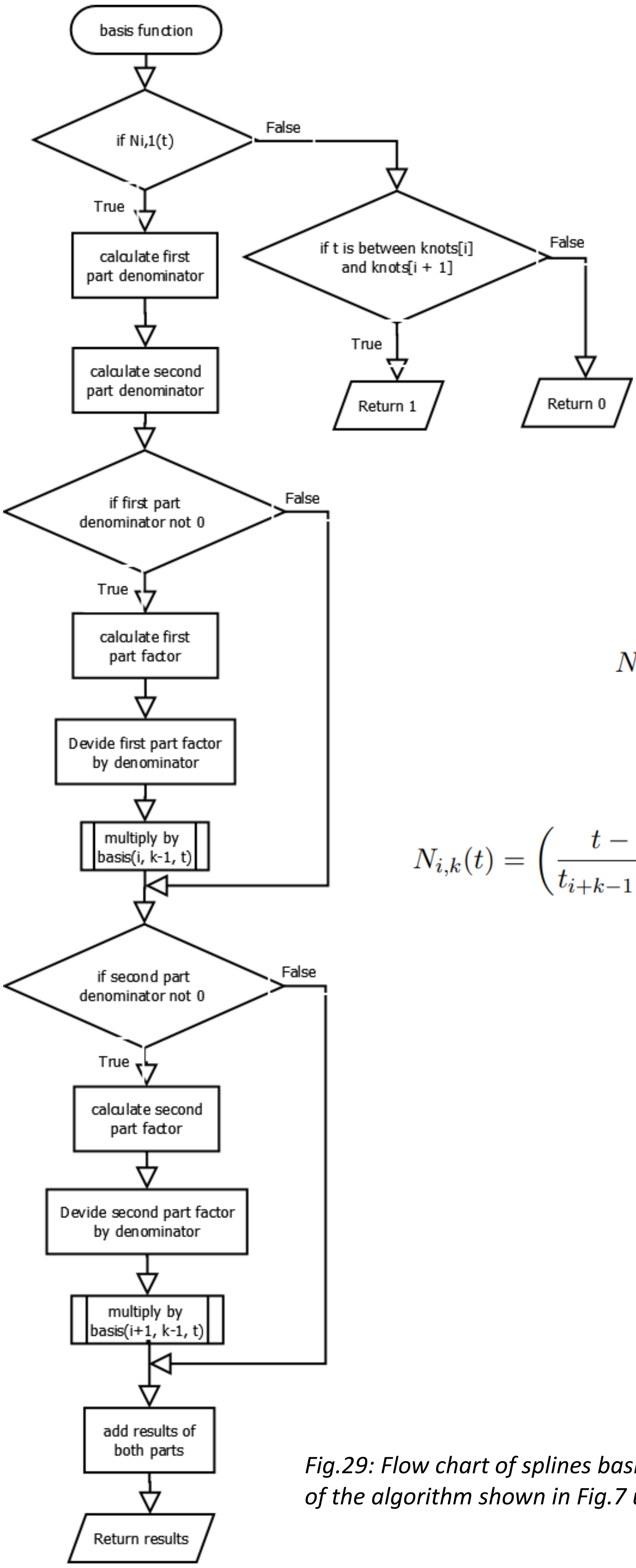
```
if (m_nodes.size() == 0 || _index < 0 || _index > m_nodes.size() - 1)
{
    std::cout << "Index out of range!" << std::endl;
    return;
}
m_nodes.insert(m_nodes.begin() + _index, _pos);
```

Fig.28: Code to check index is in range of m_nodes inside addNode()

Having the framework of the spline class in place, the implementation of the spline's basis function could start, just like in the research this will be split into two sections. The piecewise continuous function P(t), Fig.6, which will be implemented in the getPoint() function, and the $N_{i,k}$ calculation, Fig.7, which will be implemented in the basis() function. The piecewise continuous function is simple to implement and can be done with a for loop but the $N_{i,k}$ calculation needed a more complex algorithm shown in Fig.29.

$$\mathbf{P}(t) = \sum_{i=0}^{n} \mathbf{P}_i N_{i,k}(t)$$

Fig 6: Definition of the B-spline curve P(t) piecewise continuous function (Joy 2008, p.1)



basis function
if Ni,1(t)
False
True
if t is between knots[i] and knots[i + 1]
False
True
Return 1
Return 0
calculate first part denominator
calculate second part denominator
if first part denominator not 0
False
True
calculate first part factor
Devide first part factor by denominator
multiply by basis(i, k-1, t)
if second part denominator not 0
False
True
calculate second part factor
Devide second part factor by denominator
multiply by basis(i+1, k-1, t)
add results of both parts
Return results

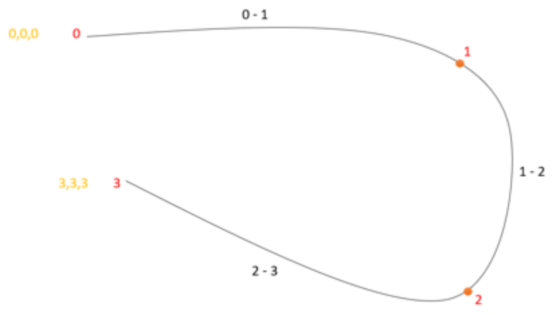
$$N_{i,1}(t) = \begin{cases} 1 & \text{if } u \in [t_i, t_{i+1}) \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,k}(t) = \left(\frac{t - t_i}{t_{i+k-1} - t_i} \right) N_{i,k-1}(t) + \left(\frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \right) N_{i+1,k-1}(t)$$

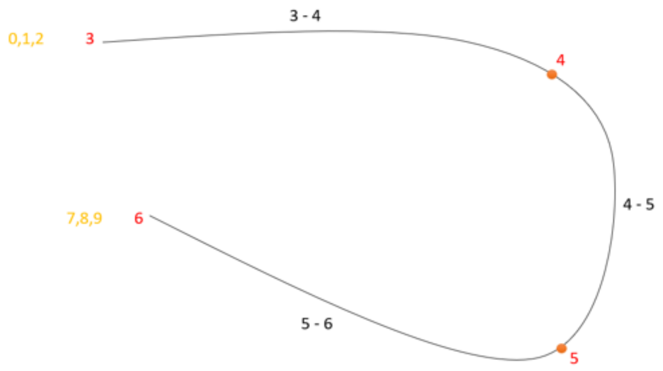
Fig 7: $N_{i,k}$ defined recursively as used in Fig.6
(Joy 2008, p.2)

Fig.29: Flow chart of splines basis(), this is the implemented of the algorithm shown in Fig.7 used by the spline class

The next part of the stripped-down spline class is the knot vector, unlike the node vector, the knot vector is self-generating using the spline degree and total number of nodes to calculate the total number of knots. Where this gets more complex is with the spline clamping, as explained in the B-spline research there are different rules for calculating knot values for both clamped and unclamped curves. Furthermore, the spline curve t value range should be the same on an identical spline curve when both clamped and unclamped. This can be achieved by starting an unclamped knot vector at the value, 0 – degree, as the first degree number of knots and the last degree number of knots t-value don't lie on the curve, this can be seen more clearly in fig.30/31.

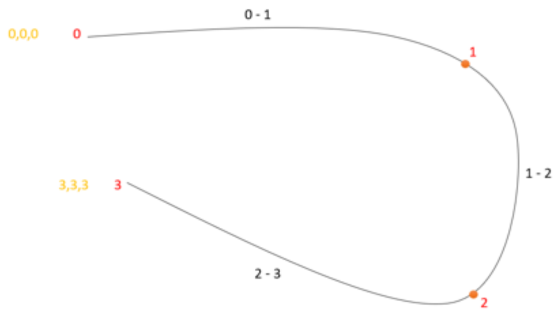


Clamped cubic B-spine curve t-values with the knot vector [0,0,0,0,1,2,3,3,3,3]

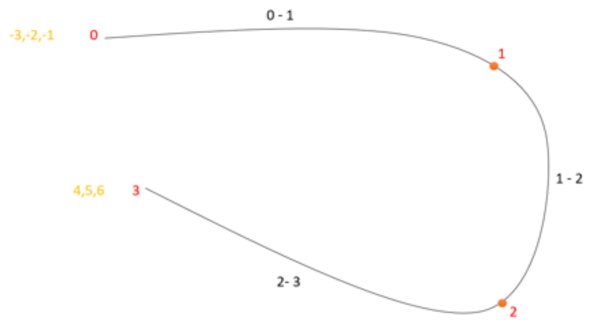


Unclamped cubic B-spine curve t-values with the knot vector [0,1,2,3,4,5,6,7,8,9]

Fig.30: A clamped and unclamped cubic B-spline with three curve sections t-value range without knot vector correction



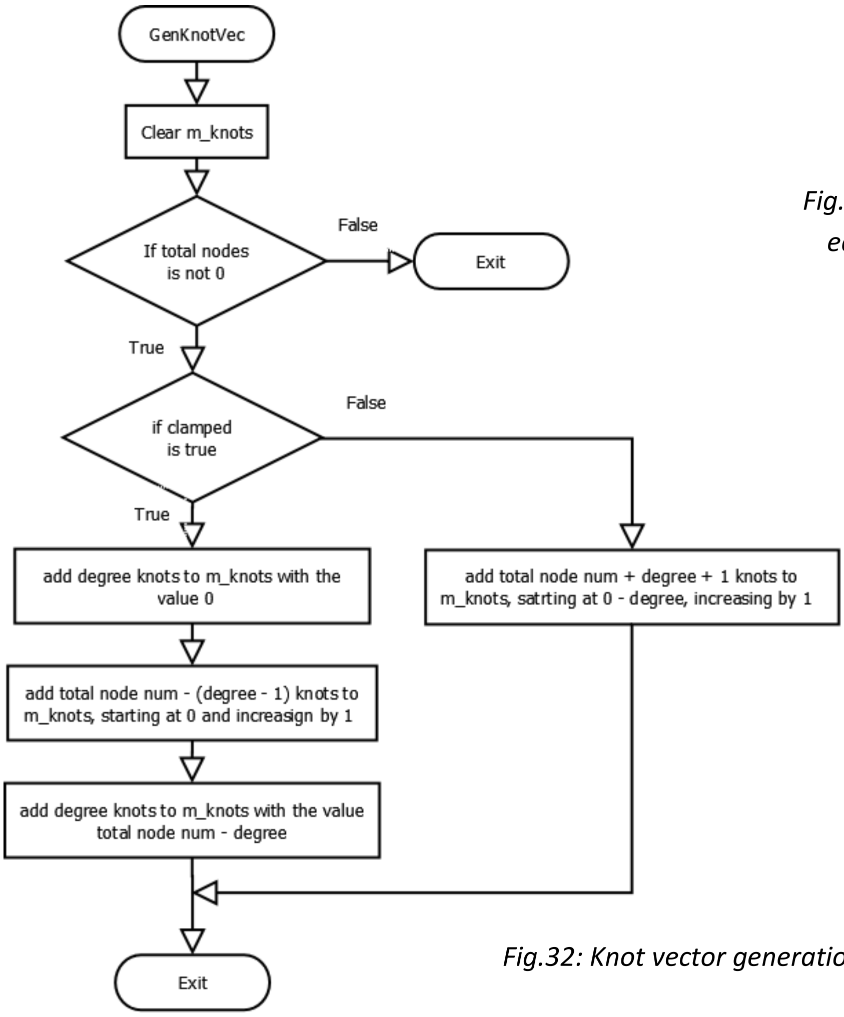
Clamped cubic B-spine curve t-values with the knot vector [0,0,0,0,1,2,3,3,3,3]



Unclamped cubic B-spine curve t-values with the knot vector [-3,-2,-1,0,1,2,3,4,5,6]

Fig.31: A clamped and unclamped cubic B-spline with three curve sections t-value range with knot vector correction

The algorithm implemented to generate a knot vector that can meet all of these conditions can be seen in Fig.32 alongside the rules for generating clamped and unclamped knot vectors in Fig.9/10. getMinKnot() and getMaxKnot() functions were also created to return the knot values at the start and the end of the curve, this is calculated by simply returning the knot at index degree and the knot at index knot length – degree.


GenKnotVec
Clear m_knots
If total nodes is not 0
False
Exit
True
if clamped is true
False
True
add degree knots to m_knots with the value 0
add total node num + degree + 1 knots to m_knots, satrting at 0 - degree, increasing by 1
add total node num - (degree - 1) knots to m_knots, starting at 0 and increasign by 1
add degree knots to m_knots with the value total node num - degree
Exit

$$\begin{aligned} t_i &= t_1, & i \leq k \\ t_{i+1} - t_i &= \textbf{constant}, & k \leq i < n+2 \\ t_i &= t_{k+(n+1)}, & i \geq n+2 \end{aligned}$$

*Fig.9: Clamped knot vectors with k equal knot values at each end (*B-splines, University of Cambridge 1999*)*

$$t_{i+1} - t_i = \textbf{constant}, \forall i$$

*Fig.10: Unclamped knot (*B-splines, University of Cambridge 1999*)*

Fig.32: Knot vector generation function

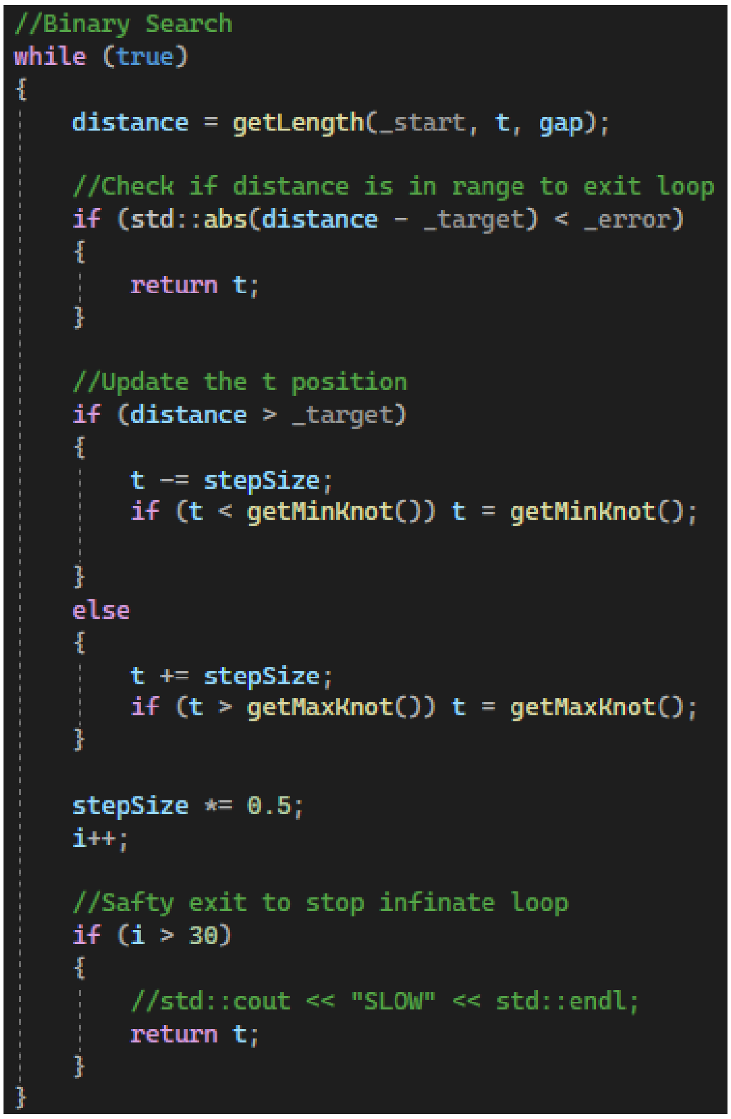
With all of the above implemented the spline resource can now generate spline curves, however it is still missing some functions needed to renderer the curve to the screen. The rendering itself will be handled by a spline renderer component using the line shader that will be added later, the spline resource itself needs step() and getLength() functions for this spline renderer to use.

