Czech Technilcal University in Prague
Faculty of Electricel Engineering
Department of Computer Graphics and Interaction

Bachelor thesis

Roller Coaster Simulator

Tereza Hyková

Supervisor: Ing. Jaroslav Sloup

Study Programme: Software engineering and Management

Field of Study: Web and multimedia

May 27, 2010

Aknowledgements

I would like to thank Ing. Jaroslav Sloup for supervising this thesis, and my friends and family for their support during my studies.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague, May 27, 2010 .....

Abstract

This thesis describes the design and implementation of a roller coaster editor and simulator. It introduces the way to describe most of existing roller coasters including its shape, special mechanisms (e.g. lift or brake) and appearance. It implements an editor that provides all the tools required to design a roller coaster, which can be previewed in the simulator. The thesis then describes the physical quantities affecting the train's movement and implements a simplified physical model used in the simulation. The user can experiment with different types of cars and ride his own roller coaster with an on-ride camera.

Abstrakt

Práce se zabývá návrhem a implementací editoru a simulátoru horské dráhy. Řeší, jak popsat většinu existujících horských drah včetně jejich tvaru, speciálních mechanismů (např. brzda nebo výtah) a vzhledu. Popisuje implementaci editoru, který nabízí veškeré nástroje potřebné k navržení libovolné horské dráhy. Ta může být následně spuštěna v simulátoru. Práce dále popisuje fyzikální veličiny, které ovlivňují pohyb vlaku, a implementuje zjednodušený fyzikální model. Uživatel pak může s nastavením vlaku experimentovat a projet se po své dráze z pohledu prvního vagonu.

Contents

List of Figures

Chapter 1

Introduction

Roller coaster is one of the means for an active relaxation providing high level of adrenaline and excitement. Theme parks around the world keep competing for the world records in the height, speed or amount of inversions, trying to lure more and more customers who look for an excitement. However, there are still very few parks that own these high level roller coasters, because they are extremely expensive. Visiting such a theme park is quite difficultly accessible experience for many people not just because of the price, but also because of the distance – for example there is no big roller coaster in the Czech Republic at all.

With creating a virtual roller coaster ride we unfortunately cannot achieve the main thrill it provides – the real feeling of sitting in the train and riding the coaster's hills and loops with its huge speed. We can, however, try to simulate the visual experience and compensate the lack of movement with the ability to build any roller coaster according to the user's imagination including both its shape and appearance.

The goal of this thesis is to implement a roller coaster editor, where the user will be able to create any desired track. It will provide all the necessary tools for editing any roller coaster, support mechanisms like brakes or lifts, and allow the user to change the roller coaster style. Afterwards he will be able to start its simulation based on a simplified physical model. The user will be able to change the train's appearance, and important physical quantities that affect its movement. He will also be able to switch to the on-ride view.

In the following sections, we will make a short roller coasters overview (Section 2) and look at the existing roller coaster editors and simulators and their abilities (Section 3). Then in Section 4 we will analyze and find the solutions for the key problems we need to deal with first. Section 5 will describe the implementation of the parts specific to this application. We will look at the results of the final application in Section 6 and in Section 7 we conclude.

Chapter 2

Roller coasters overview

Roller coaster is an amusement ride developed for theme parks where it creates the biggest attraction and sometimes even a symbol of the park itself. The biggest parks try to cover as many roller coaster types as possible to provide the variety of shapes, colors, and levels of excitement. In this Section we will focus on these roller coaster types and the elements they usually contain.

2.1 Roller coaster types

There are many groups of types of roller coasters, which differ in their manufacturer, the track style, materials they are made of, height and speed, elements that are usually connected with it, and many other factors. Since it is impossible to just assign a fixed type name to a roller coaster, we will rather cover the most common groups that are to be found in Europe.

Classic sitting roller coaster This type of roller coaster is probably the most common and the most ordinary type there is. It forms a circuit and the train's energy is gathered while it is being lifted up on the first initial hill. These are usually steel coasters, but wooden ones also belong to this category. The elements they can contain depend on other factors – classical steel coasters can contain nearly any element there is, while for example inversions are very rare on wooden ones.

Shuttle roller coaster Shuttle coasters provide the car's energy by launching them from the station with a huge force, rather than lifting them first like the classic ones. These can be either circular (these are also called launched coasters) or not (accelerator coasters) – the train arrives at the station the same way it started after it loses energy during the ride.

Inverted and suspended roller coaster This is a type of roller coaster, where the trains hang on the rails instead of being attached to it from the top. Unlike suspended roller coasters, where the cars hang (and freely swing during the ride to the sides), inverted roller coaster's cars are tightly attached to the track. These types are not usually very high or fast, but contain many inversions and are very interesting shape-wise.

Bobsled roller coaster Bobsled coasters, as the name suggests, are derived from the track design used for bobsleigh sports. The trains are not attached to the track at all. The track has a half-pipe shape and the cars are very small (they usually carry up to two passengers). These roller coasters cannot contain any inversions or even any vertical drops, as the ride would become very unsafe.

2.2 Roller coaster elements

In this Section we will introduce the common elements to be found on roller coasters. Each of them is specified with the effect they have on passenger, usually in the direction and amount of g-force. Most of them are however just combination or modifications of the basic ones. We will mention just a few of them.

Loop We can see loop in Figure 2.1. It is a very popular element with positive G's all along the track (if the train has enough speed).

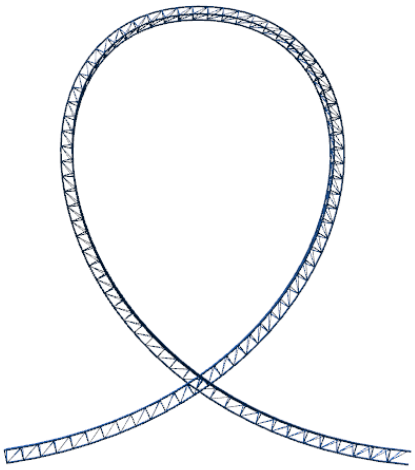


Figure 2.1: A drop shaped loop.

Twist Twist (sometimes also called roll) is an element where the track gradually rotates by 360°. A heartline twist shown in Figure 2.2 keeps the passenger's heart in the center of the rotation, while an inline twist does not.

Zero-gravity roll Zero-gravity roll (also known as zero-G roll) is an element very similar to twist, but with a hill (there is hardly any elevation at twist) in the middle where the train is upside down. At the top of the hill the g-force is equal to zero, hence the name zero-g roll. The element is shown in Figure 2.3.

Figure 2.2: A heartline twist with a clockwise rotation.

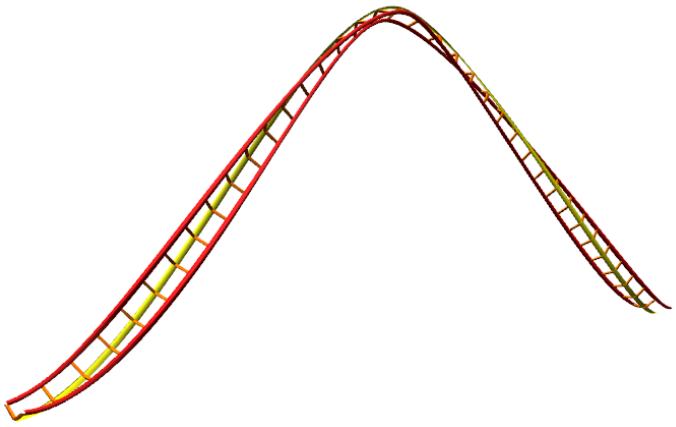


Figure 2.3: A zero-G roll.

Chapter 3

Existing editors and simulators

After the search for existing roller coaster editors and simulators we find two main approaches: the first being a professional simulator with precise tools of editing and the second is an integrated tool in a game with simplified controlling. These are represented by two simulators: No Limits and Roller Coaster Tycoon. There are other applications, but mostly outdated, no longer accessible, or they have very limited functionality. All of these are omitted from this overview.

In studying these applications we focus mainly on the following fields:

- the principles of defining the track – how much freedom the editors provide and how the track shape is defined,
- the drawing of the track, especially the methods used for reducing the number of triangles in lower levels of detail,
- the platform and the cost of the product.

3.1 No Limits

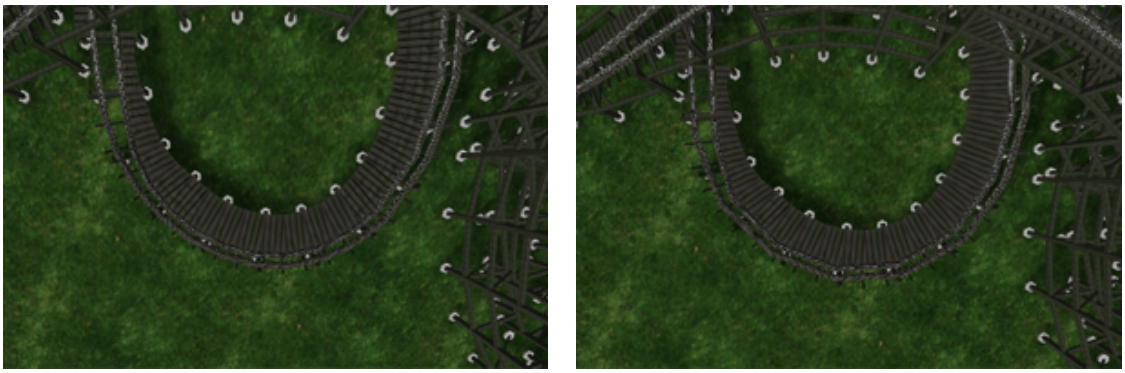
No limits is a very professional roller coaster simulator that has been developed in years 2000-2007 in OpenGL by a team of programmers. The full version costs $29.95, the trial demo version is for free but many features are disabled. For the research purpose we test only this demo version.

The whole roller coaster track is defined by an editable spline consisting of segments – in this case segment being a cubic Bézier curve. A roll is set in each point of this spline (i.e. the ending point of one segment and starting point of another) – the editor allows the user to use both relative and absolute roll and predefined steps or increments are available: 45°, 10°, 1° and 0°.

Each segment is assigned a type – track, station, transport, lift or brake, each having more custom settings. The editor allows saving and loading prefabricated pieces from custom file format. It doesn't check the coaster validation during editing at all but reports errors and warnings when user starts a simulation (which is disabled in demo version).

In the simulation the program displays a detailed roller coaster graphics. The physics is very realistic and all the physical quantities such as speed, acceleration and G's are displayed during the ride.

The program provides two or more levels of details (the distance is adjustable in program settings). A visible difference between these two levels is in drawing smaller amount of lines (referred to as segline) in one segment (Fig. 3.1), the distance between sleepers, and fewer details in track support geometry (Fig. 3.2).



(a) (b)

Figure 3.1: Differences between the distance of seglines in different LODs. b) shows one LOD lower than a).

(a) (b)

Figure 3.2: Differences between the distance between sleepers and support geometry in different LODs. b) shows the track from bigger distance than a).