The getLength() function will return the length between two t values on the spline curve, this function does not need to return the exact length of the curve as a close approximation will be faster to calculate and simpler to implement. This function will use the arc-length algorithm to find the length of the curve segment. This algorithm, shown in Fig.33, divides the curve segment into lots of very small sections (the user can enter a value that changes how many segments) and then treats each section like a straight line. Then it adds up the length of all these straight lines to give an overall length approximation of the curve segment, as explained by Bazett 2019. This function will also check if the starting t value is larger than the end t value and flip the inputs and then flip the output so the function can also return negative lengths if moving backwards along the spline curve, this will be useful in the step() function.

$$l_i = \int_{t_i}^{t_{i+1}} \sqrt{(x'(t))^2 + (y'(t))^2 + (z'(t))^2} dt$$

Fig.33 Arc-length approximation (Wang 2002 p.391)

The last part of the basic spline resource is the step() function, this function will start at a t-value, move a set distance along the spline curve, and then return the t-value at this distance. This can be easily added by going through all t-values on the spline and using getLength() until the correct t-value at the correct length is found, however this is very slow. It is important that this function is fast as it will be called regularly by an object that is moving along the spline, the spline renderer and the track mesh creator. The getLength() function can't be made faster as lowering the number of iterations it does will also result in lower length accuracy, Wang 2002. There for the only solution is to make the step() function as efficient as possible. The best way to achieve this is by using a binary search as it is O(log n + 1) in its worse-case complexity (proven by Lin 2019) and in most cases the t-value at the correct length will be found in under 17 iterations meaning getLength() can be called less times.



```
//Binary Search
while (true)
{
    distance = getLength(_start, t, gap);

    //Check if distance is in range to exit loop
    if (std::abs(distance - _target) < _error)
    {
        return t;
    }

    //Update the t position
    if (distance > _target)
    {
        t -= stepSize;
        if (t < getMinKnot()) t = getMinKnot();

    }
    else
    {
        t += stepSize;
        if (t > getMaxKnot()) t = getMaxKnot();
    }

    stepSize *= 0.5;
    i++;

    //Safty exit to stop infinate loop
    if (i > 30)
    {
        //std::cout << "SLOW" << std::endl;
        return t;
    }
}
```

Fig.34: Binary search used in splines step() function

Spline Renderer Basic

The spline renderer is a component that will handle rendering the spline curve, it will also render some models to display other information to the user such as node locations and later the roll angle of the spline. Just like the spline this will first be implemented as a basic spline renderer and be added to as more functionality is added to the spline resource. First it will draw the spline curve, connect the nodes and render a simple model at each node position. To achieve this, it will use two shaders, a line shader for the spline curve and a model shader for the models. To render the spline curve it will use the splines step() and getPoint() functions, moving a set distance along the spline then getting the point, it will then connect these points with a line. This works simile to the arc-length approximation without needing to implement the algorithm itself as it is using the spline's functions.

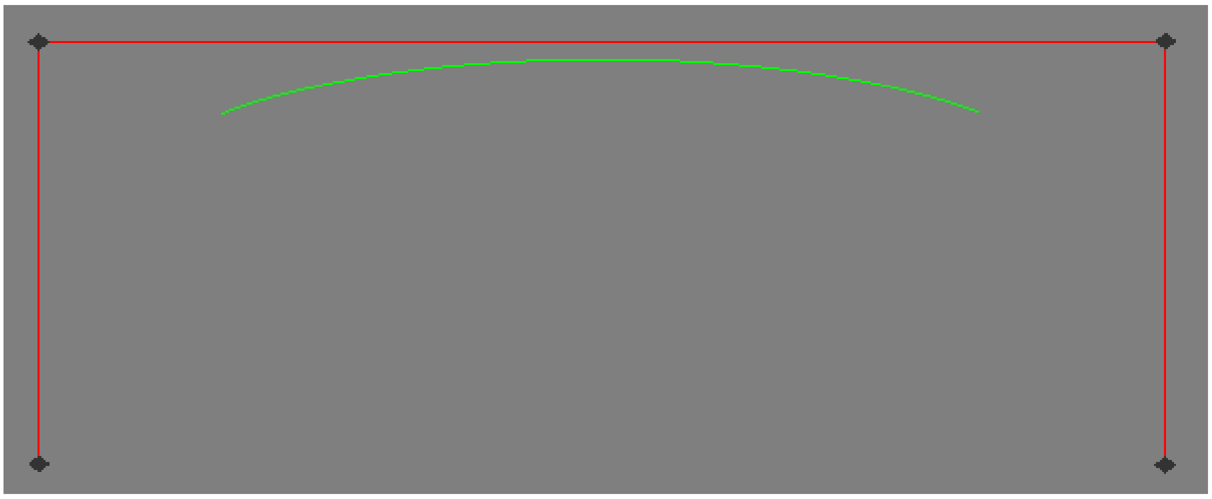


Fig.35: A unclamped cubic B-spline renderer by the program

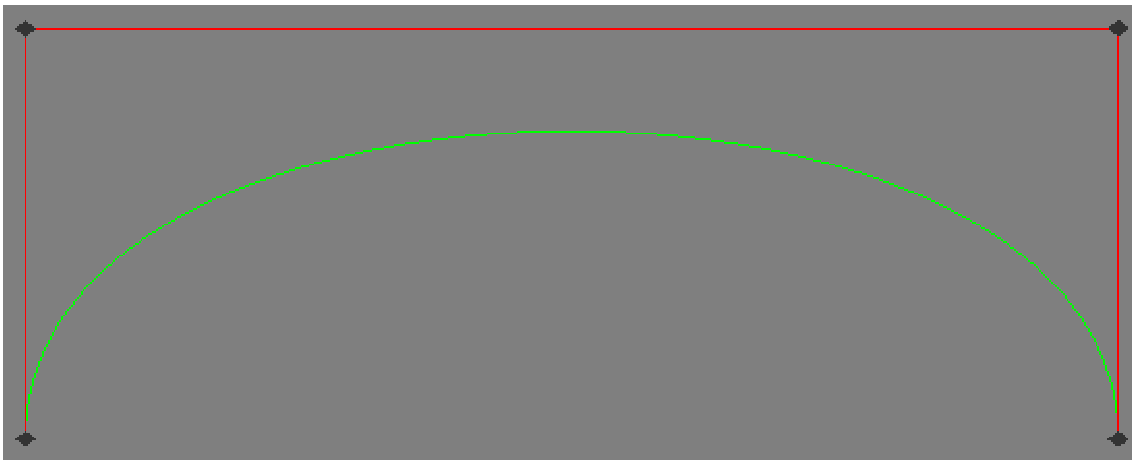
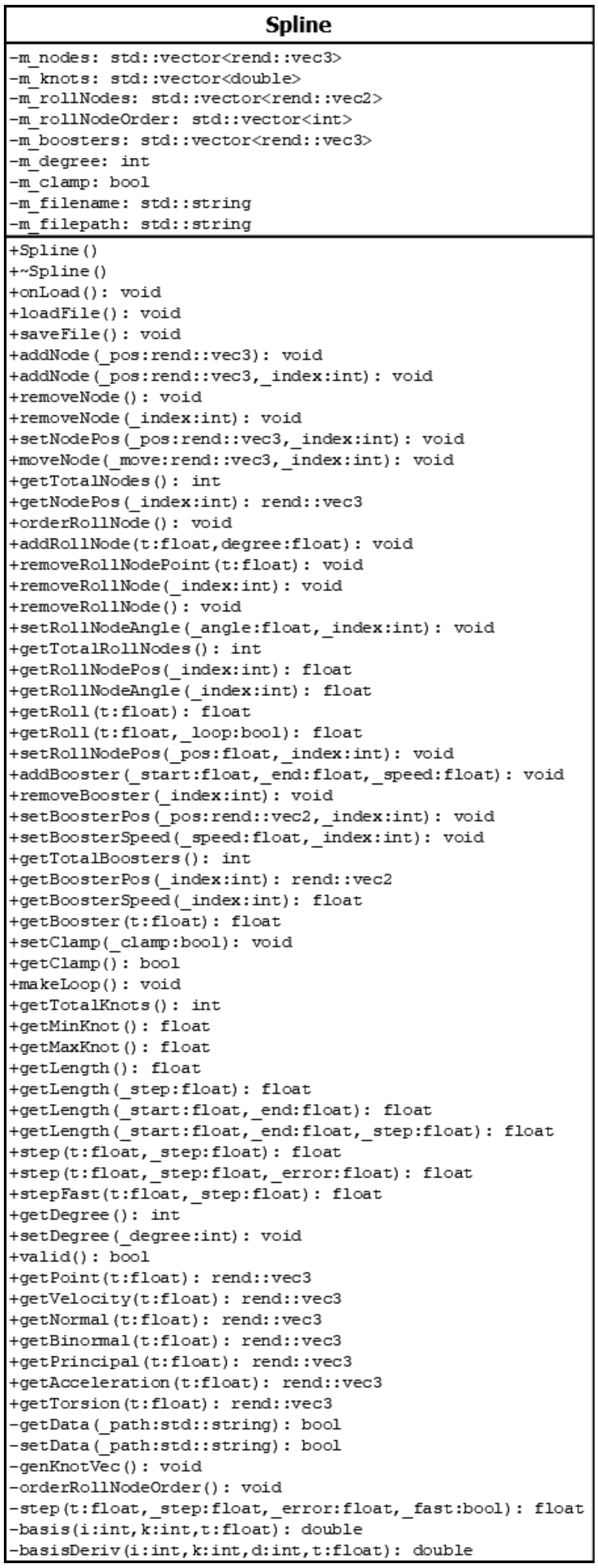


Fig.36: A clamped cubic B-spline renderer by the program

Spline Advanced



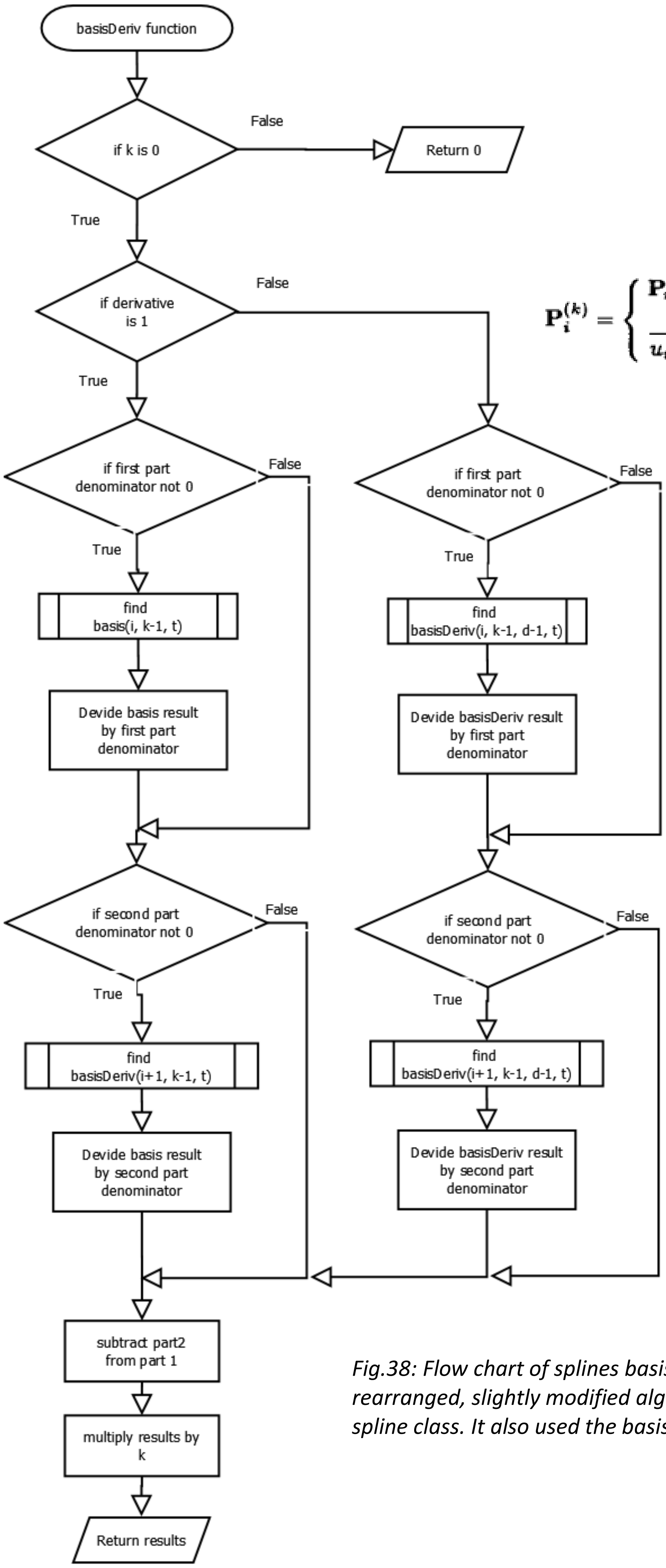
```
Spline
-m_nodes: std::vector<rend::vec3>
-m_knots: std::vector<double>
-m_rollNodes: std::vector<rend::vec2>
-m_rollNodeOrder: std::vector<int>
-m_boosters: std::vector<rend::vec3>
-m_degree: int
-m_clamp: bool
-m_filename: std::string
-m_filepath: std::string
+Spline()
+~Spline()
+onLoad(): void
+loadFile(): void
+saveFile(): void
+addNode(_pos:rend::vec3): void
+addNode(_pos:rend::vec3,_index:int): void
+removeNode(): void
+removeNode(_index:int): void
+setNodePos(_pos:rend::vec3,_index:int): void
+moveNode(_move:rend::vec3,_index:int): void
+getTotalNodes(): int
+getNodePos(_index:int): rend::vec3
+orderRollNode(): void
+addRollNode(t:float,degree:float): void
+removeRollNodePoint(t:float): void
+removeRollNode(_index:int): void
+removeRollNode(): void
+setRollNodeAngle(_angle:float,_index:int): void
+getTotalRollNodes(): int
+getRollNodePos(_index:int): float
+getRollNodeAngle(_index:int): float
+getRoll(t:float): float
+getRoll(t:float,_loop:bool): float
+setRollNodePos(_pos:float,_index:int): void
+addBooster(_start:float,_end:float,_speed:float): void
+removeBooster(_index:int): void
+setBoosterPos(_pos:rend::vec2,_index:int): void
+setBoosterSpeed(_speed:float,_index:int): void
+getTotalBoosters(): int
+getBoosterPos(_index:int): rend::vec2
+getBoosterSpeed(_index:int): float
+getBooster(t:float): float
+setClamp(_clamp:bool): void
+getClamp(): bool
+makeLoop(): void
+getTotalKnots(): int
+getMinKnot(): float
+getMaxKnot(): float
+getLength(): float
+getLength(_step:float): float
+getLength(_start:float,_end:float): float
+getLength(_start:float,_end:float,_step:float): float
+step(t:float,_step:float): float
+step(t:float,_step:float,_error:float): float
+stepFast(t:float,_step:float): float
+getDegree(): int
+setDegree(_degree:int): void
+valid(): bool
+getPoint(t:float): rend::vec3
+getVelocity(t:float): rend::vec3
+getNormal(t:float): rend::vec3
+getBinormal(t:float): rend::vec3
+getPrincipal(t:float): rend::vec3
+getAcceleration(t:float): rend::vec3
+getTorsion(t:float): rend::vec3
-getData(_path:std::string): bool
-setData(_path:std::string): bool
-genKnotVec(): void
-orderRollNodeOrder(): void
-step(t:float,_step:float,_error:float,_fast:bool): float
-basis(i:int,k:int,t:float): double
-basisDeriv(i:int,k:int,d:int,t:float): double
```

Fig.37: The full spline resource planned to be implemented

Now that the basic spline resource has been implemented its full range of functionality can be added, this includes: roll nodes (banking), booster sections, the derivative basis function, overloaded API functions and a file management system.

The overloaded spline functions keep the same functionality they were given in the basis spline resource but with more varied function input options. This was added to make working with the spline API easier as while they are not all needed for this project, the spline resource is part of Red Engine so they will be useful for other projects using the spline and help futureproofed this project.

The basis derivative function is the next major feature to be added, the first derivative of the spline curve is needed to get the slope angle of the curve, this will be needed to animate objects along the spline. Just like the basis function the derivative will be split into two parts, getVelocity() will hold the piecewise continuous function P(t), Fig.6. The function for the derivate of $N_{i,k}$ will be inside the basisDeriv() function and use the algorithm in Fig.12 discussed in the research section. This algorithm will be slightly re-arranged to make it simpler to write in code form and also modified so rather than just returning the first derivative it will be able to return any derivative. Knowing the other derivatives is not needed for this project but again will be useful for other projects and help futureproof this project. The algorithm implemented for the derivative can be seen in Fig.38, while the piecewise continuous function P(t), Fig.6, is implemented using a for loop.


basisDeriv function
if k is 0
False
Return 0
True
if derivative is 1
False
True
if first part denominator not 0
False
True
find basis(i, k-1, t)
Devide basis result by first part denominator
if second part denominator not 0
False
True
find basisDeriv(i+1, k-1, t)
Devide basis result by second part denominator
if first part denominator not 0
False
True
find basisDeriv(i, k-1, d-1, t)
Devide basisDeriv result by first part denominator
if second part denominator not 0
False
True
find basisDeriv(i+1, k-1, d-1, t)
Devide basisDeriv result by second part denominator
subtract part2 from part 1
multiply results by k
Return results

$$\mathbf{P}_i^{(k)} = \begin{cases} \mathbf{P}_i & k = 0 \\ \dfrac{p-k+1}{u_{i+p+1} - u_{i+k}} \left(\mathbf{P}_{i+1}^{(k-1)} - \mathbf{P}_i^{(k-1)} \right) & k > 0 \end{cases}$$

Fig.12: Basis derivative function for a B-spline curve, $N_{i,p-k}(u)$ is defined recursively as $N_{i,k}(t)$ in Fig.7 (Piegl 1997, p.97)

Fig.38: Flow chart of splines basisDeriv(), this is the implemented of a rearranged, slightly modified algorithm shown in Fig.12 used by the spline class. It also used the basis function shown in Fig.29

Another functionality added to the spline resource are the roll nodes, the getter and setter along with the vector functions work the same way as the normal nodes, however rather an having a position in 3D space, the roll nodes are represented by a t position along the spline curve. The getRoll() function will return the roll degree of a t position on the spline curve, this is achieved by finding the closest roll nodes on either side of the target t position and then preforming linear interpolation on both of their roll angles. This allows the angle of roll to gradually change between nodes resulting in smooth animation of objects along the spline.

$$f(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

Fig.39: Linier Interpolation Algorithm used for roll nodes angle (Noor 2015, p279)

Alongside the roll nodes, boosters have also been added to the spline resource, these set a minimum ‘speed’ value to sections of the spline curve, this value can be used by the animated model’s physics to prevent it from going under the speed specified. This lets the booster be used as a chain lift when modeling RCs, the getBoost() function will return the boost speed for a t position along the spline curve. The function checks for overlapping booster sections and returns the fasters speed value so multiple boosters can be overlapped without causing any problems.

The final element added to the spline resource was a file management system, this is needed so the user can save splines to an external file. This not only allows the user to keep a spline after the program is closed but will also allow the user to swap between different splines while using the program. The file system will be built on top of the existing spline resource so all the splines variables will be treated as ‘working data’, then when the setData() function is called this data will be saved to a text file. The getData() function can then be used to update the ‘working data’ by reading a spline text file and updating the spline variables with the data stored there.

The file format of the spline text file will use vec3 values to store the spline data with each vec3 having a character assigned to it, this will function as an identifier tag that will tell the program what type of data the vec3 is storing. There will be four identifying tags to store all the data necessary, this data includes: m_nodes, m_rollNodes, m_boosters, m_degree and m_clamped. The text file name will be entered by the user and stored in the m_filename variable, the spline text file will be stored in a spline folder with in the programs data file.

Identifier	X	Y	Z
d	Spline degree	Spline clamped (1 true, 0 false)	N/A (set to 0)
n	Node x pos	Node y pos	Node z pos
r	Roll node pos on spline	Roll Node degree of rotation	N/A (set to 0)
b	Booster start pos on spline	Booster end pos on spline	Booster speed

Fig.40: File format for spline text files

When the getData() function is called the program will try and load the spline text file with a matching name that the user entered. If the program fails to find a matching text file it will create a new file with default spline values, this will let the user create new splines within the program and prevent them from ever having to open the spline folder manually.

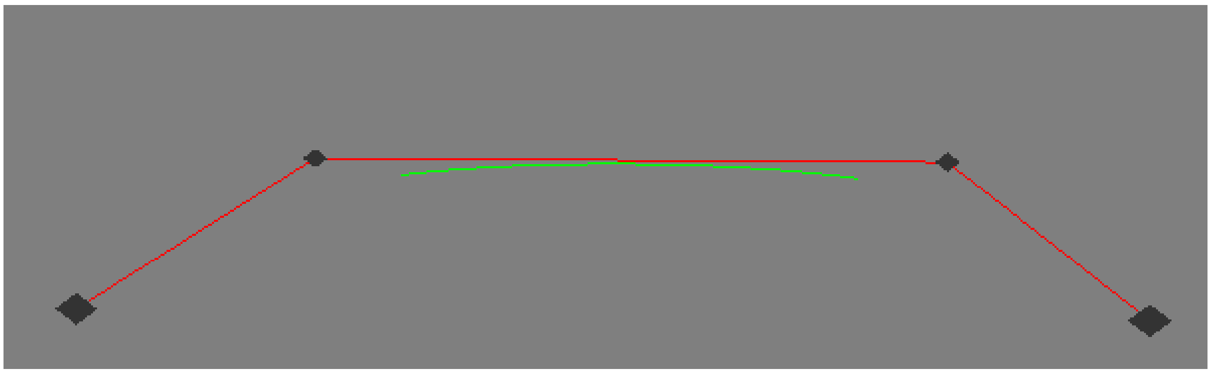


Fig.41: Default spline created by the program when creating a new spline text file

Spline Renderer Advanced

To accommodate the new functionality of the spline resource the spline renderer has had roll markers added so the user can see the location and roll angle of roll nodes. The parts of the spline with boosters placed on it will now also be rendered in a different color so the user can see where boosters start and end.

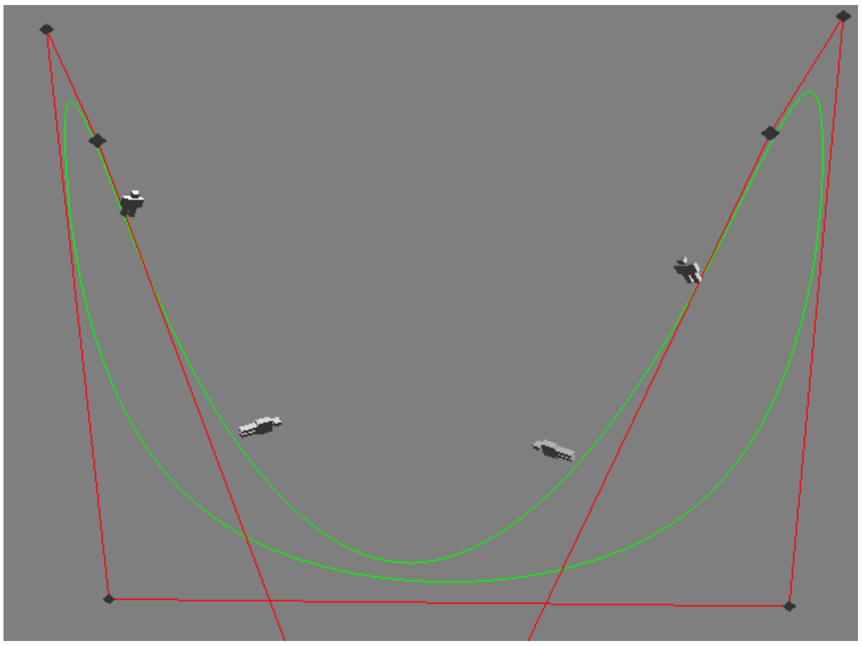


Fig.42: Roll node markers on spline

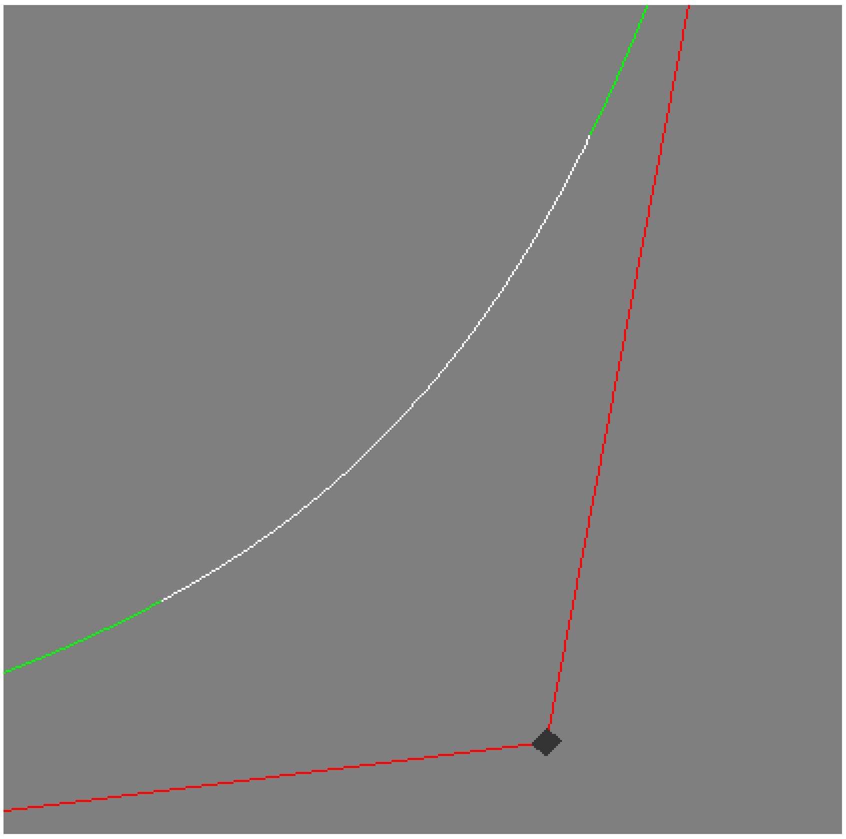
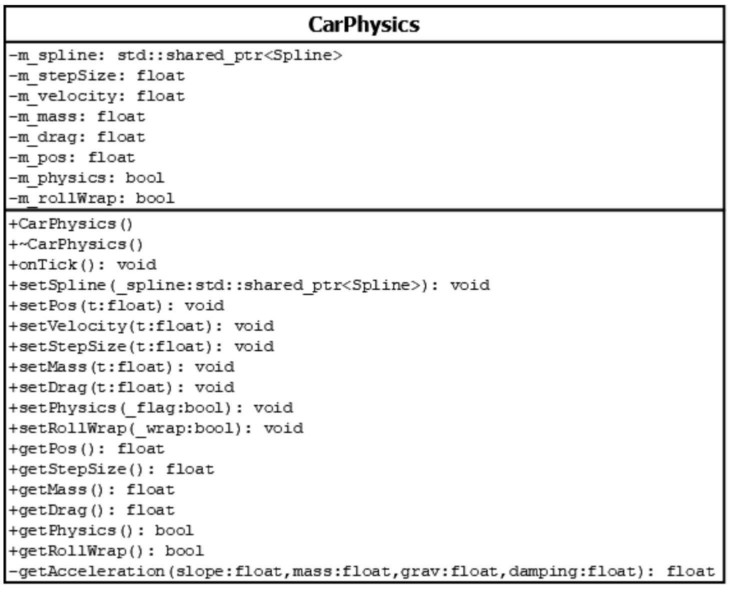


Fig.43: Boosters on spline

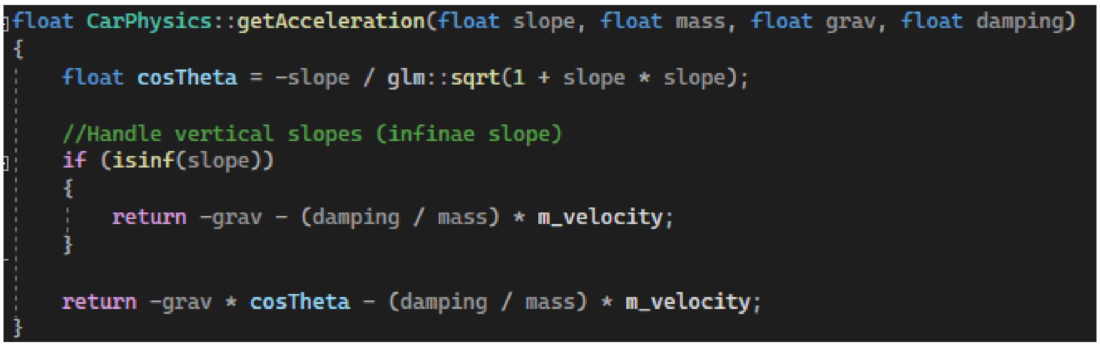
Animated Model

The animated model is a very simple component that moves its entity's position along the spline curve, it can either move at a fixed speed along the spline or use simple physics, where the user can set the mass and drag. It also changes its entity's rotation vector so it matches the spline tangent vector at the point on the spline and also rotates it by the spline's roll value at the point.



```
CarPhysics
-m_spline: std::shared_ptr<Spline>
-m_stepSize: float
-m_velocity: float
-m_mass: float
-m_drag: float
-m_pos: float
-m_physics: bool
-m_rollWrap: bool
+CarPhysics()
+~CarPhysics()
+onTick(): void
+setSpline(_spline:std::shared_ptr<Spline>): void
+setPos(t:float): void
+setVelocity(t:float): void
+setStepSize(t:float): void
+setMass(t:float): void
+setDrag(t:float): void
+setPhysics(_flag:bool): void
+setRollWrap(_wrap:bool): void
+getPos(): float
+getStepSize(): float
+getMass(): float
+getDrag(): float
+getPhysics(): bool
+getRollWrap(): bool
-getAcceleration(slope:float,mass:float,grav:float,damping:float): float
```

Fig.44: Animated model component



```cpp
float CarPhysics::getAcceleration(float slope, float mass, float grav, float damping)
{
    float cosTheta = -slope / glm::sqrt(1 + slope * slope);

    //Handle vertical slopes (infinae slope)
    if (isinf(slope))
    {
        return -grav - (damping / mass) * m_velocity;
    }

    return -grav * cosTheta - (damping / mass) * m_velocity;
}
```

Fig.45: Simple physics implementation (using formula in Fig.16)

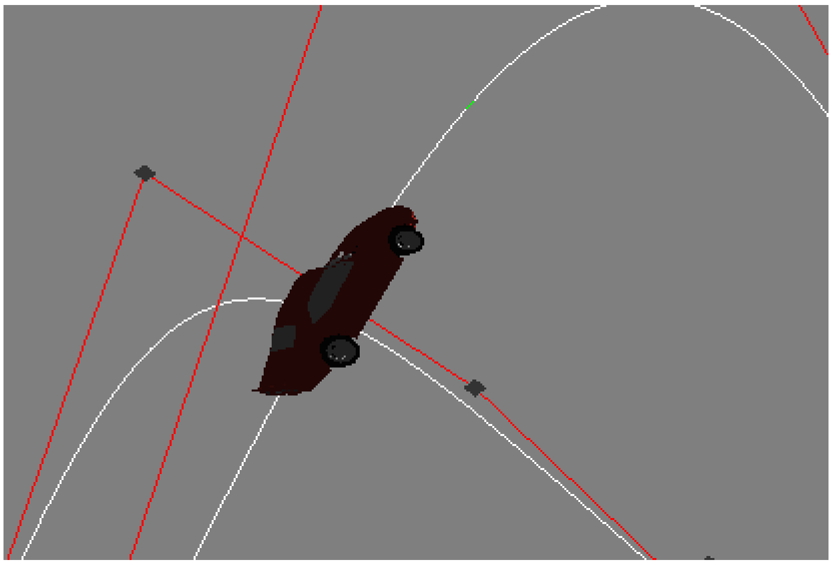


Fig.46: Animated model output

Track Create

The track model creation component takes a template .obj and then maps it along the spline curve, just like the spline implementation this will first be implemented so it can perform its basic functionality, and will then be added to, in this case to improve its performance in model generation time. This component is made up of three key parts:

- A .obj reader function that can read all the data in a template .obj file and store it in local variables.
- A model mapping function that will modify this template data and add to it so that the template model is mapped along the whole spline curve.
- A .obj file writer that will create a new .obj file using the modified data from the mapping function (this file can then be passed back into the program and rendered by a separate model renderer).

The advantages of using this layout is that the program will create a new .obj file, this means any created model can be loaded into other programs such as blender or unity which makes the program more useful.