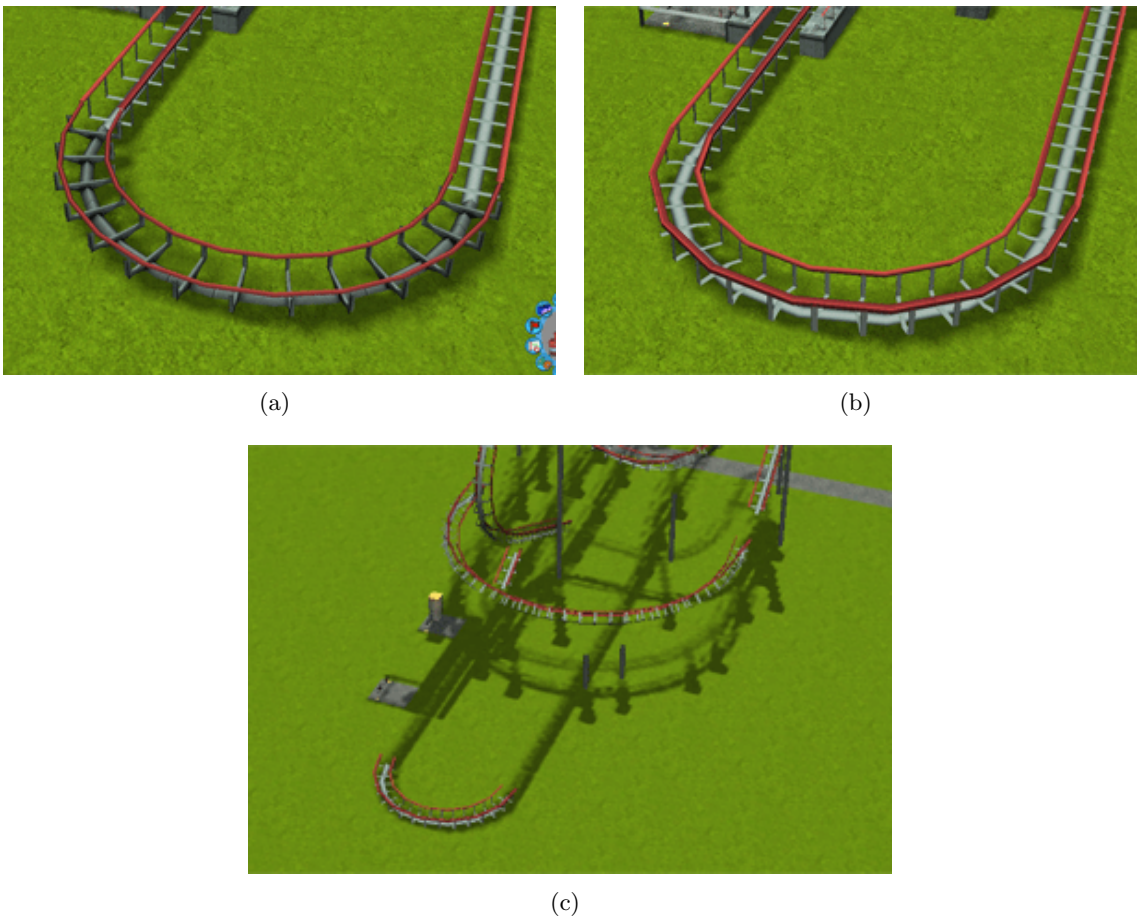
3.2 Roller Coaster Tycoon

Roller Coaster Tycoon (RCT) is a PC game from Atari Interactive, Inc – a building strategy where user builds his own theme park. RCT is a series of three games, but the principle of designing a rollercoaster is the same in all of them, therefore I will be describing

just the newest one (2004, this game is currently sold for $29.95 and a demo version is available), which is unlike the previous two in 3D and allows the user to actually ride the coaster from 1st person view. RCT is not oriented specially on building roller coasters but still contains a quite advanced roller coaster editor. Unlike No Limits, this program is not a professional tool but rather concentrates on enabling a design of a track as fast and as easily as possible, which results in big restrictions in the freedom of editing.

The editor consists purely of putting predefined track pieces together (especially in RCT 3 there is a very wide range of them). The track is built continuously piece by piece without the option to edit parts of the rollercoaster in the middle – the user has to delete all the track pieces from the end to get back to a desired place.

The drawing of the roller coaster is rather simple – the geometry constructing the track is very obvious even from close distance. There are at least three levels of detail (Fig. 3.3), distance dependent on the game settings of computer performance. With increasing distance the textures are replaced with a simple material, the segments are drawn with a smaller number of seglines and in the end the distant track pieces even start to disappear leaving only their shadow.



(a) (b)

(c)

Figure 3.3: The differences between various levels of detail in RCT 3. The distance of the camera from the track gradually increases from a) to c).

3.3 Roller Coaster Simulation - bachelor thesis by Jan Uher (2009)

This program created in Java and OpenGL is a freeware and contains a very simple and limited track editor and simulator. The editor is based on editing a 2D spline (3D vertices in XY plane with the Z coordinate fixed to 0) from Catmull-Rom curves. The 2D limitation prevents the roller coaster from having any turns at all, making it impossible to design any realistic track. It also lacks the option of rolling the spline along its tangent.

The physical model lacks rolling resistance and the air resistance is simply subtracted from the acceleration, which results in incorrect behavior in case the train should lose energy and stop.

The geometry is drawn based on the parameter t of the curve, even in the case of sleepers and support pillars. There is only one roller coaster style available. The application does not support LOD.

This program is a good start but unfortunately too far from what we will aspire to achieve because of its limited abilities and errors. We will omit it from the final conclusion.

3.4 Conclusion

No Limits and Roller Coaster tycoon offer us two different points of view on the way of designing a roller coaster. No Limits gives us much more freedom, but it is also more difficult to create a realistic track while in RCT it is rather easy but the possibilities are limited.

Both programs are for free only as demo versions, but offer high level roller coaster editing tools in both approaches. They also support levels of detail to lower the performace requirements.

Chapter 4

Analysis and design decisions

In this chapter we will analyze what is needed before we can start the actual implementation. In Section 4.1 we will talk about how to describe a roller coaster track and in Section 4.2 we will discuss the physics and physical quantities connected with the train's movement.

4.1 Roller coaster track description

Before we can design the format itself, we first need to analyze what is actually required to completely define a roller coaster track. In section 4.1.1 we focus on defining the overall shape of the track. Section 4.1.2 then describes how to determine the orientation of the track, which defines how a train is actually positioned in each point. Section 4.1.3 introduces individual properties of track segments, e.g. brakes and lifts.

4.1.1 Shape

To describe a shape of a roller coaster, we will look at the track as if it was just a curved line in 3D space and neglect all the other details which are not important for this step. We will describe the whole track by a spline – a set of segments, each consisting of starting and ending key point, which is shared by two adjacent segments (ending key point of one segment is the starting key point of the following one) as shown in Figure 4.1. But we need to find a way how to find the points between the key points. At this point, we need to meet several conditions so that the shape of a roller coaster remains valid:

- Continuity: roller coaster track shape must not have any sharp edges so that the ride is smooth and so that the train does not collide if the angle is too sharp.
- Intuitive behavior: we need to achieve as simple track shape editing as possible so that the roller coaster editor is user friendly. This means that we have to find a way to define the shape that a normal user can understand and work with, and assure that the result of the changes the user makes in already defined shape will correspond to what he would expect.

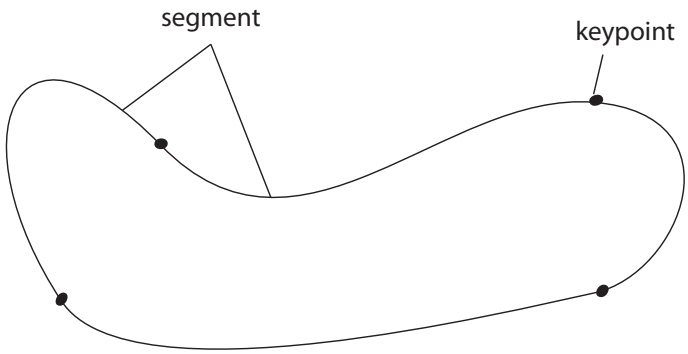


Figure 4.1: A spline consisting of segments and key points.

- Closed circuit: we want to be able to tell that the track is a closed circuit (as is usual for the most of roller coasters in the world) while the option to leave it open is still available (these roller coasters are called shuttle roller coasters – trains are usually launched from a station with large speed and return the same way)

There are several ways to describe a curved line:

We can use a suitable polynomial interpolating function to find any vertex between two adjacent key points. This option has many disadvantages: we either have to use linear interpolation – it is simple and intuitive but creates a sharp edge at any key point that is not in the middle of a straight line, therefore it does not meet our continuity requirement. Next option is to use polynomial interpolation of higher degree (linear interpolation is actually just a polynomial with degree of one) that takes into account more key points and defines a polynomial that all these points lie in. However, this approach is not unambiguous because we can always find more different polynomials that go through the same points (Fig. 4.2).

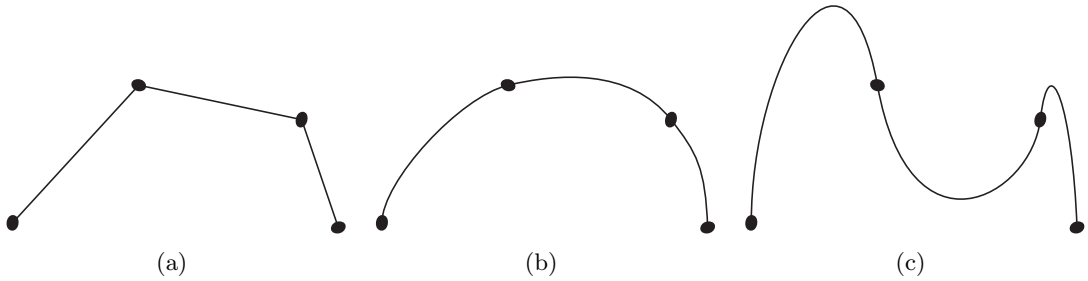


Figure 4.2: The same four points differently interpolated with polynomials. We can see the ambiguity of polynomial interpolation. a) shows linear interpolation, b) one way of a polynomial interpolation and c) another way of polynomial interpolation.

More suitable solution of our problem is a use of curves that have their own interpolating function. There are many kinds of curves to choose from, we will look at Bézier curve. While linear Bézier curve is nothing more than a straight line between two points, quadratic one operates with an extra point that creates the curvature and cubic one adds yet one more point to have more control over this curvature.

Cubic Bézier curve (Figure 4.3) is a parametric curve, which consists of two key points and two auxiliary points, and the interpolating function has the following definition:

$$B(t) = (t-1)^3 P_0 + 3(t-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3, t \in [0,1] \tag{4.1}$$

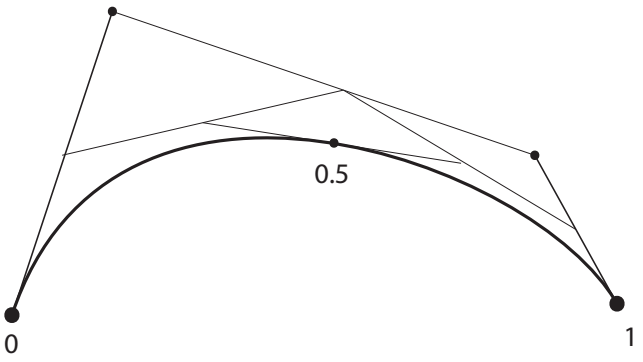


Figure 4.3: A Bézier curve with an example of finding the point at t=0 using de Casteljau algorithm. [4]

We access the points inside with parameter t belonging to interval $\langle 0, 1\rangle$. This parameter is not uniformly distributed along the curve, the distribution depends on the position of two auxiliary points. Further we will refer to a fixed step on the curve meaning a constant multiplier of this parameter.

Cubic Bézier curve is commonly used in many graphics editing programs – may they be bitmap, vector or 3D editors. Therefore, many computer users have already seen and used it before. Even though they do not need to know how the interpolation works, they are usually well aware of how these curves behave. For us, Bézier curve brings the advantage of explicitness, simple implementation and optional continuity – it offers up to C1 continuity depending on the position of two auxiliary points on two adjacent curves (Fig. 4.4). There is parametric (C) and geometric (G) continuity, we will not discuss the principals of each but rather mention the ones that are important for us. [4]

- C0 continuity – two auxiliary points belonging to one key point do not create a straight line. It means that the segments are connected to each other in a knot (the key point they are connected with).This results in sharp edges though, which we need to avoid.
- G1 continuity – two auxiliary points belonging to one key point create a straight line, but the distance between first auxiliary point and key point and between the other auxiliary point and key point is not the same. Two segments are connected in knot and the tangent in the ending point of first segment equals the tangent of the starting point of the next segment. Two curves connected with G1 continuity do not rapidly change the direction in knot.
- C1 continuity – two auxiliary points belonging to one key point create a straight line, and the distance between them and the knot is the same. It assures that the direction

does not change and neither does speed of a point moving on this spline with a constant step.

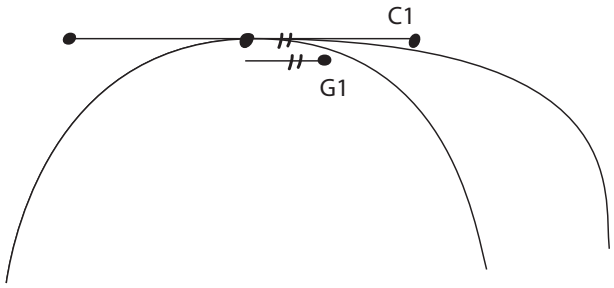


Figure 4.4: The difference between G1 and C1 continuity.

For a valid track shape we only need the G1 continuity (since we will not be able to use speed depending on parameter t anyway) but we will need to put a restriction on auxiliary points editing so that it stays preserved.

4.1.2 Orientation

With defined shape of the track, we need to define the local frame of reference in each point, which will determine the orientation of a train in that point. One axis of this frame of reference is obviously the tangent of the spline. Therefore we only have to define the frame's up vector, which is, on a typical roller coaster, a direction where a passenger's head is pointing away from the track. On the same straight line the passenger can sit normally, upside down or even rotate in any direction that is perpendicular to the track at that point.

Because we only have a curve in 3D space, we cannot easily determine the direction of this up vector – it can be anywhere in the plane given by the current point of spline and its tangent. Although there are algorithms that can determine the local frame of reference based on the curvature of the spline, for example Frenet-Serret Frame [2], which poses two major problems. First, it is not defined on straight segments, which are quite common to roller coasters. Second, and more important, it does not behave intuitively. In a turn its up vector points to the center of the turn. To see why this is undesired behavior, imagine a simple turn parallel to the ground.

This problem has two phases. The first one is determining where the up vector points solely from the track shape. We will demonstrate this problem on a loop (Fig. 4.5(a)): the starting up vector points straight up and as the track goes upwards, it gradually rotates the up vector and in the end we find ourselves completely upside down at the top peak of the loop. Then in the second half the up vector returns to the original position. All this happens just because of the shape.

While this may seem very obvious, let's see another picture: exactly the same track shape, but on this one the up vector rotates in the first half which results in up vector pointing straight up (Fig. 4.5(b)). While this may be in some cases desired, we want to have full control over such behavior.

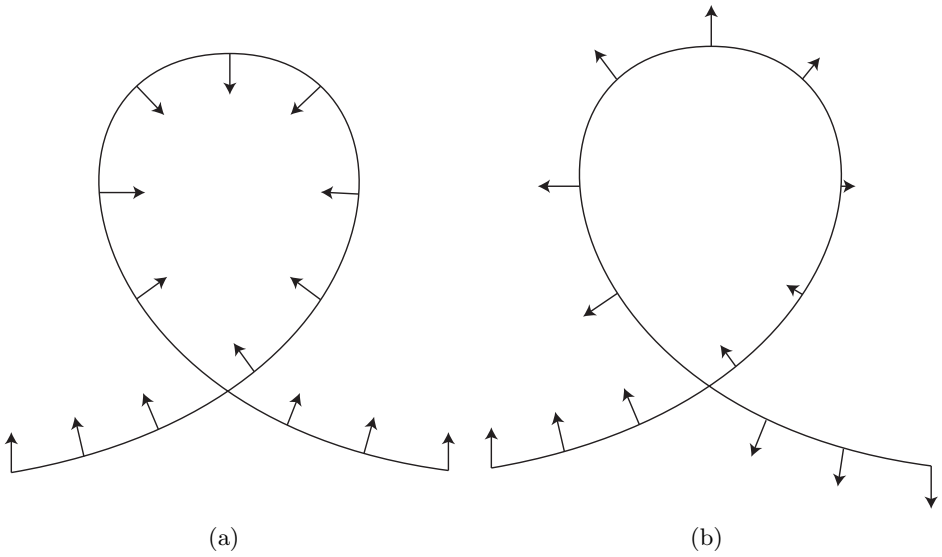


Figure 4.5: A loop shaped track with two different orientations.

First solution would seem to lie in defining a rotation of the original frame of reference (being classical coordinate system) in each key point. We can define that by three angles referred to as yaw, pitch and roll (Fig. 4.6) which however causes big problems with interpolation or gimbal locks. [1]

Next option is to define rotation with quaternions. They would remove the problems of yaw, pitch, roll solution, but still are not suitable for our application since the knowledge of frame of reference in key points give us no information about what the segment looks like in between them and rotation interpolation would not give us correct results (for example loop created from just one segment starting with the same frame of reference as it ends with – interpolation would see it as a straight segment).

Since we cannot just get the up vector from the information of any point inside of the curve, we need to use more information than given in just one point - we will iterate through the curve. From given up vector at the start of the curve (straight up in our case) and the shape of the curve we can calculate a new up vector using the previous one.

In each point we use the last up vector. Using a cross product of the last up vector and tangent in the current point we calculate a binormal – a vector that is perpendicular to both tangent and last up vector. Another cross product of tangent and this binormal will give us our new up vector. Now the tangent, binormal and new tangent give us a new reference frame – a coordinate system with the center being our specified vertex on spline (Fig. 4.7).

This approach has the obvious disadvantage of having to iterate to each point from the complete start of the spline with explicitly defined up vector. If do not iterate with a step small enough, we might skip some rapid changes in the spline shape and end up having different up vector than we want. To make things faster, we will remember the last up vector at each key point and we will prepare several points in each segment so that we lower the time to get to our desired point.

At this point we still cannot describe all shapes that we want – for example a twist – a

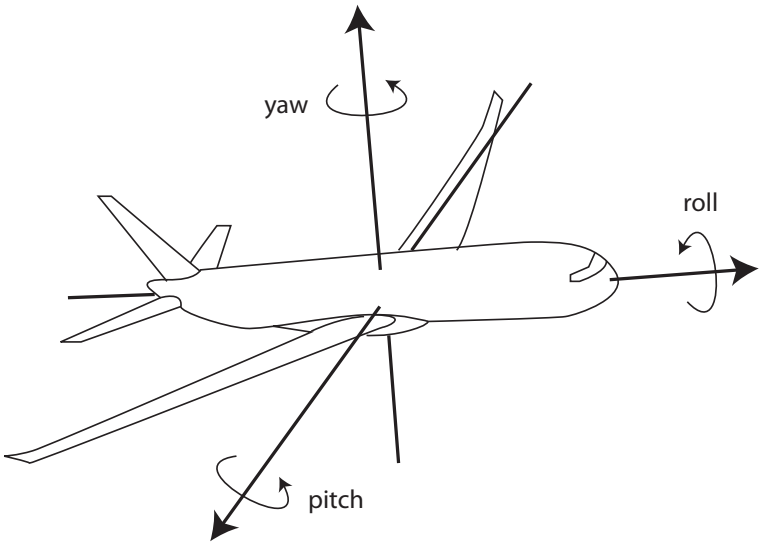


Figure 4.6: Yaw, pitch and roll demonstrated on axes of an airplane.

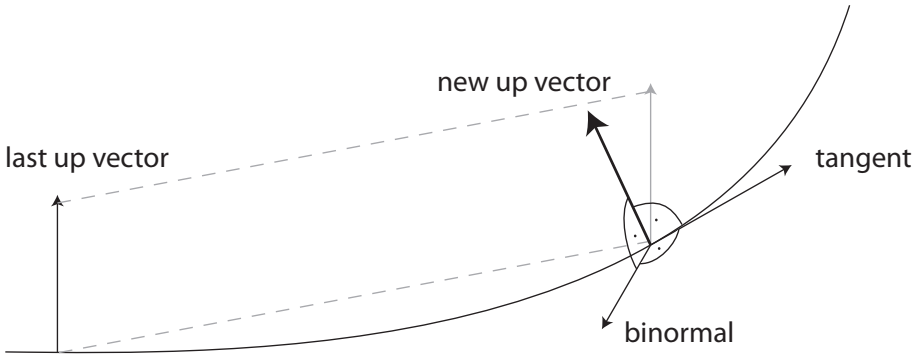


Figure 4.7: Computing the local frame of reference using last up vector.

common roller coaster element where the track is nearly straight line, but the train gradually rotates by 360°. This is where a user must explicitly specify a roll – an angle that tells us how the up vector should rotate along its tangent (Figure 4.8). We will define roll at each segment, positive roll meaning rotation clockwise and negative meaning rotation counter-clockwise, so that we can define rotation in both directions. This number is very easily interpolated – a simple linear interpolation of 0° (roll in starting point) and user defined angle in ending point is what we need.

After rotating the up vector we need to recalculate the binormal in order to have correct new frame of reference (Figure 4.9) – an another cross product of tangent and the new up vector will return the desired binormal. We also have to make sure all the vectors of the frame (up vector, binormal and tangent) are normalized (their length is equal to 1) .

Note that we always need to rotate the up vector the previous segment ends with, so we always need to pass the up vector in the end point to the next segment. We can either save

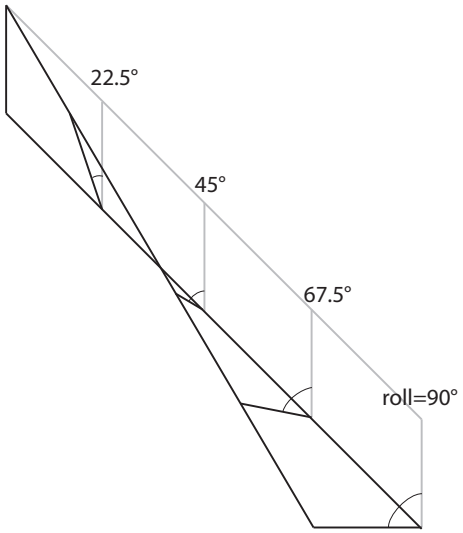


Figure 4.8: Roll and its interpolation applied on a segment.

the last rolled vector in each point and start with roll 0° in each segment or we can save the last up vector yet before roll and pass just the value of roll that adds up in each segment giving us the absolute roll value instead of relative one. Neither of these approaches has any advantages or disadvantages, we choose the second one.