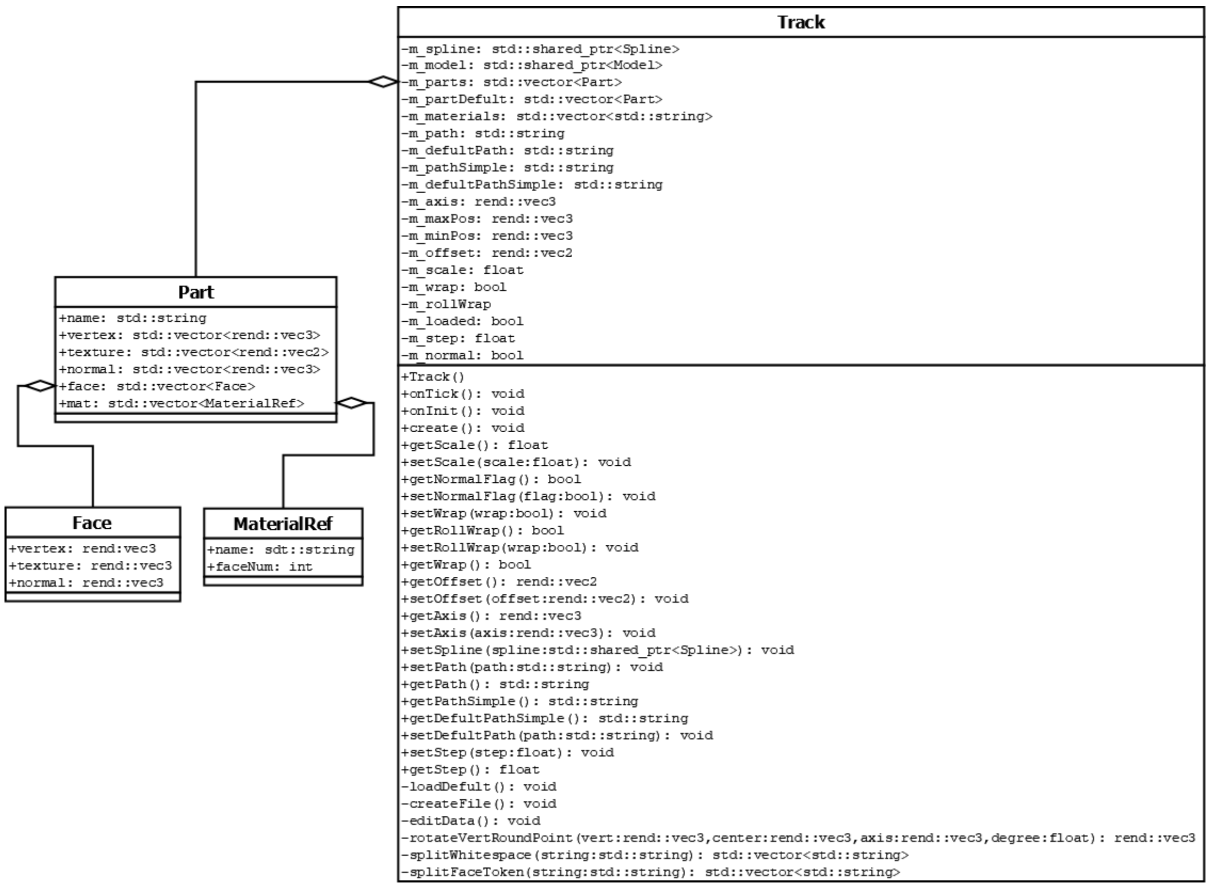

Track
-m_spline: std::shared_ptr<Spline>
-m_model: std::shared_ptr<Model>
-m_parts: std::vector<Part>
-m_partDefult: std::vector<Part>
-m_materials: std::vector<std::string>
-m_path: std::string
-m_defultPath: std::string
-m_pathSimple: std::string
-m_defultPathSimple: std::string
-m_axis: rend::vec3
-m_maxPos: rend::vec3
-m_minPos: rend::vec3
-m_offset: rend::vec2
-m_scale: float
-m_wrap: bool
-m_rollWrap
-m_loaded: bool
-m_step: float
-m_normal: bool
+Track()
+onTick(): void
+onInit(): void
+create(): void
+getScale(): float
+setScale(scale:float): void
+getNormalFlag(): bool
+setNormalFlag(flag:bool): void
+setWrap(wrap:bool): void
+getRollWrap(): bool
+setRollWrap(wrap:bool): void
+getWrap(): bool
+getOffset(): rend::vec2
+setOffset(offset:rend::vec2): void
+getAxis(): rend::vec3
+setAxis(axis:rend::vec3): void
+setSpline(spline:std::shared_ptr<Spline>): void
+setPath(path:std::string): void
+getPath(): std::string
+getPathSimple(): std::string
+getDefultPathSimple(): std::string
+setDefultPath(path:std::string): void
+setStep(step:float): void
+getStep(): float
-loadDefult(): void
-createFile(): void
-editData(): void
-rotateVertRoundPoint(vert:rend::vec3,center:rend::vec3,axis:rend::vec3,degree:float): rend::vec3
-splitWhitespace(string:std::string): std::vector<std::string>
-splitFaceToken(string:std::string): std::vector<std::string>
Part
+name: std::string
+vertex: std::vector<rend::vec3>
+texture: std::vector<rend::vec2>
+normal: std::vector<rend::vec3>
+face: std::vector<Face>
+mat: std::vector<MaterialRef>
Face
+vertex: rend:vec3
+texture: rend::vec3
+normal: rend::vec3
MaterialRef
+name: sdt::string
+faceNum: int

Fig.47: Basic track component before optimization

OBJ reader function

The algorithm shown in fig.48 reads every line in a .obj file saving and sorting the data in vector m_partsDefult as it goes. This vector stores the template model data without editing it so it can be used in the model mapping function. The algorithm has been written in a way that it can ignore invalid lines within the .obj file and save linked .mtl files in the vector m_materials, this means objects that have textures can keep their textures.

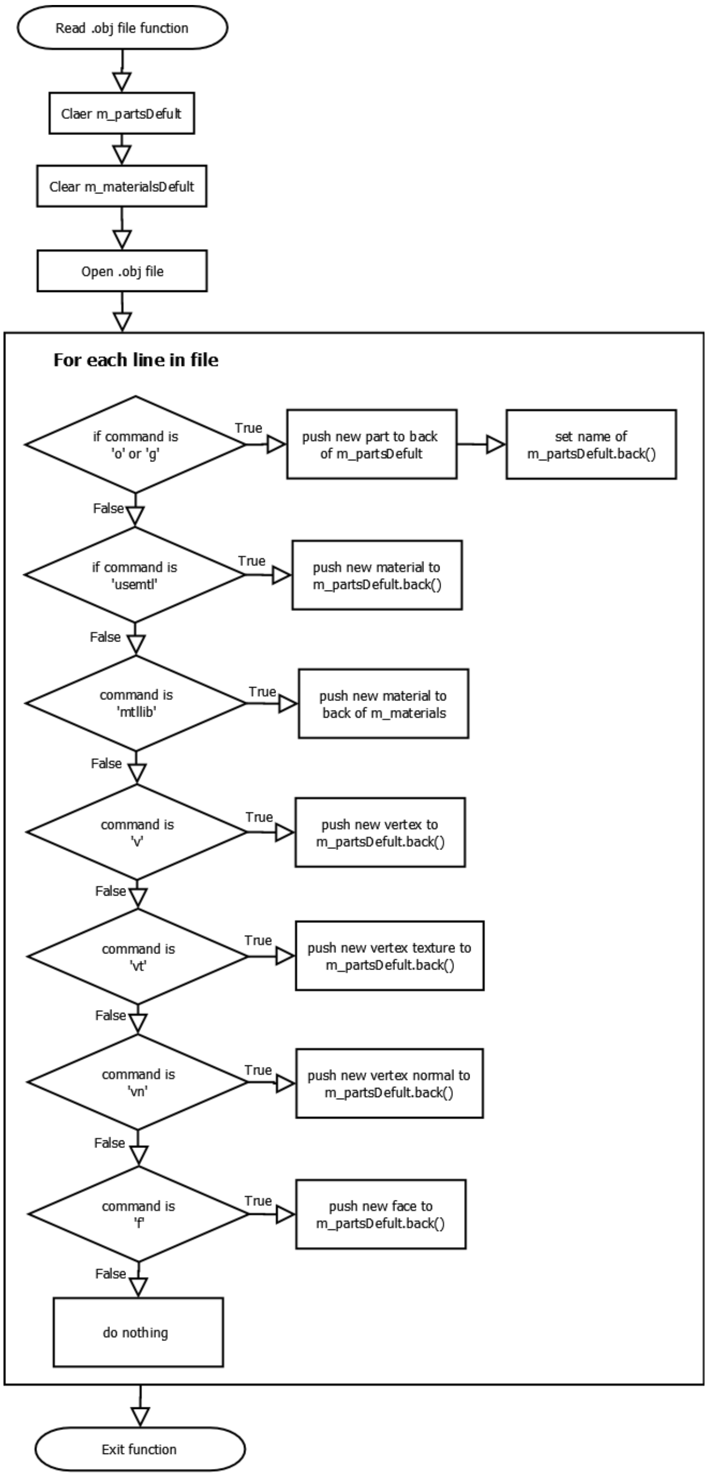
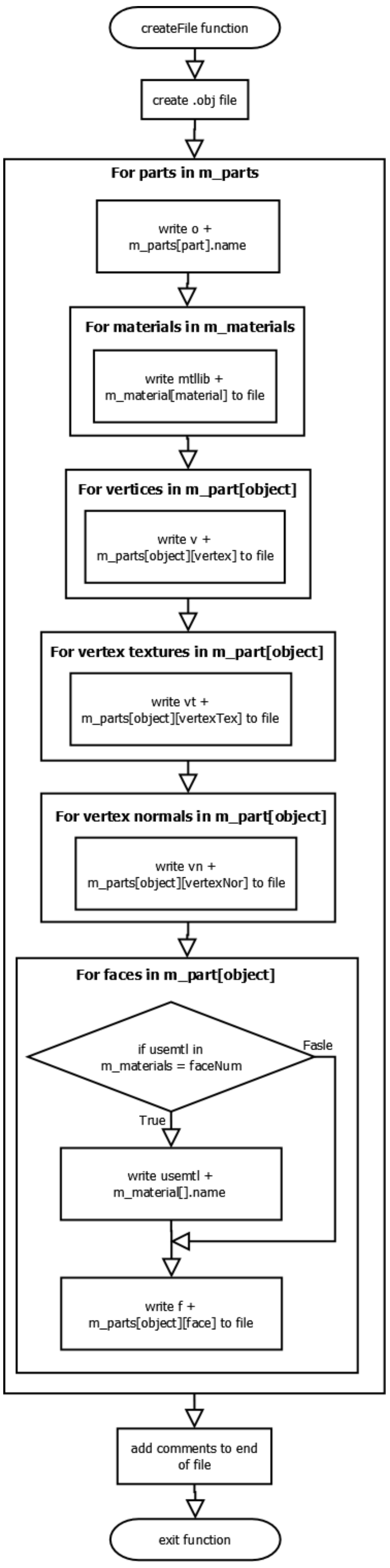

Read .obj file function
Claer m_partsDefult
Clear m_materialsDefult
Open .obj file
For each line in file
if command is 'o' or 'g'
True
push new part to back of m_partsDefult
set name of m_partsDefult.back()
False
if command is 'usemtl'
push new material to m_partsDefult.back()
command is 'mtllib'
push new material to back of m_materials
command is 'v'
push new vertex to m_partsDefult.back()
command is 'vt'
push new vertex texture to m_partsDefult.back()
command is 'vn'
push new vertex normal to m_partsDefult.back()
command is 'f'
push new face to m_partsDefult.back()
do nothing
Exit function

Fig.48: .obj reader function

OBJ writer function



createFile function
create .obj file
For parts in m_parts
write o + m_parts[part].name
For materials in m_materials
write mtllib + m_material[material] to file
For vertices in m_part[object]
write v + m_parts[object][vertex] to file
For vertex textures in m_part[object]
write vt + m_parts[object][vertexTex] to file
For vertex normals in m_part[object]
write vn + m_parts[object][vertexNor] to file
For faces in m_part[object]
if usemtl in m_materials = faceNum
Fasle
True
write usemtl + m_material[].name
write f + m_parts[object][face] to file
add comments to end of file
exit function

The .obj writing function is simpler than the .obj reading function as it just needs to write the data in m_parts into an empty file and add the correct keyword at the start of each line. Most of the hard work for this was done by the .obj reader function as it organized all of the inputted data making it easy for the whole component to keep the data organized. The function essentially goes through each model data type in m_parts and adds them to the new file one line at a time, as mentioned as m_parts is already organized going through it in order will guarantee the data written to the file will also be in the correct order without this function needing to edit the data in any way.

Fig.49: .obj writer function

Model Mapper Function

The model mapper function takes the data that is stored in the m_defultpart vector which holds all the data for the template track model and adds it to the back of the m_parts vector. This process is repeated, adding multiple of the template models to m_parts until the combined length of all the template model sections can reach the end of the spline. Well the function adds the data to m_parts, it edits the vertex data so that the different template models added get moved to different sections of the spline.

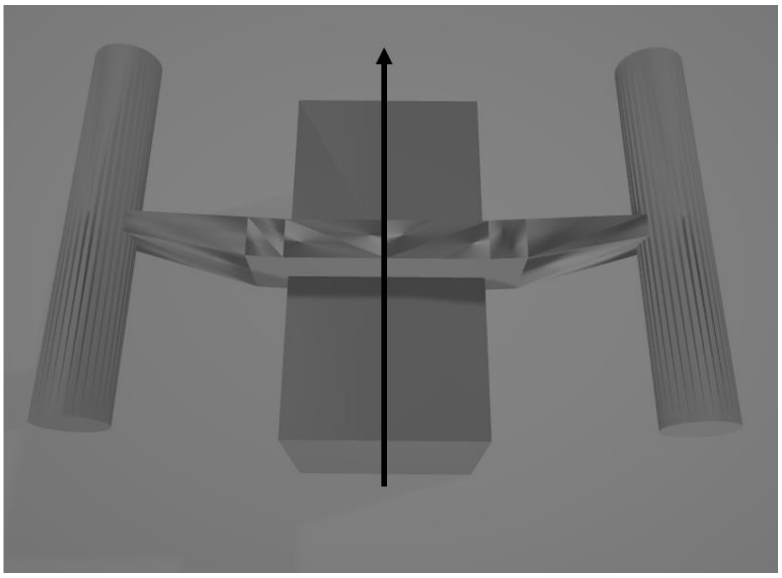


Fig.50: Template models length direction

For this explanation, assume the template model's length direction is along the z-axis (though x and z can be swapped if the length direction is x). To understand this process, think of the template model as only a list of individual vertices. These vertices can be thought of as vec3 objects that store three directionless distance values, instead of referring to these values as x, y and z, they will now be left, up and distance.

The first step in moving a vertex is to move its distance value along the spline curve using the step() function and get the world position of this point using the getPoint() function. This point is saved as a new vec3, referred to as the vertex's new position. Next, multiply the vertex's left value by the spline's normalized binormal at this point and add this value to the vertexes new position. At this stage, the template model is mapped along the spline on its x and z axis but remains flat s the y-axis still needs to be mapped.

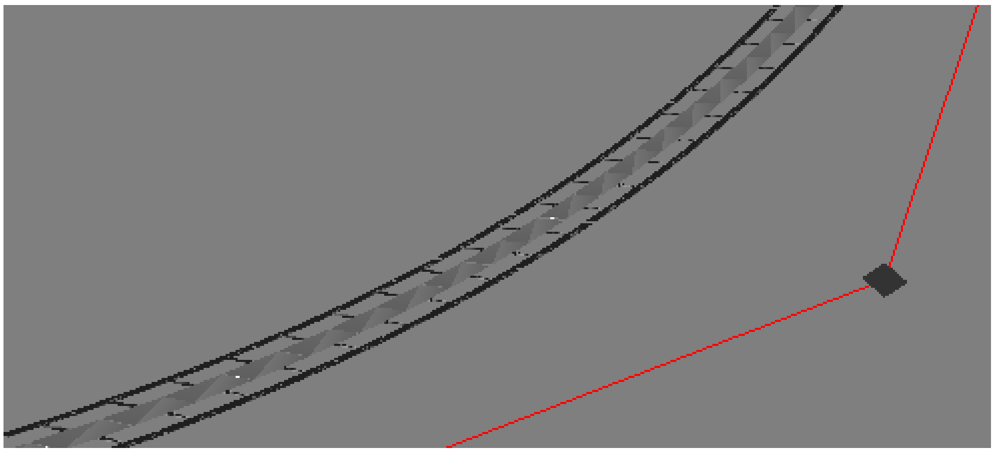


Fig.51: Template model mapped along a spline using x and z axis (without y)

To add height to the model the vertexes up value should be multiplied by the spline points normalized normal and added again to the vertexes new position. Now the template model is fully mapped along the spline and the vertexes new position can be saved in m_parts. This method of mapping the template model to the spline was developed from scratch through lots of trial and error.

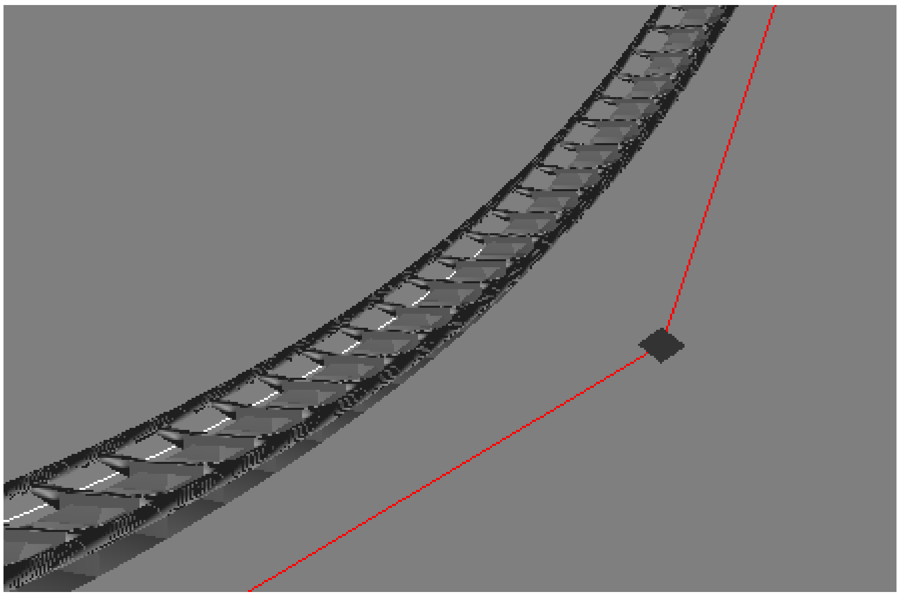


Fig.52: Template model mapped along a spline using all axis

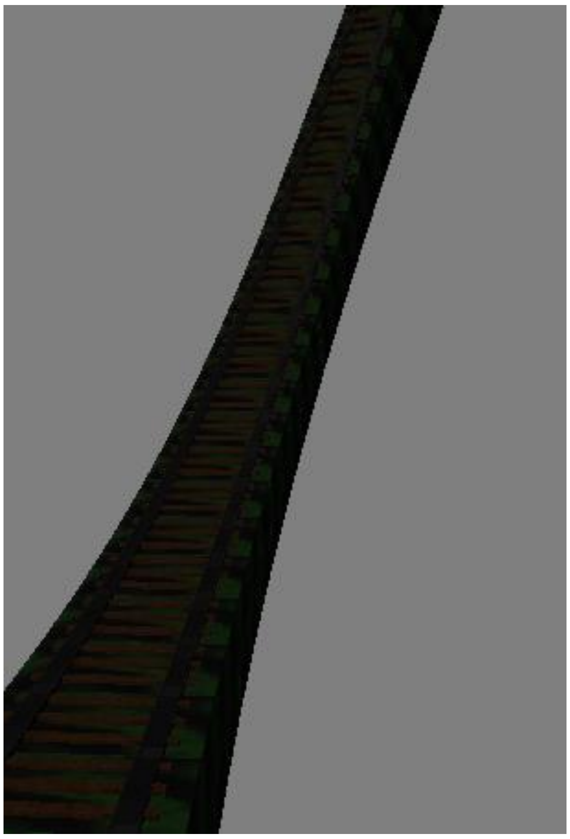


Fig.53: Template model (with texture) mapped along a spline using all axis

Track Creation Optimization

Now that the track model creator component has its functionality fully implemented, optimization to the generation time can be worked on. The largest slowdown is the spline resource's step() function, this function itself can't be made faster as it uses iteration and lowering the number of loops will result in a loss of accuracy. Therefore, all optimizations need to focus on reducing the number of times this function is called and shortening the distance that needs to be calculated by the function.

All optimization testing was done on a spline that needed a total of 250 model sections to construct a full spline model. The model used for testing was the RC track model shown in Fig.54, testing was done on a Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz (6 cores). The spline generation scales logarithmically without optimizations. The time taken to generate the model without optimizations is approximately 1 day.

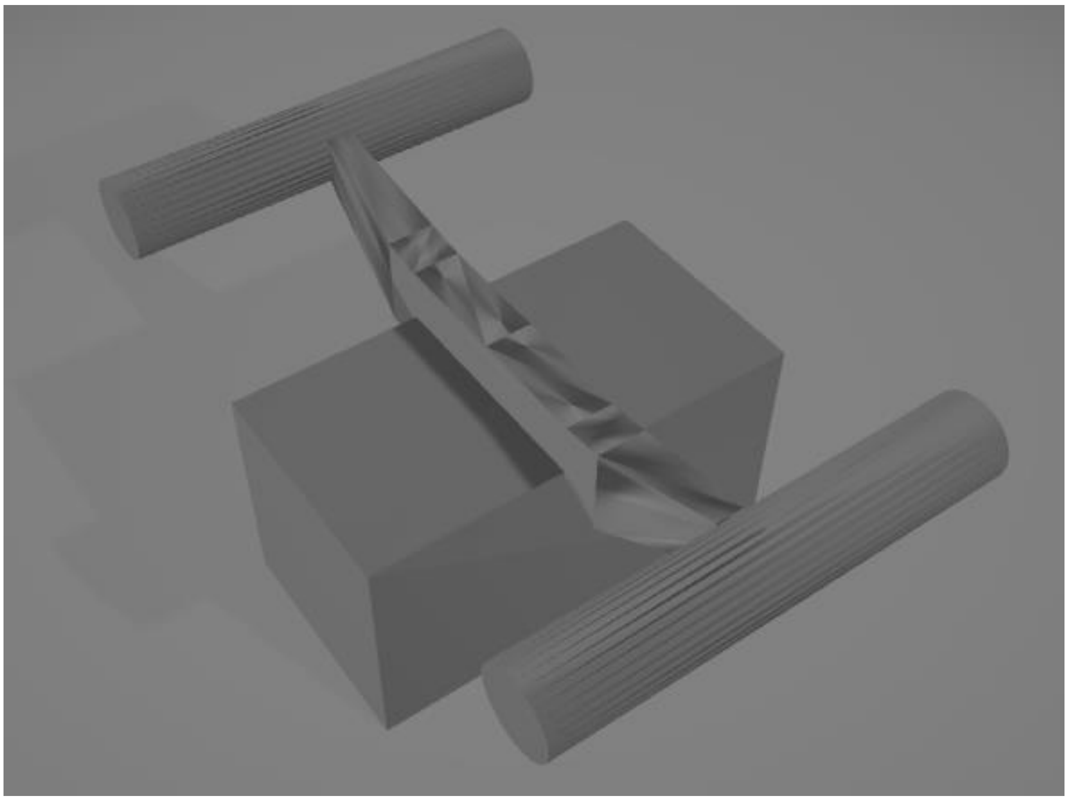


Fig.54: RC track section model

Distance Caching

As the length of the spline only effects the model along one axis there is no need to find the length along the spline for two vertexes that have the same value on this axis. Taking advantage of this, a distance caching system can be implemented that checks if a vertex value along the model's length axis as already been found and if so, skips finding the same value a second time. This reduces the number of step() calls from 168 to 6 per section, this dramatically reduces the model generation time, however it still takes over 55 minutes for a full spline model to generate.

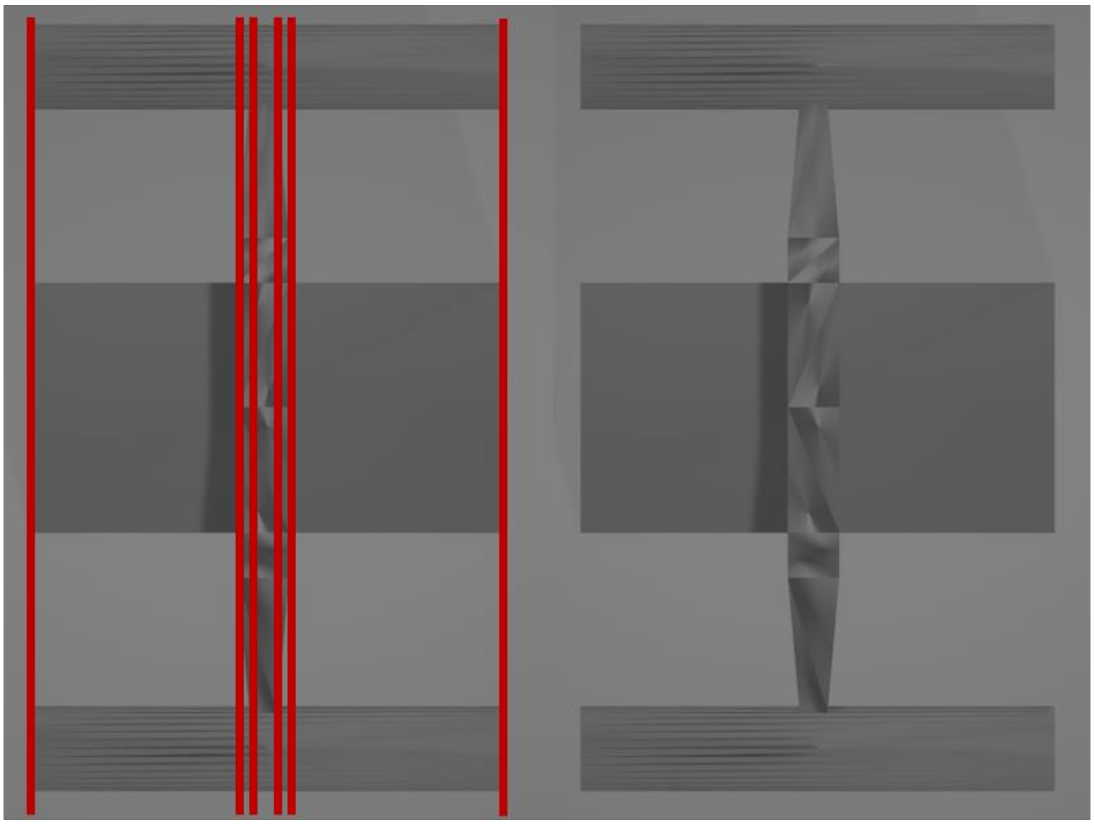


Fig.55: RC track model shown in Fig.54 with red line showing vertexes that have the same value along the length axis

Two stage step () per section

Currently the step() function is finding the distance between each vertex cache and the start of the spline, this causes sections further down the spline to need to constantly preform the step() function with large distance values. This can be decreased by doing a large step() to the start of the spline section and then preforming smaller step()s from this position to each individual vertex cache. This while adding an additional step() function call to each section decreases the overall distance calculated by all a section step() functions combined.

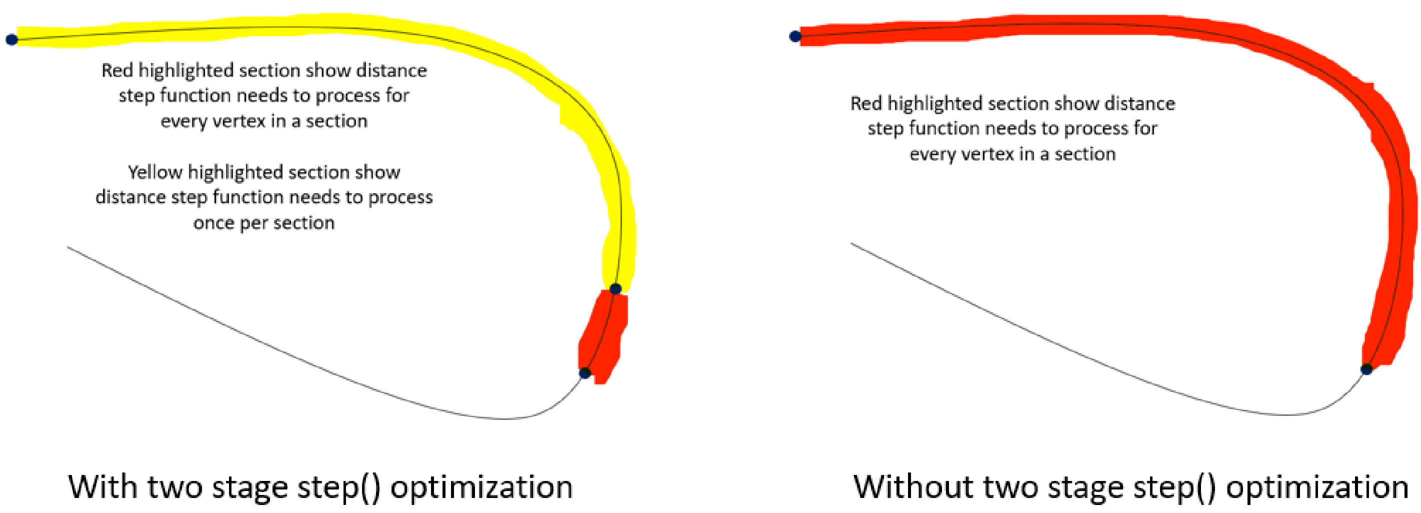

Red highlighted section show distance step function needs to process for every vertex in a section
Yellow highlighted section show distance step function needs to process once per section
With two stage step() optimization
Red highlighted section show distance step function needs to process for every vertex in a section
Without two stage step() optimization

Fig.56: Diagram showing sections of a spline the step() function needed to be ran on per section

Well, this optimization does decrease the model generation time to 2 minutes it also creates small gaps between sections of the model. This is caused as there are now two error values as a step() is being called from a step() position on the curve, when only one step() is used all vertexes have the same error value so the gaps are removed. To get around this the track model generator will have two modes:

- Fast – Has the gaps in the model, can be used by the user while editing the spline in the program for quick generation time.
- Detailed – Removes the gaps, but turns off two stage step() optimization, can be used when the final spline has been created by the user and they want an exact model output.

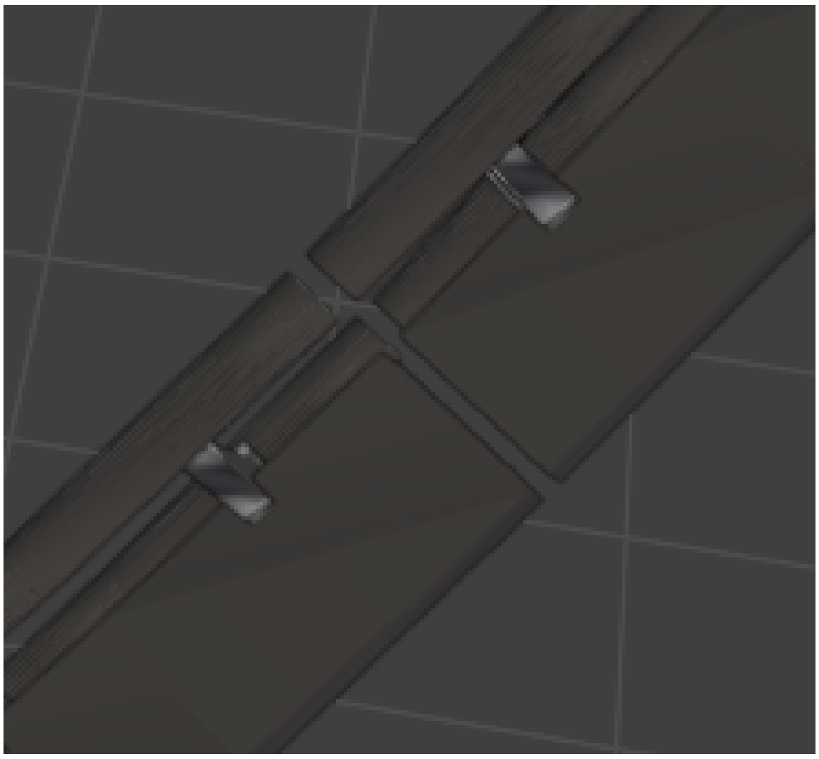


Fig.57: Gap between generated model sections when using two stage step() optimization (shown in blender)

Multithreading

The final way to optimize the model generation time is by adding multithreading so multiple sections can be generated at a time. This cannot be implemented by splitting the sections equally between each thread as each section takes a different amount of time to generate, this method would leave some threads finished while others would still have a large number of sections to generate. The way to get around this is creating a pool of sections, each thread can take one section from this pool at a time, this layout means that all the threads will always be generating a section until there are no sections left in the pool.

Testing was then performed to find the optimal number of threads, the CPU tested on had 6 cores and not surprisingly adding any more than six threads had very little impact on the overall generation time. This is useful as the program can check how many cores the machine it is running on has, then use that many threads for the generation, this means it auto-optimizes itself based on the machine it is running on.