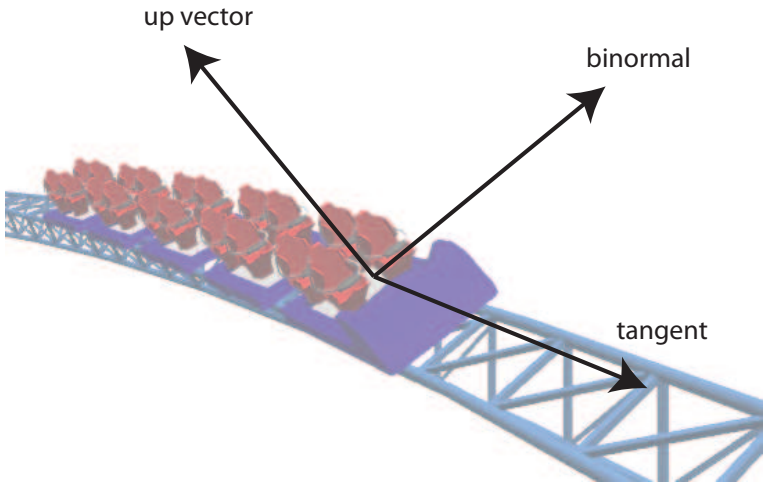


Figure 4.9: The resulting frame of reference shown on a real track.

4.1.3 Behavior

We can find many different mechanisms hidden in a roller coaster track, none of them less important than the other. Every roller coaster has a station for trains to stop and passengers

to get in and out of the train. Many roller coasters use lift to get the train on top of the first hill and gather enough energy for the whole ride. Shuttle coasters have very powerful boosts to increase train's speed on a horizontal track instead of pulling it up hill first by previously mentioned lift.

We will define this on each segment separately in our simulator. Each segment will have an assigned behavior type saying how it will affect the train's speed. These types are a normal track, a lift, a brake, a normal station, a shuttle station and a boost. The features of these types combine – for example a normal station acts like a brake when train is entering and as a lift after it reaches minimum speed. Shuttle station even combines brake, lift, and a boost. Each of these types will have the parameters to set such as minimum speed (lift), maximum speed (brake) or a force (boost). We will talk about this topic later when we discuss how this influences the actual movement of the train.

4.2 Physics

In this Section we will talk about what influences the movement of the train. We will discuss the important physical quantities in this area: gravity (Section 4.2.1), friction (Section 4.2.2), acceleratаion (Section 4.2.3), speed (Section 4.2.4), and g-force (Section 4.2.5). [5]

Although we will mostly use quantities that are normally vector quantities, we can limit them to scalar ones in our calculations. The roller coaster trains are very tightly attached to its track from all sides (there are exceptions such as a bobsled roller coaster where trains freely move in a pipe-shaped track, but our simulator does not cover these kinds of coasters) and therefore it is the shape of the track which defines the direction. We will only be interested in the value of these quantities, where positive value means it affects the train to move forward (in the direction of tangent) and negative for the backwards direction.

4.2.1 Gravitation force

We will calculate the gravitation force acting on an object on an inclined plane, where the inclined plane is the roller coaster track, as shown on Figure 4.10.

The force F is dependent on the object's mass m, gravity g and the slope of the hill defined by the tangent of the track at the specified point. The slope is characterized by the angle α between the vector pointing down and the tangent. The final force is then:

$$F = F_g \cdot cos\alpha \tag{4.2}$$

This means that if the tangent is parallel to the ground, the gravitation does not affect the object at all while if the tangent is perpendicular to the Earth's surface, it affects the object with its full impact.

4.2.2 Frictional force

Friction is the force acting against the movement of the object. This is the force that causes the roller coaster train to gradually lose its energy during the ride and eventually

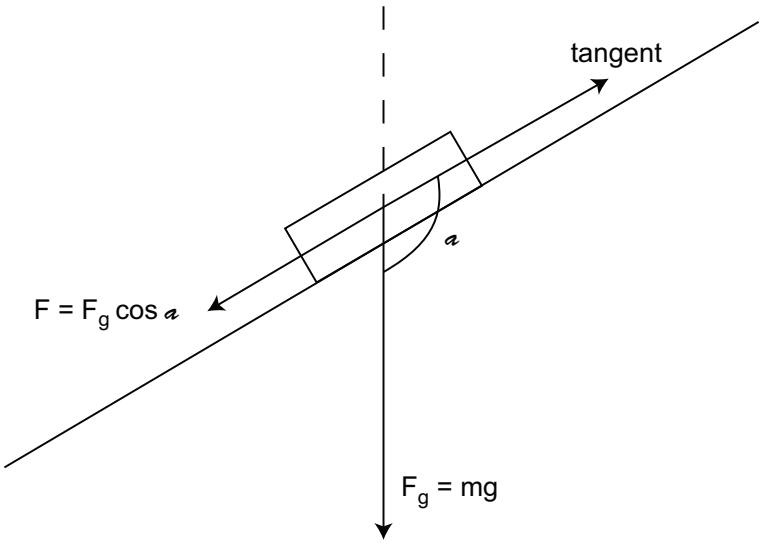


Figure 4.10: Gravitation force acting on an object on an inclined plane.

stops it completely. Although the friction can be significantly minimized, we can never avoid it completely. We will include the air resistance and the rolling resistance.

Rolling resistance

Rolling resistance is a special type of friction affecting a rolling wheel along a surface. This resistance is much smaller than normal friction (which is the reason for inventing the wheel). It can be calculated as

$$F = F_n \cdot \frac{b}{r} \tag{4.3}$$

where F is the rolling resistance, F_n is the normal force, b is the rolling resistance coefficient and r is the wheel radius.

The rolling resistance coefficient is a dimensionless scalar value, which describes how the two objects (the surface and the wheel) affect the resistance. It mainly depends on the materials the two objects are made. For example a steel wheel on an ice plane has a very small coefficient while a rubber wheel on a rough asphalt terrain has a high one.

The wheel radius value tells us that larger wheel will be affected by the friction less than a small one.

Normal force is the force perpendicular to the surface (plane defined by tangent and binormal in our case) and is the force that prevents the object from falling through. The scheme can be seen on Figure 4.11.

Normal force F_n is computed the same way as gravitation force in Section 4.2.1, but this time the angle α is the angle between the up vector and the vector pointing straight down.

$$F_n = mg \cdot cos\alpha \tag{4.4}$$

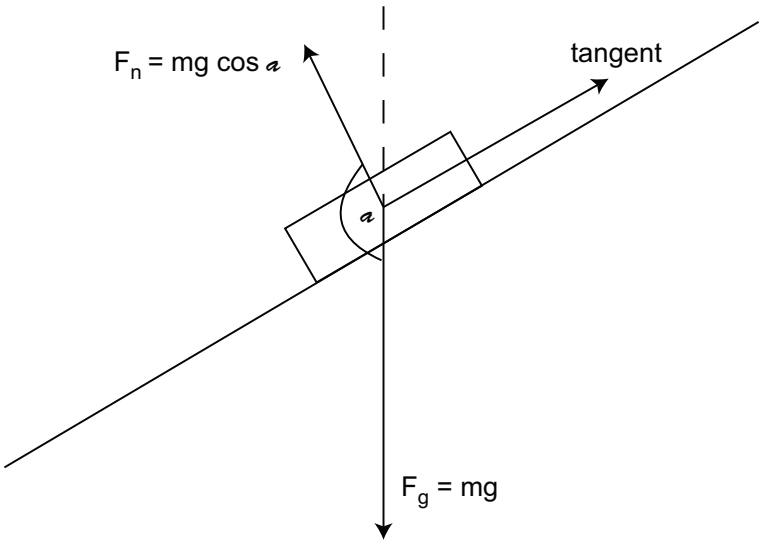


Figure 4.11: Normal force acting on an object on an inclined plane.

The fact that we need to take into account is that the roller coaster cars do not only have wheels on top of the rails. In order to attach the car tightly to the rails in all the loops and banked turns, they also have wheels from the sides and from the bottom. Therefore we need to add another friction force that affects the binormal direction – the same formula with the angle α being the angle between binormal the vector pointing straight down. Without this friction a track rolled by 90° to either side from a normal state (up vector pointing straight up) would have no force friction affecting it at all.

Because there are always side wheels from two sides of the rails and they point in the opposite direction, we only need to calculate the force ones. The same goes for the wheels touching the rail from the top and from the bottom. The resulting rolling resistance will then be:

$$F = F_u \cdot w + F_s \cdot (\frac{w}{2}) \tag{4.5}$$

where F is the rolling resistance, F_u resistance acting on the wheels on top/bottom, F_s resistance acting on the wheels from sides and w the number of sets (set of wheels being the three wheels from top, side and bottom) per a roller coaster car.

Air resistance

Air resistance is the force caused by friction of an object and the environment - in our case the air. The calculations are quite difficult and its size depends mainly on the environment. It is significant mainly in thick fluids, and the effect of the friction of air and our train is quite negligible. We will therefore not include air resistance in the calculations.

4.2.3 Acceleration

Acceleration is a vector quantity specifying the change of velocity over time. The acceleration a of an object positioned on the track is calculated as the sum of all the forces F acting on it divided by its mass m:

$$a = \frac{F}{m} \tag{4.6}$$

In our case these forces are gravitation force described in Section 4.2.1 and friction force calculated in Section 4.2.2. Our trains can have multiple cars, each is affected by the forces on its own but the result affects the whole train. For example, the first car of the train which would weight a hundred times as much the others would affect the train's movement much more than any of the other lighter cars. We then need to add all the forces acting on each of the car first and then divide it by the total weight of the whole train.

4.2.4 Speed

The train speed can be computed from its current acceleration. We will use Euler method [3] which is the simplest numeric procedure for solving ordinary differential equations (there are higher-order methods such as Runge-Kutta methods for more accurate results). Although it is accurate only for constant acceleration, it is still sufficient, even if our acceleration changes. After putting our quantities in the formula, we get:

$$v(t_0 + \Delta t) = v(t_0) + \Delta t \cdot a(t_0) \tag{4.7}$$

where $v(t_0)$ is the speed of the train in time t_0, Δt is the time elapsed and $a(t_0)$ is the acceleration at time t_0. The new train position can be then calculated as:

$$p(t_0 + \Delta t) = p(t_0) + \Delta t \cdot v(t_0) \tag{4.8}$$

where $p(t_0)$ is the position of the train in time t_0, Δt is the time elapsed and $v(t_0)$ is the speed of the train at time t_0.

4.2.5 G-force

G-force is a measure of acceleration. It is the quantity that is responsible for the thrilling factor of a roller coaster. It is measured in Gs, where G is the size of Earth's gravity. While g-force can have any direction, in a roller coaster case we are usually interested in the vertical direction that either pushes the passengers into their seats (positive Gs) or out of them (negative Gs). For example, the g-force of 1 G affects a stationary object resting on Earth's surface, because the normal force of the surface pushes it from the bottom with the same acceleration that gravity pushes it down, which is 1 G.