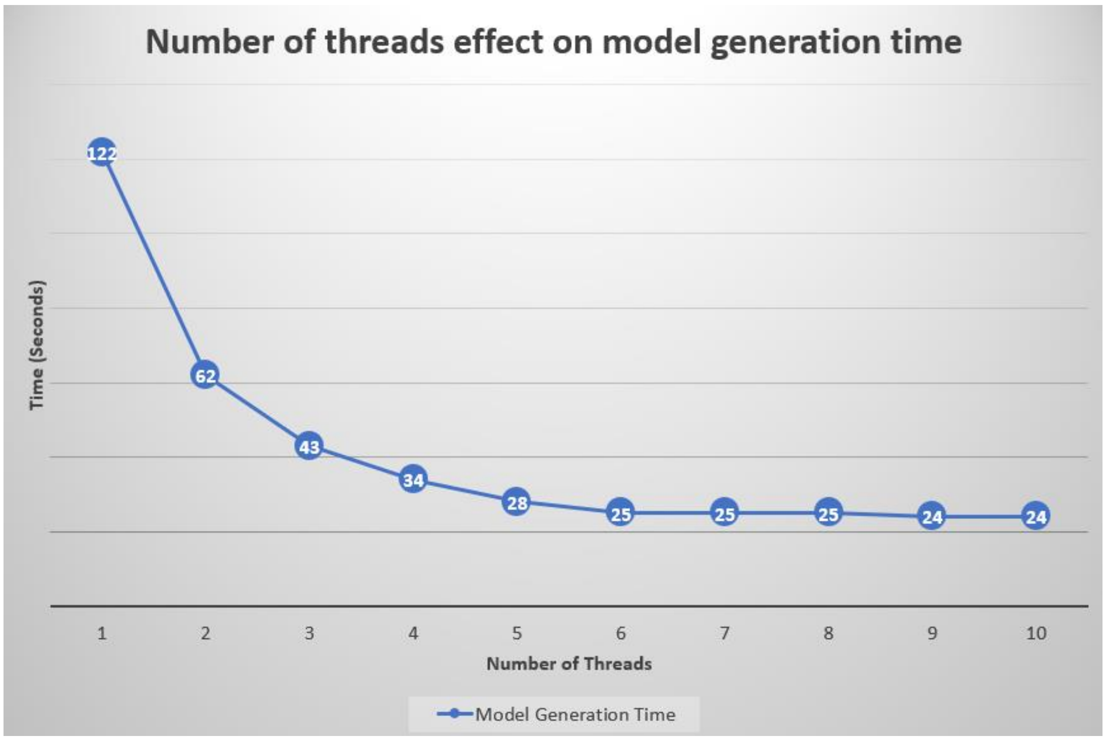


Fig.58: Diagram showing the effects different thread numbers had on model generation time (machine tested on had 6 cores and generation was set to fast mode)

With all of these optimizations the model can now be generated in 25 seconds in fast mode and 214 seconds in detailed mode, these times are acceptable for model generation.

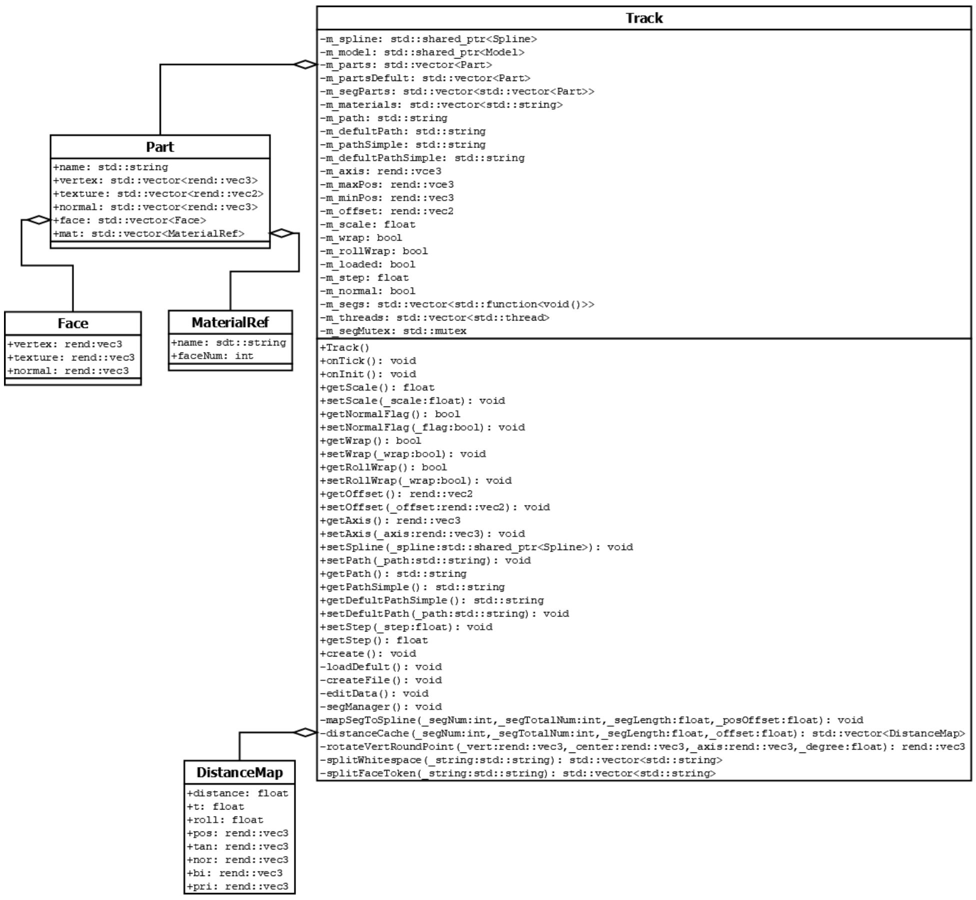
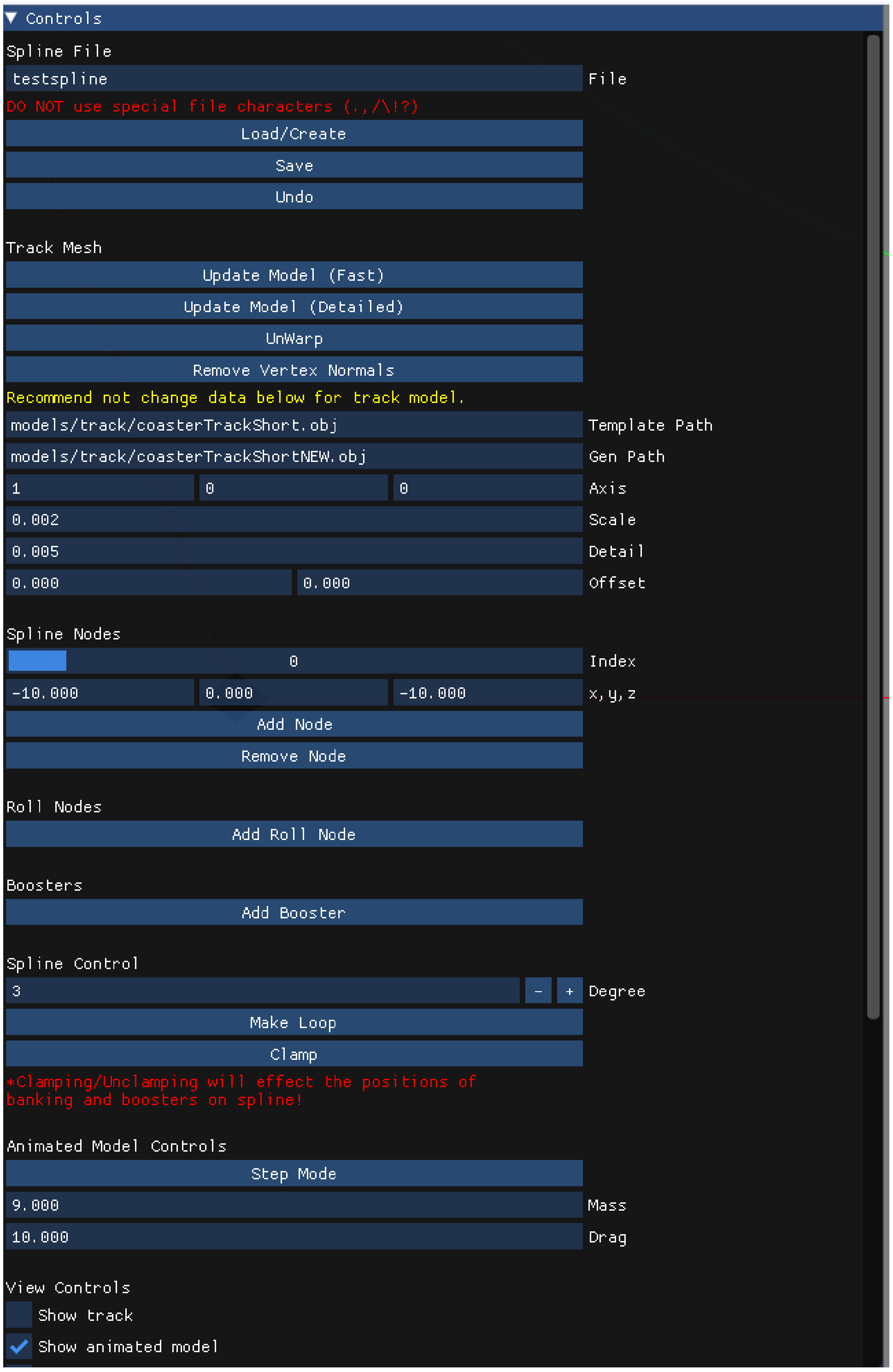

Track
-m_spline: std::shared_ptr<Spline>
-m_model: std::shared_ptr<Model>
-m_parts: std::vector<Part>
-m_partsDefult: std::vector<Part>
-m_segParts: std::vector<std::vector<Part>>
-m_materials: std::vector<std::string>
-m_path: std::string
-m_defultPath: std::string
-m_pathSimple: std::string
-m_defultPathSimple: std::string
-m_axis: rend::vce3
-m_maxPos: rend::vce3
-m_minPos: rend::vec3
-m_offset: rend::vec2
-m_scale: float
-m_wrap: bool
-m_rollWrap: bool
-m_loaded: bool
-m_step: float
-m_normal: bool
-m_segs: std::vector<std::function<void()>>
-m_threads: std::vector<std::thread>
-m_segMutex: std::mutex
+Track()
+onTick(): void
+onInit(): void
+getScale(): float
+setScale(_scale:float): void
+getNormalFlag(): bool
+setNormalFlag(_flag:bool): void
+getWrap(): bool
+setWrap(_wrap:bool): void
+getRollWrap(): bool
+setRollWrap(_wrap:bool): void
+getOffset(): rend::vec2
+setOffset(_offset:rend::vec2): void
+getAxis(): rend::vec3
+setAxis(_axis:rend::vec3): void
+setSpline(_spline:std::shared_ptr<Spline>): void
+setPath(_path:std::string): void
+getPath(): std::string
+getPathSimple(): std::string
+getDefultPathSimple(): std::string
+setDefultPath(_path:std::string): void
+setStep(_step:float): void
+getStep(): float
+create(): void
-loadDefult(): void
-createFile(): void
-editData(): void
-segManager(): void
-mapSegToSpline(_segNum:int,_segTotalNum:int,_segLength:float,_posOffset:float): void
-distanceCache(_segNum:int,_segTotalNum:int,_segLength:float,_offset:float): std::vector<DistanceMap>
-rotateVertRoundPoint(_vert:rend::vec3,_center:rend::vec3,_axis:rend::vec3,_degree:float): rend::vec3
-splitWhitespace(_string:std::string): std::vector<std::string>
-splitFaceToken(_string:std::string): std::vector<std::string>
Part
+name: std::string
+vertex: std::vector<rend::vec3>
+texture: std::vector<rend::vec2>
+normal: std::vector<rend::vec3>
+face: std::vector<Face>
+mat: std::vector<MaterialRef>
Face
+vertex: rend:vec3
+texture: rend::vec3
+normal: rend::vec3
MaterialRef
+name: sdt::string
+faceNum: int
DistanceMap
+distance: float
+t: float
+roll: float
+pos: rend::vec3
+tan: rend::vec3
+nor: rend::vec3
+bi: rend::vec3
+pri: rend::vec3

Fig.59 Final track component implemented

UI

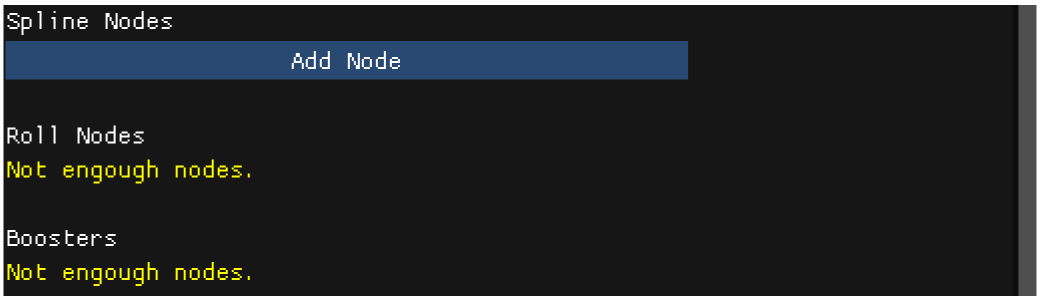
A menu component needs to be implemented in the project to provide a GUI to the user to enable them to interact with the spline resource and other components in the project. This GUI will use the ImGui library as it provides elements such as sliders, checkboxes, color input, text input and number input. The Menu component itself will have no complex logic within it and instead serve as a connection between the user and all the other components, which will be done by setting/getting data from other components API functions.



```
▼ Controls
Spline File
testspline                                    File
DO NOT use special file characters (.,/\!?)
Load/Create
Save
Undo

Track Mesh
Update Model (Fast)
Update Model (Detailed)
UnWarp
Remove Vertex Normals
Recommend not change data below for track model.
models/track/coasterTrackShort.obj            Template Path
models/track/coasterTrackShortNEW.obj         Gen Path
1              0              0               Axis
0.002                                         Scale
0.005                                         Detail
0.000                    0.000                Offset

Spline Nodes
                      0                       Index
-10.000        0.000          -10.000         x,y,z
Add Node
Remove Node

Roll Nodes
Add Roll Node

Boosters
Add Booster

Spline Control
3                                     - +     Degree
Make Loop
Clamp
*Clamping/Unclamping will effect the positions of
banking and boosters on spline!

Animated Model Controls
Step Mode
9.000                                         Mass
10.000                                        Drag

View Controls
☐ Show track
✓ Show animated model
```

Fig.60: Part of the GUI implemented in the program

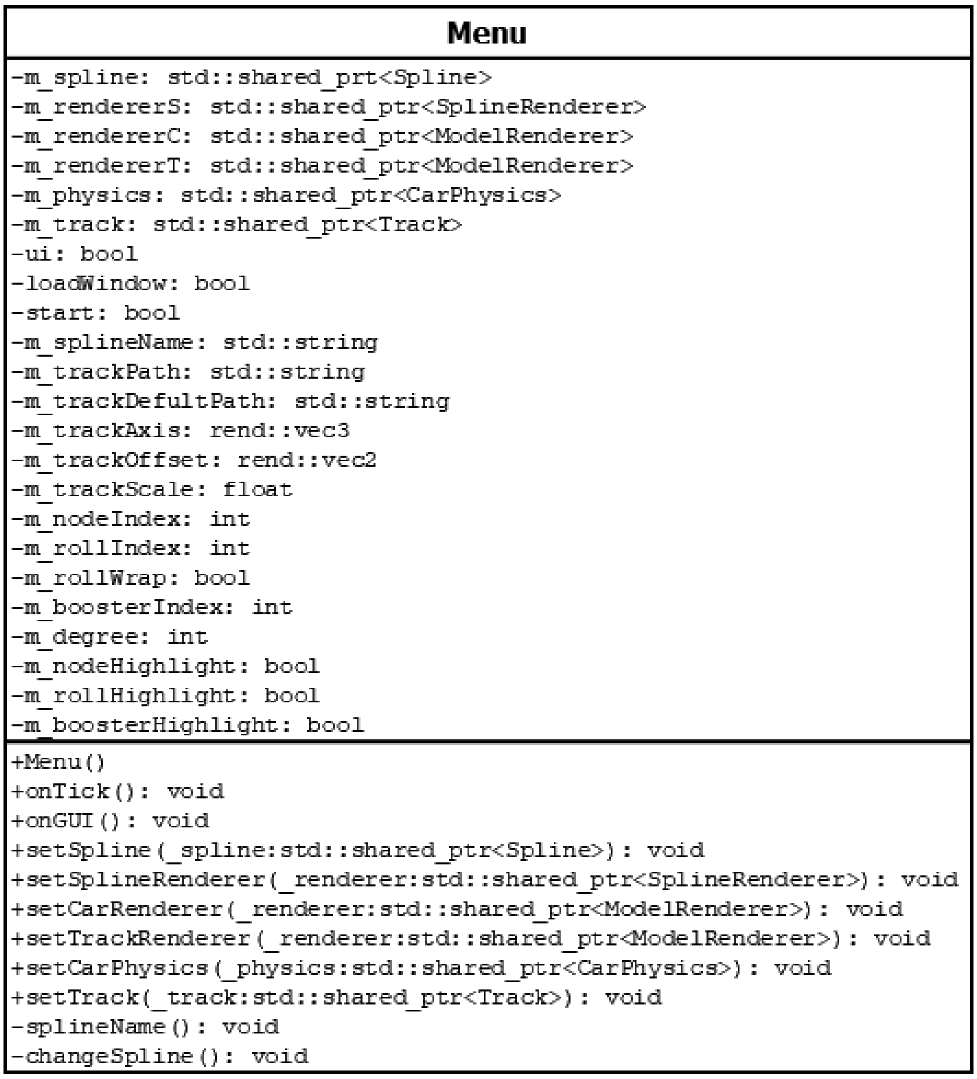
Some elements of the GUI are adaptive and will show and hide themselves depending on the state of connected components, this is done to make it clear to the user what they have access to with the current state of the program and can prevent them entering invalid values into functions. An example of this is removing the option to remove a node if there are currently no nodes in the spline m_node vector.


Spline Nodes
Add Node

Roll Nodes
Not engough nodes.

Boosters
Not engough nodes.

Fig.61: Element of the GUI adapting to spline currently having no nodes



```
Menu
-m_spline: std::shared_prt<Spline>
-m_rendererS: std::shared_ptr<SplineRenderer>
-m_rendererC: std::shared_ptr<ModelRenderer>
-m_rendererT: std::shared_ptr<ModelRenderer>
-m_physics: std::shared_ptr<CarPhysics>
-m_track: std::shared_ptr<Track>
-ui: bool
-loadWindow: bool
-start: bool
-m_splineName: std::string
-m_trackPath: std::string
-m_trackDefultPath: std::string
-m_trackAxis: rend::vec3
-m_trackOffset: rend::vec2
-m_trackScale: float
-m_nodeIndex: int
-m_rollIndex: int
-m_rollWrap: bool
-m_boosterIndex: int
-m_degree: int
-m_nodeHighlight: bool
-m_rollHighlight: bool
-m_boosterHighlight: bool
+Menu()
+onTick(): void
+onGUI(): void
+setSpline(_spline:std::shared_ptr<Spline>): void
+setSplineRenderer(_renderer:std::shared_ptr<SplineRenderer>): void
+setCarRenderer(_renderer:std::shared_ptr<ModelRenderer>): void
+setTrackRenderer(_renderer:std::shared_ptr<ModelRenderer>): void
+setCarPhysics(_physics:std::shared_ptr<CarPhysics>): void
+setTrack(_track:std::shared_ptr<Track>): void
-splineName(): void
-changeSpline(): void
```

Fig.62: Menu Component

Testing

Once all the elements of the program had been implemented the program was tested, a range of test were preformed to confirm all areas of the project are working as intended.

General Testing

General functionality testing was performed on most key components/resources in the project.

Test ID	Testing	Results	Comment
1	Adding node	Success	
2	Removing node	Success	
3	Changing degree	Success	
5	Clamping	Success	
6	Unclamping	Success	
7	Adding roll node	Success	
8	Removing roll node	Success	
9	Adding booster	Success	
10	Removing booster	Success	
11	Saving Spline	Fail	File created had incorrect file name
12	Saving Spline	Success	File now creates with correct name
13	Load Spline	Success	
14	get point on spline	Success	
15	Get velocity of point on spline	Success	
16	Get roll of point on spline	Success	
17	Get boost of point on spline	Success	
18	Make loop	Success	
19	Get length	Success	
20	Step along spline	Success	

Fig.63: General spline tests

Test ID	Testing	Results	Comment
21	Changing Scale	Success	
22	Changing template axis	Success	
23	Wrapping	Success	
24	Unwrapping	Success	
25	Set offset	Success	
26	Fast generation	Success	
27	Slow generation	Success	

Fig.64: General model generator tests

Test ID	Testing	Results	Comment
28	Model match spline tangent	Success	
29	Model match spline banking	Success	
30	Model move at constant speed along spline	Success	
31	Model use physics to move along spline	Success	
32	Model physics correctly effected by spline boosters	Success	

Fig.65: General animated model tests

Test ID	Testing	Results	Comment
33	Opening and Closing	Success	
34	All adaptive elements adapting to program status correcly	Success	
35	All text inputs	Success	
36	All int inputs	Success	
37	All float inputs	Success	
38	All int slider inputs	Fail	Two sliders had the same ImGui ID
39	All int slider inputs	Success	Sliders ID's were fixed
40	All float splider inputs	Success	
41	All buttons	Success	
42	All checkboxs	Success	
43	All colour inputs	Success	
44	Confirm popup menu	Success	

Fig.66: General UI tests

Model Testing

To test that the .obj files that the model generator created were high quality some were imported into blender to run more detailed tests on the meshes. These tests included testing for holes in the mesh, checking total number of vertexes and faces, and general appearance checks. The .obj files tested passed all of these tests.

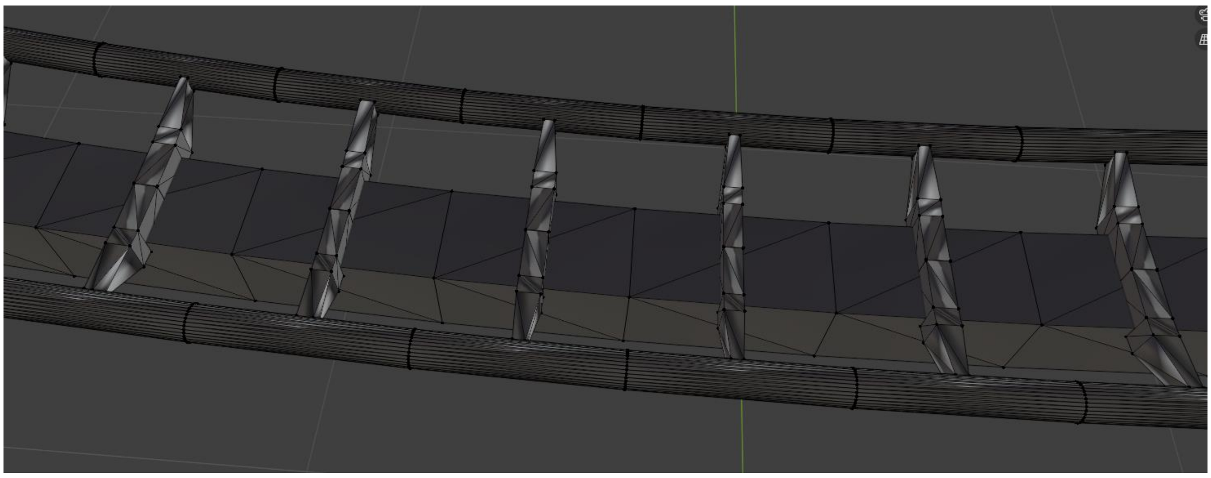
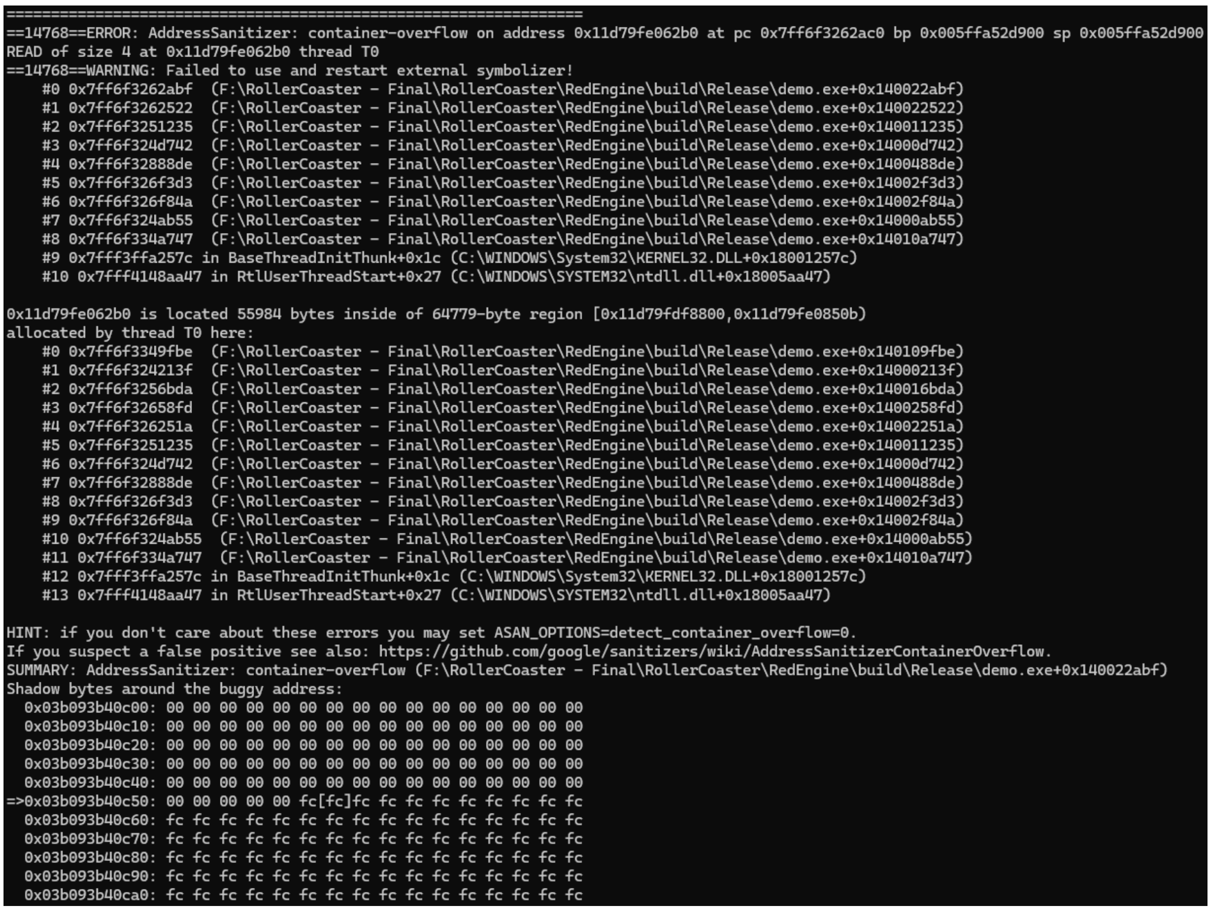


Fig.67: Track model being tested in blender

Address Sanitizer Testing

The project was run with an address sanitizer turned on to perform more detailed memory testing to confirm that the program was memory safe. This found a small bug inside the spline addNode() function where the functions input validation was incorrect, this was then patched and the test repeated, this time the test succeeded.



```
=================================================================
==14768==ERROR: AddressSanitizer: container-overflow on address 0x11d79fe062b0 at pc 0x7ff6f3262ac0 bp 0x005ffa52d900 sp 0x005ffa52d900
READ of size 4 at 0x11d79fe062b0 thread T0
==14768==WARNING: Failed to use and restart external symbolizer!
    #0 0x7ff6f3262abf  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x140022abf)
    #1 0x7ff6f3262522  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x140022522)
    #2 0x7ff6f3251235  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x140011235)
    #3 0x7ff6f324d742  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14000d742)
    #4 0x7ff6f32888de  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x1400488de)
    #5 0x7ff6f326f3d3  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14002f3d3)
    #6 0x7ff6f326f84a  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14002f84a)
    #7 0x7ff6f324ab55  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14000ab55)
    #8 0x7ff6f334a747  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14010a747)
    #9 0x7fff3ffa257c in BaseThreadInitThunk+0x1c (C:\WINDOWS\System32\KERNEL32.DLL+0x18001257c)
    #10 0x7fff4148aa47 in RtlUserThreadStart+0x27 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x18005aa47)

0x11d79fe062b0 is located 55984 bytes inside of 64779-byte region [0x11d79fdf8800,0x11d79fe0850b)
allocated by thread T0 here:
    #0 0x7ff6f3349fbe  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x140109fbe)
    #1 0x7ff6f324213f  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14000213f)
    #2 0x7ff6f3256bda  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x140016bda)
    #3 0x7ff6f32658fd  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x1400258fd)
    #4 0x7ff6f326251a  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14002251a)
    #5 0x7ff6f3251235  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x140011235)
    #6 0x7ff6f324d742  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14000d742)
    #7 0x7ff6f32888de  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x1400488de)
    #8 0x7ff6f326f3d3  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14002f3d3)
    #9 0x7ff6f326f84a  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14002f84a)
    #10 0x7ff6f324ab55  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14000ab55)
    #11 0x7ff6f334a747  (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x14010a747)
    #12 0x7fff3ffa257c in BaseThreadInitThunk+0x1c (C:\WINDOWS\System32\KERNEL32.DLL+0x18001257c)
    #13 0x7fff4148aa47 in RtlUserThreadStart+0x27 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x18005aa47)

HINT: if you don't care about these errors you may set ASAN_OPTIONS=detect_container_overflow=0.
If you suspect a false positive see also: https://github.com/google/sanitizers/wiki/AddressSanitizerContainerOverflow.
SUMMARY: AddressSanitizer: container-overflow (F:\RollerCoaster - Final\RollerCoaster\RedEngine\build\Release\demo.exe+0x140022abf)
Shadow bytes around the buggy address:
  0x03b093b40c00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x03b093b40c10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x03b093b40c20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x03b093b40c30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x03b093b40c40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x03b093b40c50: 00 00 00 00 00 fc[fc]fc fc fc fc fc fc fc fc fc
  0x03b093b40c60: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
  0x03b093b40c70: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
  0x03b093b40c80: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
  0x03b093b40c90: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
  0x03b093b40ca0: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
```

Fig.68: Address sanitizer output in failed memory test

Performance Testing

The programs performance was monitored while preforming the other tests, its framerate never fell below 30fps, this is considered good performance. For more detail on model generation performance check the track creation optimization section.