When computing the direction (visualization on Figure 4.12), we have to, unlike with the movement on the spline, manipulate with vector quantities. The G-force can be calculated from two vectors of velocity in time t_0 and $t_0 + t$. From these two and the time t that elapsed between them we can calculate the acceleration as

$$\vec{a} = \frac{\vec{v}(t_0) - \vec{v}(t_0 + \Delta t}{\Delta t} \tag{4.9}$$

By adding the gravity $\vec{g}$ (with vector pointing down) we get the vector $\vec{a}'$. Then to get the value of the g-force G_F in vertical direction we compute a dot product of $\vec{a}'$ and negated up vector at the current point of the spline. To convert it to G units, we divide it by the value of gravity g (9.81 m/s^2 on the Earth):

$$F_G = \frac{(\vec{a} + \vec{g}) \cdot (-\vec{up})}{g} \tag{4.10}$$

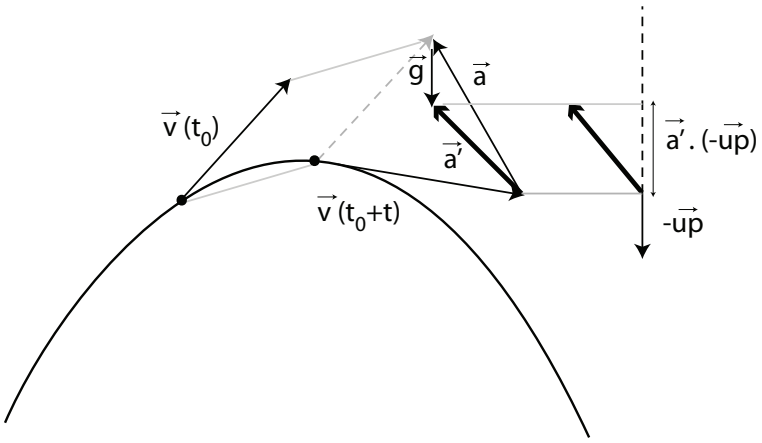


Figure 4.12: The computation of g-force on top of a hill.

4.3 Editor

One part of the application is a roller coaster editor. It is practically the only part where the user interface requires a special attention. It will be discussed in Section 5.7.1. It is however based on the analysis of the functionality requirements:

- Editing roller coaster's shape by editing the segments. This includes adding, removing segments and finishing and reopening the circuit.
- Attaching other coaster files, enabling to add prefabricated track pieces.
- Adjusting the segments' options – the roll, behavior types and their settings.
- Setting the spline information such as the style or train position.
- Adding, removing and positioning the support pillars.
- Saving and loading saved roller coaster.
- The availability of preview and shortcut to displaying the roller coaster in simulator.

Chapter 5

Implementation

5.1 Implementation platform

Our application is using Microsoft XNA Framework 3.1 [1]. XNA is a framework based on DirectX (9.0c in the case of XNA 3.1) and is created for game developers. Its libraries offer wide variety of useful functions in many areas; we especially use the game loop and tools for manipulating with vectors, matrices and other structures necessary in computer graphics.

The programming language is C#, which has been chosen mainly by the determined XNA framework. It is an object oriented language with syntax based on C++, stripped from the low-level programming features such as working with pointers (although the option is still available in unsafe mode) and the necessity to manually allocate or deallocate the memory (it has a garbage collector). The whole project has been created in the Microsoft Visual Studio 2008 IDE.

For implementing the user interface (UI) we use Windows Forms library [2]. It offers all the basic UI elements we need in our application and poses the least problems with integration into XNA framework (the other options is for example WPF [3]. The only problem is the problematic keyboard input. We avoid it by using UI elements that do not require keyboard input.

5.2 Application architecture

Figure 5.1 shows a simplified class diagram of application's classes. Only the main classes and the most important associations are displayed.

The application uses game state management [10], which consists of a screen manager and game screens. They take care of separate parts of the program (simulator and editor in our case), and the transitions between them.

Class Spline represents the roller coaster. The principals and methods of describing the roller coaster track are described in Section 4.1. SplineLoader loads and saves the created

[1] Microsoft XNA Framework 3.1 can be downloaded at http://www.xna.com/.

[2] Official documentation available on http://msdn.microsoft.com/en-us/library/dd30h2yb%28v=VS.80%29.aspx.

[3] Windows Presentation Foundation, a graphical system for rendering UI in Windows-based applications

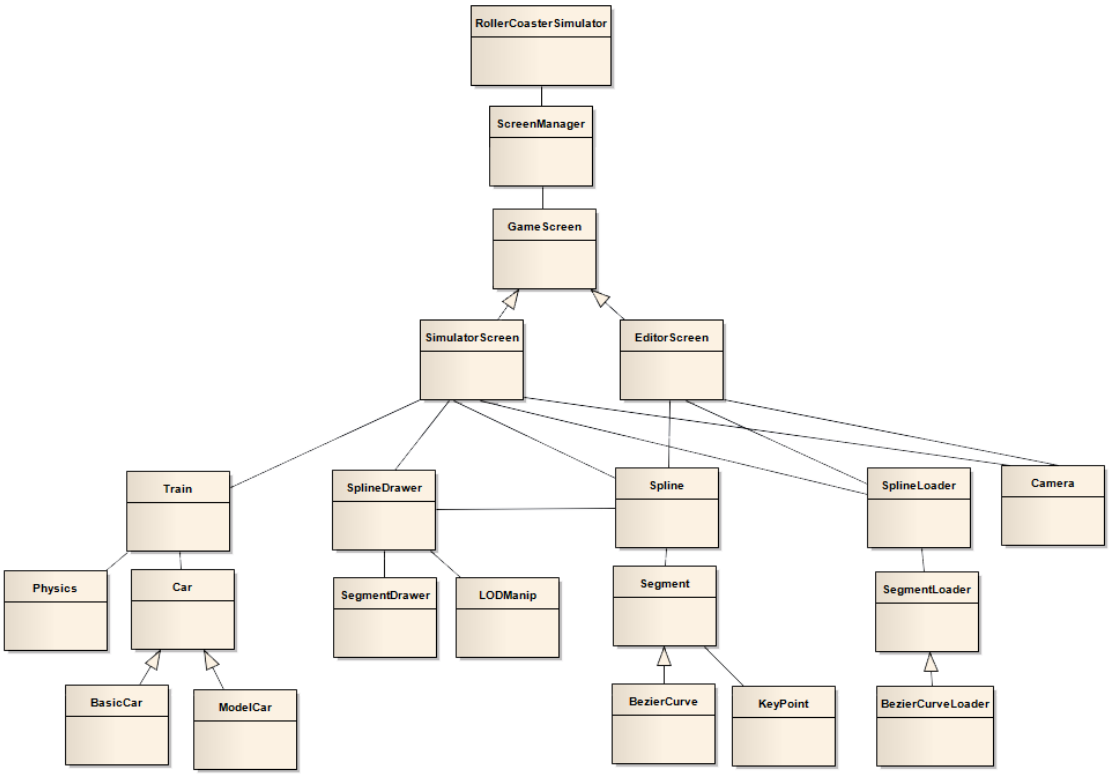


Figure 5.1: A simplified class diagram of the applications. Simple line represents composition; a lite with an arrow represents inheritance.

roller coasters to hard disk and its implementation is covered in Section 5.3. Class Train and Physics represent the roller coaster's train and its movement (Section 5.6). Spline drawer is responsible for drawing the geometry of spline in different ways including LOD (Section 5.5).

5.3 Format

In this Section we will discuss how to save the roller coaster data on hard disk and load them back into our application. The information necessary to describe a roller coaster are listed in Section 4.1.

The roller coaster is saved in a text file. Many applications (especially games) rather use binary file so that a common user cannot easily edit their content. This is usually desired in cases like game character abilities or game state advance and their direct editing would be considered as cheating. However, this is not our case since there is nothing to gain by bypassing the application. The first lines of our file contain the roller coaster information: the name, the information whether it is closed, initial train position and the name of style to be drawn. We also include the starting up vector here since we define that just once. The following example shows what the first lines of the text file might look like:

```
<roller coaster name>
<'closed' | ''>
upVector <up vector coordinates>
initialTrainPosition <initial train position>
rollerCoasterStyle <style name>
```

Then the list of segments follows. Since the segments do not necessarily need to be Bézier curve, we need to specify its type. But the application currently supports only Bézier curves, so we will concentrate only on their implementation. They are defined by starting point, ending point and two auxiliary points. The last number is the roll of the segment in degrees. These values are separated by a space.

```
seg <type>
<start> <first auxiliary point> <end> <second auxiliary point> <roll>
```

The third row of a segment definition is segment implementation independent. It gives information about its geometry to draw and behavior type. The setting of behavior is written right after and is dependent on the type selected – the values not used at the specific type are skipped:

```
<draw> <behavior> <friction coef.> <max speed> <min speed> <force>
```

The final line defining segment contains the information about the support pillar position:

```
support <'none' | support location>
```

5.4 Camera

In this section we will discuss the implementation of our camera. We will not cover all the basics of implementing the first person camera [9], but rather concentrate on the special features or problems.

Our camera is defined by 3D coordinates representing position and a rotation defining which direction it looks. For the rotations we use the yaw-pitch-roll approach, which is described in Section 4.1 and marked as unsuitable because of the problems with interpolation and gimbal lock. We will look at these problems and explain why we do not mind them in implementing camera while they were an obstacle in defining the track's orientation.

First, we do not interpolate anything. For the camera movement we only change the values of the angles and use them to calculate camera settings directly. Second, we avoid gimbal lock by limiting the absolute value of pitch to 80° or using our prepared local frame of reference at the specified position, where the perpendicularity of the tree directions is already ensured.

5.4.1 Camera modes

We want our application to offer several possible kinds of views of the world. They are all first person cameras, but differ their control, features, limitations, and hence also the implementation.

Free mode

Free mode is the state of camera, when it is almost freely controllable by the user and creates the impression of flying around the world. The direction of looking is controlled by moving the mouse, and keys W, A, S, D, Q and E change the position in different ways depending on the direction the user is looking at that moment.

This camera mode has two limitations (Figure 5.3). As mentioned above, the pitch angle is limited to the interval <-80°,80°>. Except for the method to avoid the gimbal lock problem, this is quite usual feature in many computer games or other applications with first person camera. It is very unnatural and confusing for the user to be able to rotate the camera all the way up and beyond the pitch of 90° resulting in being upside down.

The other limitation is the surface level. The limit is defined by the surface level plus avatar's height. This results in better control of walking, since the negative Y coordinate change of camera position is no longer applied once it reaches its minimum.

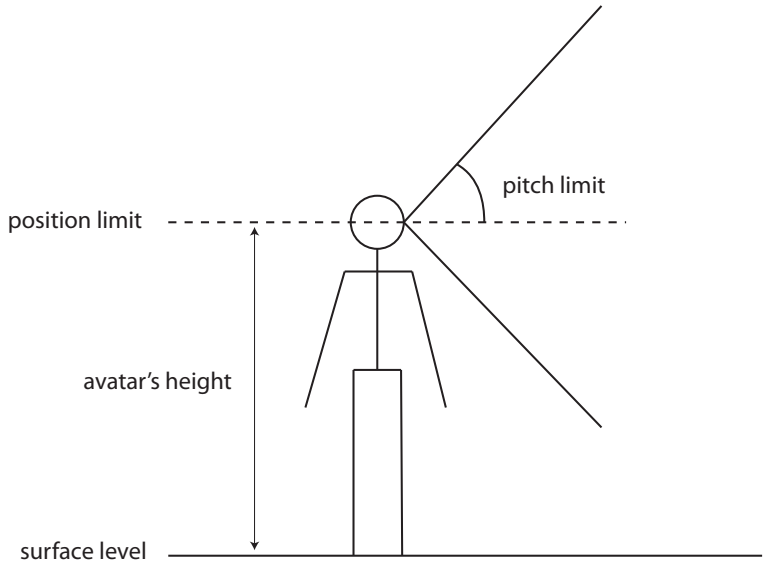


Figure 5.2: The limititations of free camera mode.

Free mode camera uses perspective projection, which means that the objects that are further away from the camera seem to be smaller. This projection emulates how a human sees the world with his eyes, hence is quite obvious choice for this camera mode.

On-ride mode

This mode represents the view of the passenger sitting in the train's first car. It is defined by the local frame of reference in the car's position instead of yaw and pitch angles, where the camera's position is offset in the up vector direction. This causes that the camera is not right on the track or inside the car's geometry, but is positioned rather where the passenger's head would approximately be.

All the camera controls are locked in this mode. The position changes with train position. The point the camera looks at is not simply defined by the tangent of the frame of reference, because the passenger's view and hence the ride experience would became dull at some places. For example on the top of hills or inside the loops, he would either see just a sky or the track's geometry instead what lies ahead of him on the ride. For this purpose, we calculate this point as the point of the spline that is 5 meters ahead from the position the camera is currently at (Figure 5.3).

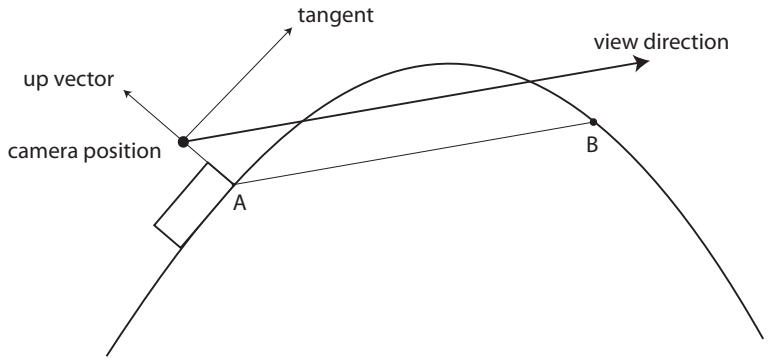


Figure 5.3: Calculating the view direction as the vector between car's position (A) and the point of the spline (B), where |AB|=5 meters.

Orthographic modes

Orthographic projection is the other kind of projection next to the perspective one (Figure 5.4). As the names suggest, perspective does not apply here. All the objects have the dimensions according to their size no matter how far away they are from the camera. This is used for technical purposes where we need to see the dimensions undistorted. In our case, they are used in the editor in top, front and side camera modes. We do not allow turning the camera in these modes.

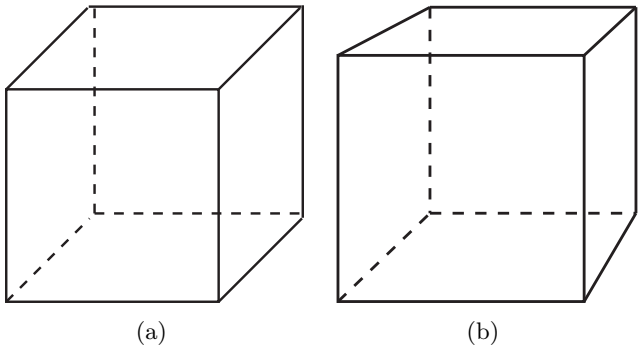


Figure 5.4: The same cube displayed with two different projections. a) ortographic projection, b) perspective projection.

Since the orthographic projection is set by the width and height of the view frustum, which is a box, the area we see is the same no matter how far away they are. Therefore we need to implement a zoom, which is simply changing the coordinate in which direction we are looking at the world (in the Y axis direction in the top mode, in the Z axis direction in the front mode and in the X axis direction in the side mode).

The change of zoom is based on changing the width and height of the view frustum depending on the position's coordinate corresponding to the current camera mode. If we are decreasing this coordinate, we zoom in by decreasing the width of the frustum, while increasing it does exactly the opposite. In these camera modes we also multiply the step length by a fitting constant, because the speed of zooming in and out would be too small.

5.5 Drawing

In this section we will discuss ways how to describe a roller coaster style, divide its parts into definable elements, have a look at how to render them in our simulator and try to reduce the computing time used for drawing using levels of detail. In the last section we will see how to place our roller coaster in the virtual world.

5.5.1 Roller coaster style

Some roller coaster designs are quite common and used all over the world, some less and some are modified to match the specific coaster's theme (for example see Figure 5.5). We want to be able to define them so that our roller coasters look like they do in reality.

Figure 5.5: Saw - the ride, a horror themed roller coaster in Thorpe park, UK.

We can divide a typical roller coaster style into three main parts: rails, sleepers and supports (Figure 5.6). Each of them follows different rules and needs its own special algorithm

to be placed and drawn correctly. When we define these, we can describe most of the common roller coaster designs. We will call the set describing rail, sleeper and support looks a roller coaster style.

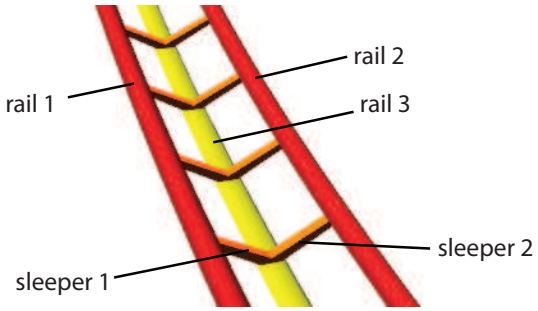


Figure 5.6: A part of track created from 3 kinds of rails and 2 kinds of sleepers.

The requirements for every style are given by abstract class CoasterStyle. Every specific style must inherit from this class and therefore define all the information needed for drawing. We will look at each of these requirements further.

5.5.2 Rail

The first element of a roller coaster track we are describing is a profile that keeps repeating itself along the whole spline. We will refer to this as a rail. The profile is defined by a set of 2D points that describe its shape and a set of normals for the lighting model.

If we want flat faces and perceived edges instead of smooth ones, we need to define each vertex twice with two different normals so that our normal vectors do not automatically interpolate where we do not want them to (Figure 5.7).

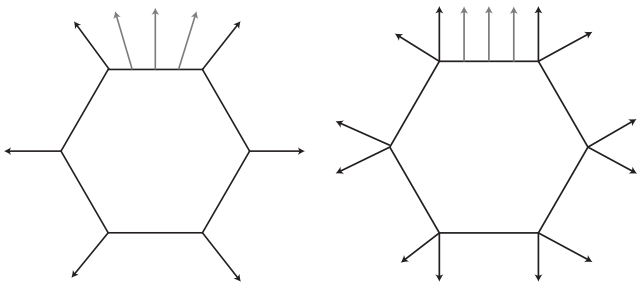


Figure 5.7: The same rail from 6-edged profile, but differently defined. a) rail defined by 6 vertices and 6 normals results in smooth edges, b) 12 vertices, 12 normals, flat faces

To avoid the necessity of explicitly defining which points belong to which rail, we will pass each shape separately (as shown in Figure 5.6 with three separate rails, although they could be described by just one profile). As we can see on Figure 5.8, the rails' shape can be described by two circles and a rectangle for the big supporting rail. To create the rail,

we extrude this profile – we copy it along the spline in predefined distances and each two consecutive profile copies define a set of quads, thus creating the track shape (Figure 5.9).

Figure 5.8: A detail of track profile of Nemesis - an inverted roller coaster in Thorpe park, UK.

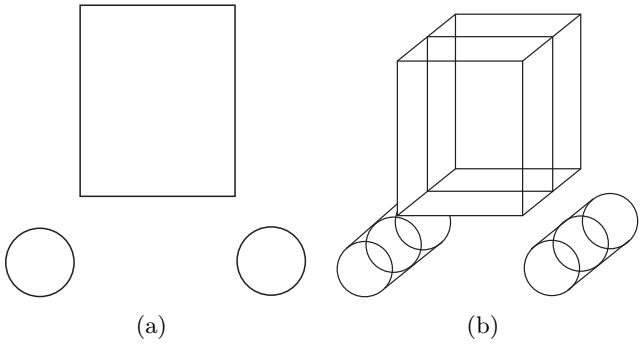
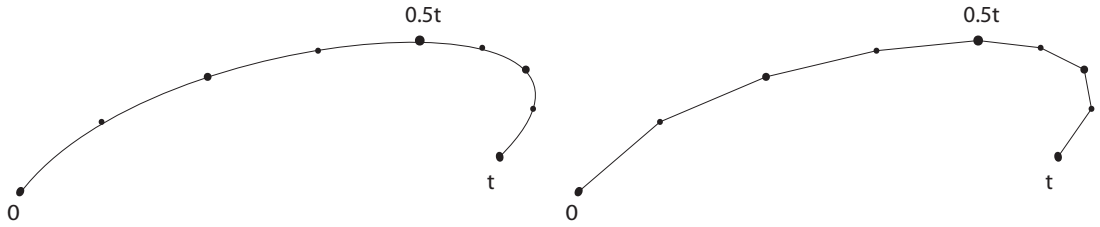


Figure 5.9: The rail profile of Nemesis - a) the basic 2D profile, b) the extruded profile creating a 3D geometry.

Since we can never draw a perfectly smooth rail – the basic geometry we can draw is a triangle which is flat – we have to divide a segment into smaller parts – seglines. These seglines are straight, but if they are short enough, it is hardly noticeable. However, the more seglines there are in one segment, the higher amount of vertices and more computer performance is required, so we need to find a suitable balance between good looking rails and computing demand.

At this point we will use the feature of Bézier curves, where the points given by constant step of parameter t are not equally distant from each other, to our advantage. As we can notice on Figure 5.10, the point density tends to be higher in the part where the curvature is steeper, which is exactly what we need – to draw shorter seglines where the turns are sharper, while straighter parts of segment can be drawn with just a few. This means that we will use just a constant step of t to determine where the seglines will be.



(a) (b)

Figure 5.10: The same Bézier curve with highlighted points at each $1/8t$. a) curve is drawn with many seglines, which makes it look smooth, b) curve is drawn with just 8 seglines.

5.5.3 Sleeper

The second element is a sleeper, which also repeats itself along the spline, but this time there is no extrusion. A sleeper cannot be defined as easily as rails because their geometry can be very variable (as shown in Figure 5.11). Therefore, we have to define the whole sleeper geometry as a set of 3D vertices, their normal vectors and indices that determine where the triangles are drawn.