Project UML

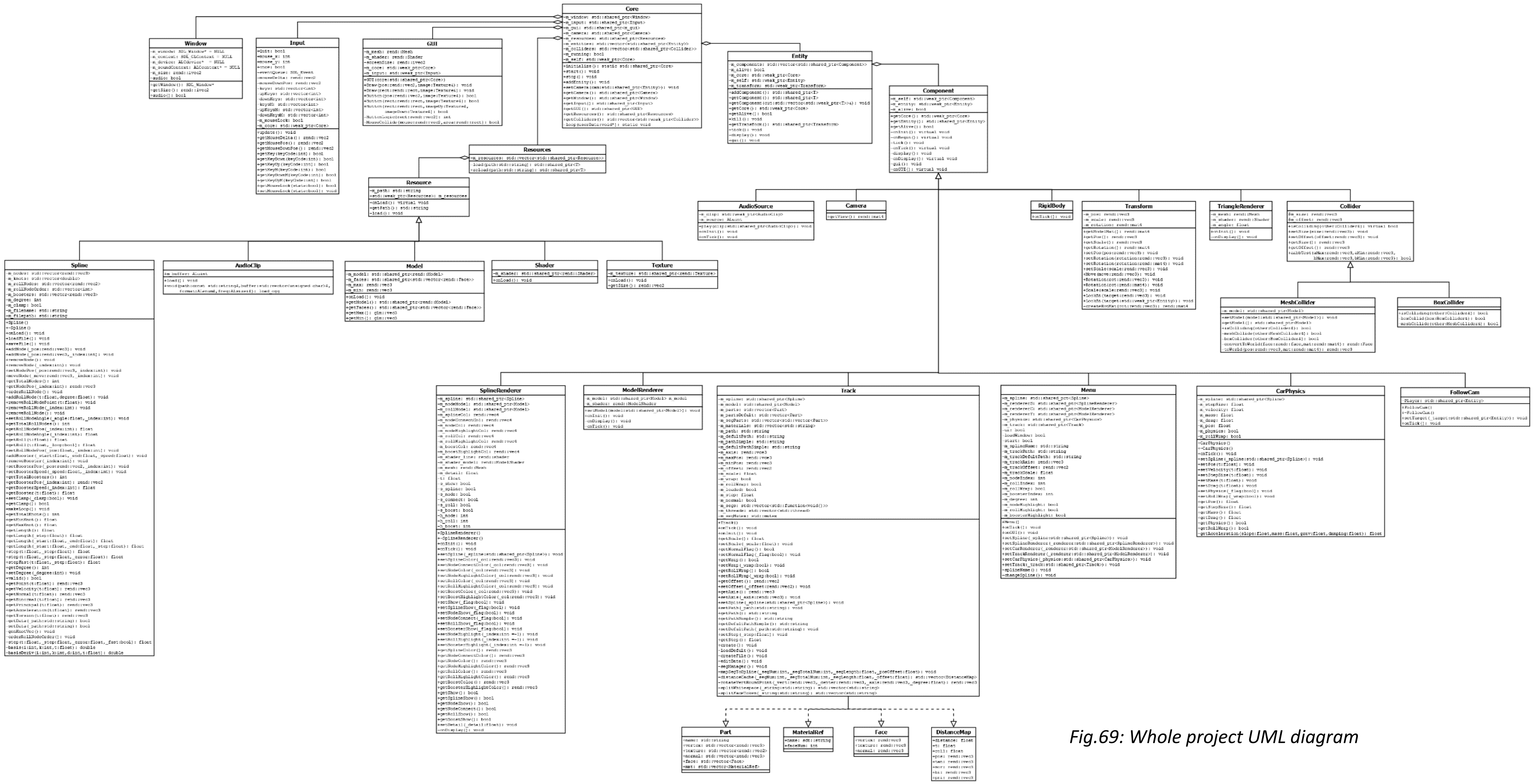
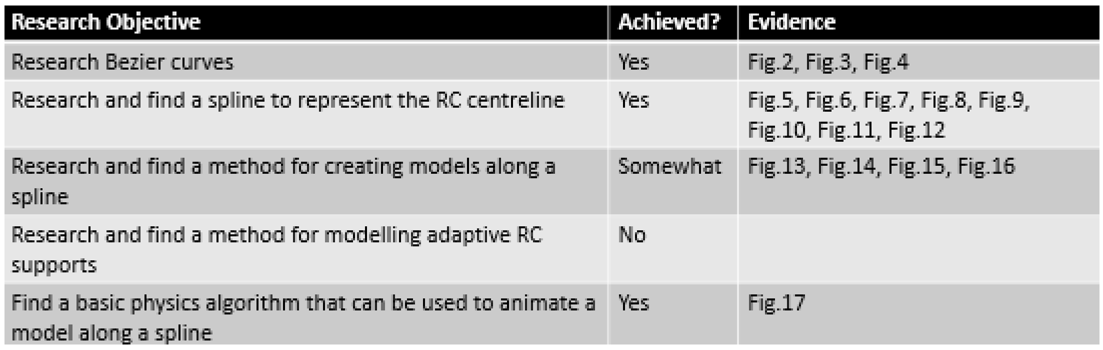


Fig.69: Whole project UML diagram

Evaluation

Research

The research conducted at the start of this project on spline and Bezier curves was more than sufficient as it provided me with all the knowledge needed for the implementation phase. I critically engage with multiple sources, comparing them together to check that information was accurate. On the other hand, the modelling research was not sufficient as many of the key formulas were not found until late in the implementation phase and even then, most of them were self-developed. More time should have been spent on this part of the project.



Research Objective	Achieved?	Evidence
Research Bezier curves	Yes	Fig.2, Fig.3, Fig.4
Research and find a spline to represent the RC centreline	Yes	Fig.5, Fig.6, Fig.7, Fig.8, Fig.9, Fig.10, Fig.11, Fig.12
Research and find a method for creating models along a spline	Somewhat	Fig.13, Fig.14, Fig.15, Fig.16
Research and find a method for modelling adaptive RC supports	No	
Find a basic physics algorithm that can be used to animate a model along a spline	Yes	Fig.17

Fig.70: Research objectives

Implementation

Due to the technical nature of this project, no surveys/real user testing was carried out, this is because the goal of the project as laid out at the beginning was to create a program that could be used for modeling RCs. The most effective method for evaluating if this goal and other implementation goals had been met was preforming pacific feature testing to ensure that everything that was implemented was working as planned. I feel like this was the correct method of testing to use as I have deep knowledge of how the program works, so I would also know how to best break it and which parts of the project are the most delicate. On the other hand, user testing could have provided a new train of thought in the testing process and discovered more obscure bugs.

Although I feel the correct testing method was used, some more detailed tests, such as for performance could have been useful, this didn't happen as I wanted to maximize the total number of tests preformed rather than doing fewer, but more detailed test.

Implementation Objective	Achieved?	Evidence
Spline		
Create a basic spline resource	Yes	Fig.35, Fig.36
Add banking/roll functionality to spline	Yes	Fig.42
Add booster functionality to spline	Yes	Fig.43
Add file save/load system to spline	Yes	Fig.40
Test spline's functionality	Yes	Fig.63
Create spline renderer	Yes	Fig.35, Fig.36
Add model renderer to spline renderer (e.g. banking node markers)	Yes	Fig.42
Model Creator		
Create spline model creator component that can create a model along a spline	Yes	Fig.52, Fig.53
Add ability to change template model to spline model creator	Yes	Fig,52, Fig.53
Add ability to create RC supports to spline model creator	No	
Have spline model creator have fast model generation times	Yes	Fig.58
Test .obj files created by spline model generator for holes	Yes	Fig.57, Fig.67
Test spline model creator's functionality	Yes	Fig.64
Animated Model		
Create a basic animated model component that can move along a spline	Yes	Fig.46
Add basic physics to animated model	Yes	Fig.45
Test animated model	Yes	Fig.65
UI		
Create a UI component that lets user interact with all components/resources listed above	Yes	Fig.60, Fig.61
Test UI	Yes	Fig.66
Testing		
Project memory testing	Yes	Fig.68
Project performance testing	Somewhat	Fig.58

Fig.71: Implementation objectives

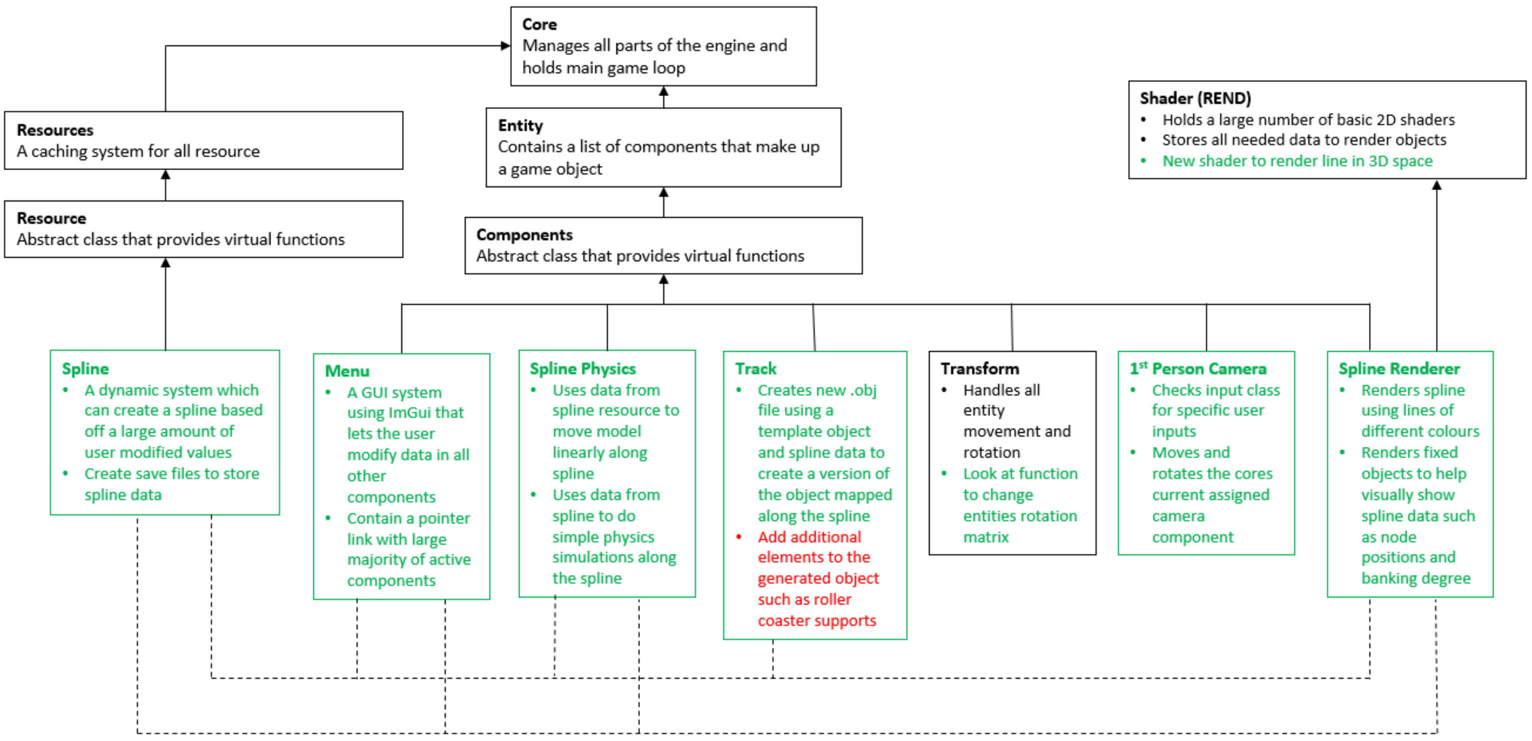

Core
Manages all parts of the engine and holds main game loop
Entity
Contains a list of components that make up a game object
Components
Abstract class that provides virtual functions
Resources
A caching system for all resource
Resource
Abstract class that provides virtual functions
Shader (REND)
Holds a large number of basic 2D shaders
Stores all needed data to render objects
New shader to render line in 3D space
Spline
A dynamic system which can create a spline based off a large amount of user modified values
Create save files to store spline data
Menu
A GUI system using ImGui that lets the user modify data in all other components
Contain a pointer link with large majority of active components
Spline Physics
Uses data from spline resource to move model linearly along spline
Uses data from spline to do simple physics simulations along the spline
Track
Creates new .obj file using a template object and spline data to create a version of the object mapped along the spline
Add additional elements to the generated object such as roller coaster supports
Transform
Handles all entity movement and rotation
Look at function to change entities rotation matrix
1st Person Camera
Checks input class for specific user inputs
Moves and rotates the cores current assigned camera component
Spline Renderer
Renders spline using lines of different colours
Renders fixed objects to help visually show spline data such as node positions and banking degree

Fig.72: Planned additions/modifications to the engine (and planned data flow), updated version of Fig.22

Conclusion

Overall, this project can be considered a success, as most research and implementation objectives were achieved, the program meets its purpose and can generate RC models using splines effectively. Although the objectives related to RC support modeling were not met, this feature had to be dropped due to time constraints. Since it was a non-crucial, complex element with no direct connections to other components, it was the logical choice to drop this element over others. These time constraints stemmed from the spline model generator's performance optimization taking much longer than anticipated to successfully implement, this caused other delays in the project leading to a development crunch period leading up to the project's deadline.

In addition to better time management, I could have conducted more thorough research on creating models along a spline before starting development. Although I did considerable research in this area at the beginning of the project, I only discovered many of the key algorithms and functions during the implementation phase. I should have conducted deeper research instead of spending most of my time learning the general theory of creating models along splines.

One aspect of the project that went well was keeping the code base well organized and well commented throughout development, this was the largest code base I have worked on and is built on top of a previous assignment's code base. Maintaining organization will streamline further development on the project even if the person doing this is unfamiliar with the code base.

This project has significantly improved my knowledge of splines, which I had only used in software like photoshop before starting research on them, this knowledge provides a solid foundation for exploring spline surfaces for use in future projects. Additionally, it has enhanced my understanding of c++ development, particularly in reading and writing to different file types, this will be invaluable for further improving my overall c++ skills.

A key area for future improvement would be implementing the RC support model creator that was dropped due to time constraints, this would make the RC models look more realistic. Additionally, implementing an RC dynamics simulation such as the algorithms provided in "Kinematic and Dynamics Simulation Research for Roller Coaster Multi-body System", Xu 2012, as this could improve the RC train animation.

References

Hunt, K, 2018., *Design Analysis of Roller Coasters*. Thesis (Masters). Worcester Polytechnic Institute.

He, X.Y., He, H.W. and Hu, Z.Y., 2015. Modeling and Simulation of a roller coaster. *Applied Mechanics and Materials*, *719*, pp.262-267.

Piegl, L. and Tiller, W., 1997. *The NURBS Book*. 2nd ed. Springer Science & Business Media

University of Cambridge, 1999. *Bezier curves*. Available from: www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/ [Accessed 6 February 2024]

Lyche, T. and Morken, K., 2008. Spline methods draft. *Department of Informatics, Center of Mathematics for Applications.,* University of Oslo*,*

Holmér, F. 2022. *The Continuity of Splines* [video]. YouTube. Available from: https://www.youtube.com/watch?v=jvPPXbo87ds&t=3242s&ab_channel=FreyaHolm%C3%A9r [Accessed 1 February 2024].

University of Clemons, 2015, *Spline Curves*. Available from: https://people.computing.clemson.edu/~dhouse/courses/405/notes/splines.pdf [Accessed 15 March 2024]

Pombo, J. and Ambrósio, J., 2007. Modelling tracks for roller coaster dynamics. *International journal of vehicle design*, *45*(4), pp.470-500.

University of Cambridge, 1999. *B-splines*. Available from: www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/ [Accessed 7 February 2024]

Prautzsch, H., Boehm, W. and Paluszny, M., 2002. *Bézier and B-spline techniques* (Vol. 6). Berlin: Springer.

Joy, K.I., 2008. Definition of a B-spline curve. *University of California Davis, [Online]. Available:* https://wwwx.cs.unc.edu/~dm/UNC/COMP258/LECTURES/B-spline.pdf [Accessed February 10]

Elbanhawi, M., Simic, M. and Jazar, R.N., 2015. Continuous path smoothing for car-like robots using B-spline curves. *Journal of Intelligent & Robotic Systems*, *80*, pp.23-56.

Holmér, J. 2015. A coder's guide to spline-based procedural geometry. *In: Unite 2015 Boston*. John Hynes veterans memorial convention center 21-23 September 2015. Available from: https://www.youtube.com/watch?v=o9RK6O2kOKo&ab_channel=Unity

Department of Mathematics, University of Texas. 2016. *Finding the normal of a plane*. Available from: https://web.ma.utexas.edu/users/m408m/Display12-5-4.shtml. [Accessed 4 march 2024]

Blender. 2024. *Blender 2.80 Manual - Making Edge/Face*. Available from: https://docs.blender.org/manual/en/2.80/modeling/meshes/editing/basics/make_face_edge.html [Accessed 6 march 2024]

Zill, D. and Dewar, J., 2011. *Algebra and trigonometry*. Jones & Bartlett Publishers.

myPhysicsLab.com. 2023. *Roller Coaster*. Available from: https://www.myphysicslab.com/roller/roller-single-en.html [Accessed 12 march 2024]

Prenter, P.M., 2008. *Splines and variational methods*. Courier Corporation.

Rogers, D.F., 2001. *An introduction to NURBS: with historical perspective*. Morgan Kaufmann.

Bazett, T. 2019. *How long is a curve??? The Arclength Formula in 3D* [video]. YouTube. Available from: https://www.youtube.com/watch?v=80J5s0pic8M&t=314s&ab_channel=Dr.TreforBazett [Accessed 3 April 2024]

Wang, H., Kearney, J. and Atkinson, K., 2002, June. Arc-length parameterized spline curves for real-time simulation. In *Proc. 5th International Conference on Curves and Surfaces* (Vol. 387396).

Lin, A., 2019. Binary search algorithm. *WikiJournal of Science*, *2*(1), pp.1-13.

Noor, N.M., Al Bakri Abdullah, M.M., Yahaya, A.S. and Ramli, N.A., 2015, January. Comparison of linear interpolation method and mean method to replace the missing values in environmental data set. In *Materials science forum* (Vol. 803, pp. 278-281). Trans Tech Publications Ltd.

Xu, G.N., Xin, H.J., Lu, F.Y. and Yang, M.L., 2012. Kinematics and dynamics simulation research for roller coaster multi-body system. *Advanced Materials Research*, *421*, pp.276-280.

Sustainability of Digital Formats: Planning for Library of Congress Collections. 2020. *Wavefront OBJ File Format*. Available from: https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml [Accessed 15 March 2024]

Bournemouth University. No data. *THE OBJ – ALIAS WAVEFRONT FILE FORMAT.* Available from: https://nccastaff.bournemouth.ac.uk/jmacey/OldWeb/RobTheBloke/www/source/obj.html [Accessed 15 March 2024]