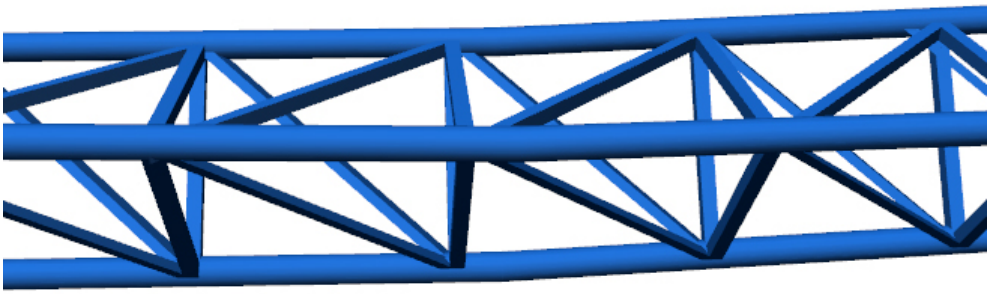


Figure 5.11: This coaster style consists of 6 different types of sleepers: three that are perpendicular to track's direction, and other three that create diagonals of the three rectangles in between.

Another difference between a sleeper and a rail is that we cannot use the constant step of t anymore, but we have to repeat the geometry in a fixed Euclidean distance along the spline instead. For details on obtaining this distance see Section 5.6.3. The distance between sleepers will be explicitly defined by the user, which can be used to define more complicated sleeper patterns as shown in Figure 5.12.

Also, sleeper position needs to be determined globally on the whole spline instead of separate segments because the sleeper geometry and the gaps between them can overlap into the following segments (Figure 5.13).

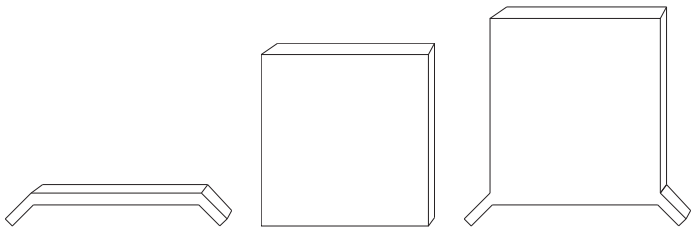


Figure 5.12: The sleepers defined to create the inverted style of Nemesis. a) shows a basic sleeper that repeats every 1 meter, b) shows another that repeats every 5 meters, c) shows the result of the combination of previous two on every 5 meters of track.

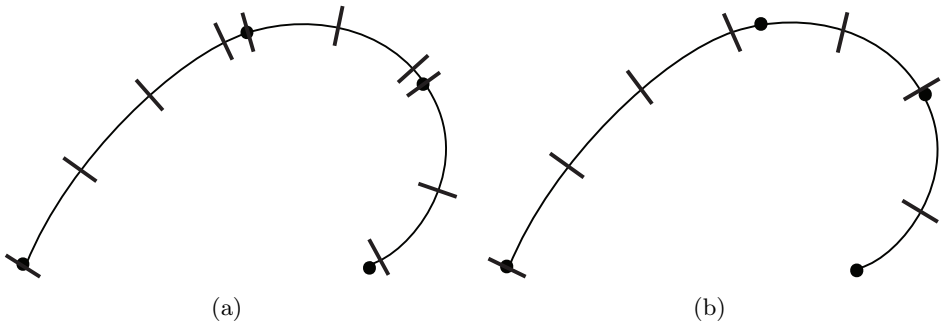


Figure 5.13: The same spline made of 4 segments (dots represent the key points). a) shows the result of sleepers defined per segment (notice the irregular gaps), b) shows sleepers defined per spline.

5.5.4 Supports

Each segment can have one support pillar where its position is defined in parameter t. The pillar's position will be determined by the track designer in our roller coaster editor.

We, for the sake of simplicity, limit supports to a cylinder. A user needs to define the pillars' radius, and the height that should be added to each pillar. This last information is included because the profile of the track or the sleepers can reach below or above zero Y-coordinate and we want the pillar to fit with the rest of track geometry (Figure 5.14).

Because the pillars can be placed at any position of the segment, we need to deal with the parts where the track is not horizontal – the pillar geometry can undesirably intersect the track's geometry as seen on Figure 5.15.

To tilt the top base of the pillar, we need to know the height of each vertex creating the top base of the cylinder. We first calculate the angle α between tangent in the point where it should be drawn and its projection into plane XZ – vector $\vec{B}$. Then using dot product of $\vec{A}$ and $\vec{B}$, we get the distance of the actual vertex of the base from its center in a tangent direction. At this point we have a right-angled triangle, we know one angle and the length of one cathetus and we can easily calculate the desired height Y of our vertex by using the tangent function. The draft can be seen on Figure 5.16.

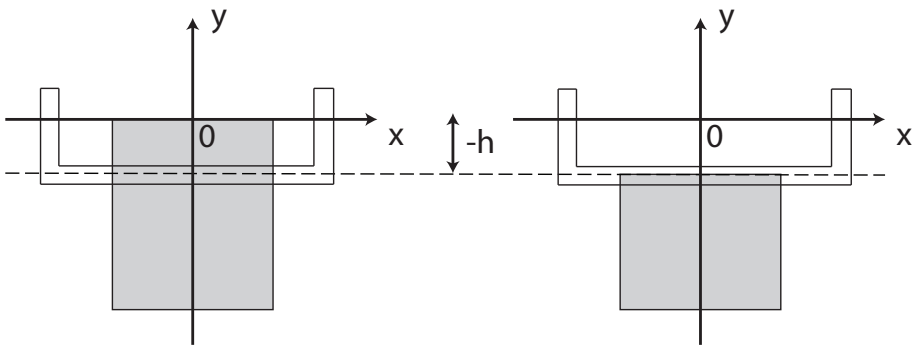
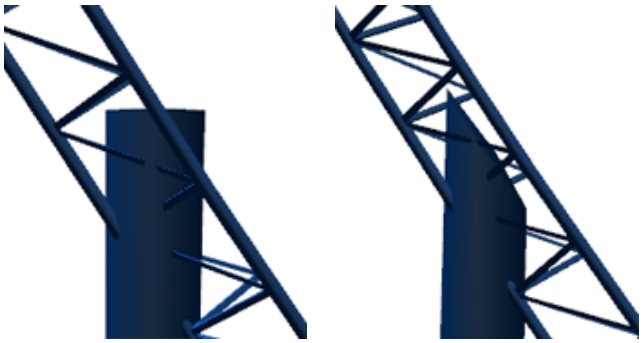


Figure 5.14: The same track profile, on the left with no pillar height adjusting, on the right with the height adjusted by the value of -h.



(a) (b)

Figure 5.15: A support pillar without (a) and with (b) tilted base depending on the track's slope.

Although the pillars look much better now, it still will not fit the track if the up vector has negative Y coordinate (train is upside down) or if the roller coaster is suspended – in these cases they will intersects the train's path. Proper placement of the supports is delegated to the user.

5.5.5 Using the local frame of reference to create track geometry

While creating the vertices to draw the supports uses its own algorithm, rails and sleepers first need to be transformed to match the spline's shape. We already know the local frame of reference at any point of the spline and we need to convert both the rail's profile and sleeper's geometry into its coordinate system. For this purpose, we can say that the X axis of the classic coordinate system corresponds to binormal of the frame, Y axis to the up vector and Z axis to the negated tangent (DirectX uses left-handed coordinate system while the frame of reference is right-handed, which means that the Z axis points in opposite direction, see Figure 5.17).

To transform vertex coordinates (and their normals) into our frame of reference, we mul-

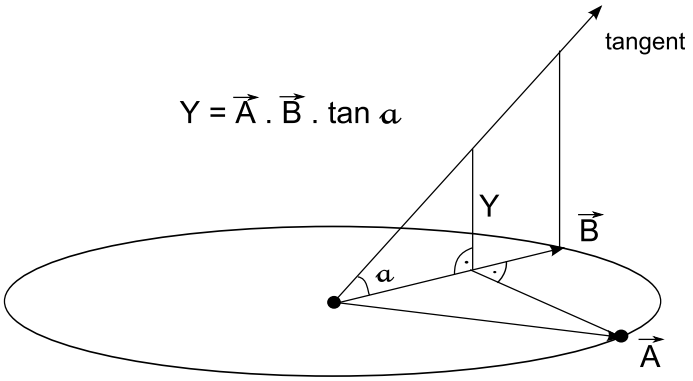


Figure 5.16: A draft of top base of pillar and calculations used to get the height Y of the point A of the base.

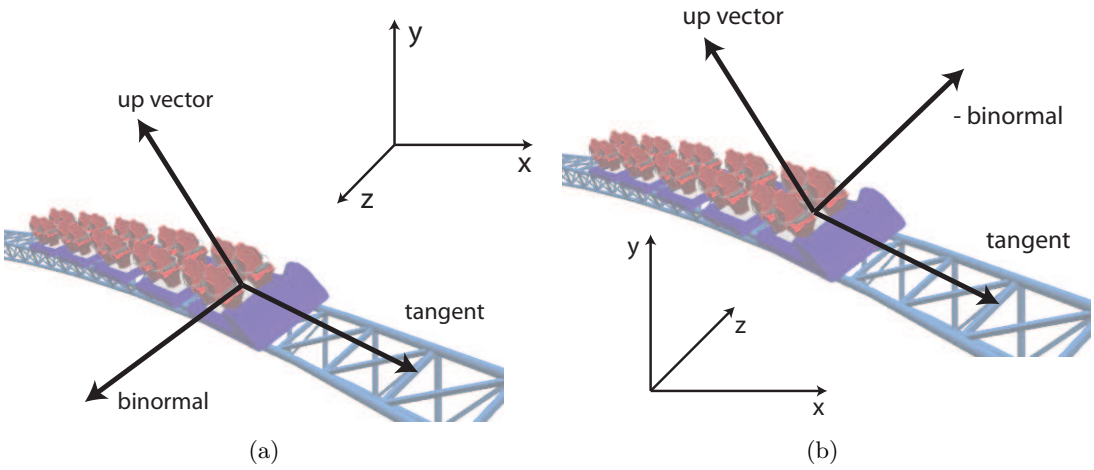


Figure 5.17: The difference between right and left handed coordinate system.

tiply each coordinate (X, Y, and 0 for profile, X, Y, and Z for sleeper) by the corresponding axis and get the result by adding all of these products. We can actually write these transformations as a single transformation matrix where the frame's vectors create its first three columns:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} binormal.x & upvector.x & -tangent.x & 0 \\ binormal.y & upvector.y & -tangent.y & 0 \\ binormal.z & upvector.z & -tangent.z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{5.1}$$

5.5.6 Representation of vertices

We are going to save all final vertices in vertex buffers and indices in index buffers. For rails, we will use one vertex and index array per segment, while for sleepers we use just one

buffer for the whole spline. These buffers are classes that send their data to graphics card where they are saved in card's local memory. If we did not use them, our program would send the data to graphics card in each frame, which would slow the application down. To use these vertices for drawing polygons, we define the first index in the index buffer, type of polygon (usually triangle) and number of these polygons to draw.

5.5.7 Level of detail

Level of detail (LOD) is a widely used method to decrease the complexity of drawn geometry and increase the efficiency of rendering. It is based on the fact that a high amount of details is not necessary on geometry that is far away from our camera (but there are also other approaches such as the importance of the object or speed the camera is moving with) and their removing will be completely unnoticed by the user. We can reduce the details by for example simplifying or removing the texture or drawing geometry with smaller amount of polygons [7].

There are two kinds of algorithms to determine the LOD: discrete LOD, which defines a finite amount of levels and the distance thresholds between them. All the objects in one region then belong to one LOD and every object has explicitly given representation for each of these levels. Then there is continuous LOD, which creates a continuous spectrum of details.

Level of detail in the Simulator

In our roller coaster simulator we will use a discrete LOD algorithm. We will focus on the parts of geometry that create the most polygons – geometry of rails and sleepers. When the spline is loaded we compute all the vertices and their normals at all available levels for the whole track. Based on the LOD, we determine what vertices should be drawn. Then with a change of conditions that the LOD determining is based on we just update what parts of what vertex buffers will be drawn. This is defined by starting index in index array and the number of polygons drawn, as described in Section 5.5.6. These values are changed when the LOD is recomputed.

The LOD of each point depends on its distance from the camera position. We will explicitly set a distance where the last and the least detailed LOD starts and divide the space in between equally to all the other available LODs.

The amount of LOD's will depend purely on the capabilities of defined roller coaster style and so will the representation of drawn geometry. We will further refer to the levels with more detailed representation as high LODs and the ones with more simple representation as low LODs.

Rails

The LOD of rails is determined per segline, because a segment can be long enough to belong to more LODs at once (imagine a very long straight segment starting at camera position), possibly even returning to a previously used LOD. Of course, segline also possess this risk, but the chance of it being so long is much smaller than with the whole segment.

We have two ways to reduce details of drawn rails. First, we decrease the number of edges in their profile we can see an example on Figure 5.18. Here we start with a circle made of 24 edges and gradually reduces it to a square.

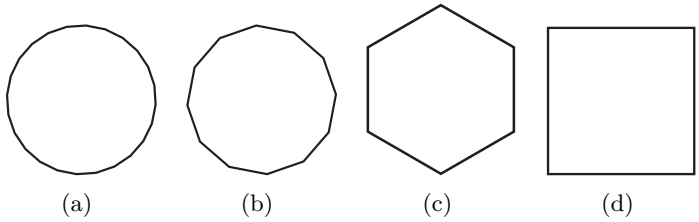


Figure 5.18: An example what removing details at rail profile might look like - the number of edges creating the profile gradually decrease from 24 to just 4.

Second, we reduce the number of seglines that a segment is drawn with. At this point we need to be careful to set this number so that the rails always connect to each other and do not create any unwanted gaps. The number of steps in higher LOD must be divisible by the number of steps in a LOD one lower so that the points where geometry switches eventually meet. On Figure 5.19 the higher LOD has eight steps while the other has just four. At point three, the level changes from 0 to 1 (the level is determined by the distance between camera position and the segline's starting point), but since the point 3 is missing on the lower LOD, we have to continue drawing level 0 until we can switch, which is at point 4. When switching back to the higher LOD again we do not need to check this restriction, it is ensured by the rule for the number of seglines.

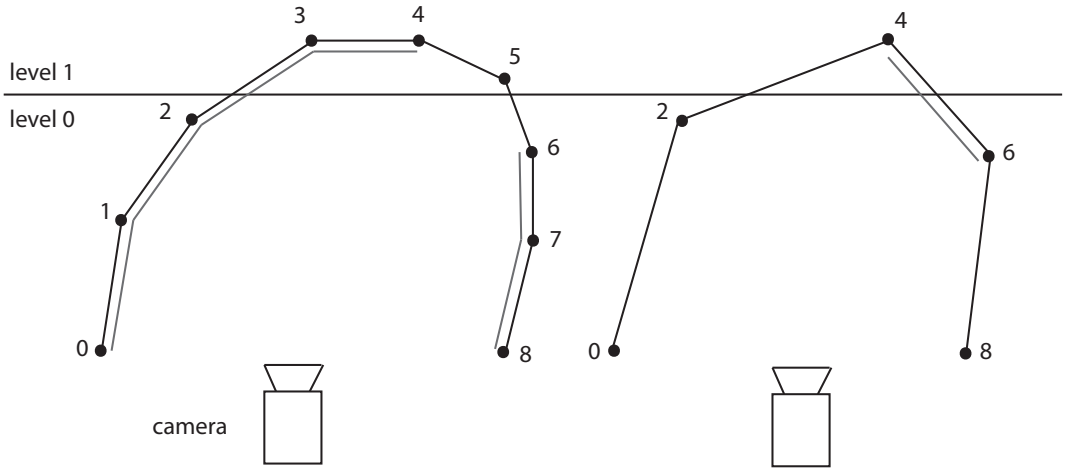


Figure 5.19: An example of dividing seglines into two LODs. Double line represents that the segline is drawn in the particular LOD.

	LOD 0	LOD 1	LOD 2
Number of rail types	3	3	3
Number of linesegs per segment	16	8	4
Number of sleeper types	6	3	3
Distance between sleepers (in meters)	1	2	4
Number of sleepers per 40 m segment	20	10	5
Total rail triangles	3840	960	240
Total sleeper triangles	960	240	120
Total triangles	**4800**	**1200**	**360**
Ratio	1	0.25	0.075

Table 5.1: The number of triangles in three different LODs on a sample roller coaster style.

Sleepers

Except for obvious geometry details reduction as used at rails we can save computing time by increasing the distance between sleepers. Unlike the number of rail seglines, this distance does not need to follow any rules. Although it is recommended to set the distance in higher LOD also as multiples of distance in lower LOD so that the change when crossing level's boundary is not aggressive. For example, with a step of 2 m and 4.1 m and LOD changing after 2nd sleeper – the distance between last sleeper of higher LOD and first sleeper of lower LOD would be just 0.1 m. We can of course set that at certain LOD the sleepers will not be drawn at all.

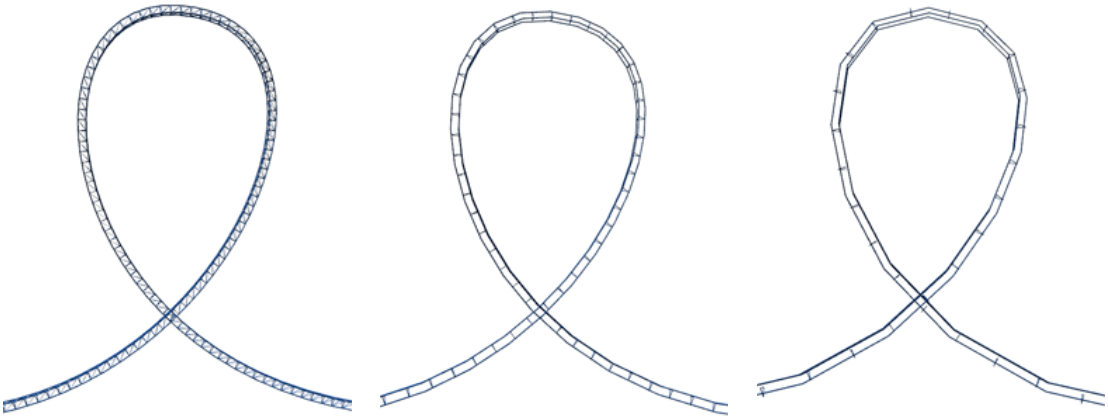
When loading the spline, all sleepers of all LODs are saved in a single list and sorted by their distance from the start of the segment. Then they are looped through and just the sleepers belonging to the LOD we currently are in are drawn.

Result

The table 5.1 shows how drawing parts of track in different LODs affect the amount of drawn triangles. As a sample we use 20 meters long segment and a triangular steel roller coaster style with three available LODs. We use all of the methods mentioned above (except for sleepers' geometry, which is the same block with 4 sides in all levels) to reduce the details of drawn segment. As we can see, we managed to reduce the amount of rendered polygons to a 25% in LOD1 and 7.7% in LOD2. The result of these modifications is shown on Figure 5.20 on a loop.

5.5.8 The virtual world

No roller coaster can exist in an empty space – it needs to have a ground to be put on and the environment around so that the passenger's ride experience is better and he does not get disoriented while going through inversions. To create the virtual world, we will render a skybox and compute how to place the roller coaster in it correctly.



(a) (b) (c)

Figure 5.20: The same loop rendered in three different LODs.

Skybox

As shown on Figure 5.21, skybox is a block surrounding all the objects in scene. In our case the only object to display is the roller coaster (we will neglect the train because its size is negligible at this moment). All 8 sides of skybox are textured by very large images that fit with each other and together create the impression of having a real world around us.

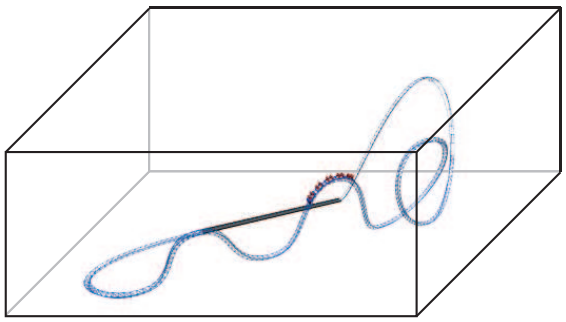


Figure 5.21: A roller coaster surrounded by a wireframe of skybox.

However the roller coaster spline can be placed anywhere in the world coordinates and be bigger than our skybox, therefore we need to place the spline correctly and adjust the skybox's size so that it fits in.

First we have to move the spline to the center of our world. For that we need a vector determining which direction and how far to offset each of the spline key points. We find it by finding maximum and minimum coordinates in every direction while iterating through the spline. The offset vector's X and Z coordinates are then an arithmetic mean of the minimum and maximum values in X and Z axis direction. We are interested in the Y axis direction separately because we do not want the roller coaster to get below surface level – Y coordinate

of the offset vector is computed as the difference between surface height and minimum Y coordinate of the spline.

To determine the skybox size we need the overall roller coaster's width, depth and height. We easily compute those from our prepared minimum and maximum spline coordinates. The skybox width and depth are equal so that the textures on the sides have the same sides' length ratio; therefore our desired horizontal size of the block is the maximum of roller coaster width and depth, multiplied by 2 because we want some free space around it. The height of the skybox is simply the maximum of the height of the roller coaster with addition of 10 meters for the same reason as before and predefined minimum height of our virtual world.

Of course if we want to display a roller coaster with absurdly large height and small horizontal size, the skybox textures will be badly deformed. But no such roller coaster exists and there is quite big margin in our resulting skybox dimensions (width and depth are multiplied by 2 while the height is just increased by a constant).

5.6 Train

Circuit roller coasters normally have several trains on ride at a time, depending on the track's and station's length. Shuttle coasters have just one for obvious reasons. In our simulator we have just one train since we do not need to maximize the coaster's capacity.

The trains differ a lot depending on the coaster's style and the cars that fit it. There are coasters with just one car creating the whole vehicle or even 10 cars, where each car can carry from 1 usually up to 8 passengers (but there are rare exceptions with even more seats). Basically it is the number and type of cars that specify what the train overall looks like.

5.6.1 Car

We need to specify several things at each car: its appearance, weight and length. While weight is important just for the purpose of computing the physics, appearance and length are what is needed for proper drawing of the whole train. We implement several options for each.

In simulator we are able to edit the train. We can have up to 5 cars, remove them, change their appearance and change their weight.

Appearance

We have two possible ways of defining the appearance of a single car. The first one is by a classical vertex buffer and index buffer as described in Section 5.5.6. But we have to insert all the vertices and indexes manually and therefore it is very hard to create any detailed good looking car.

The second option is inserting a model. In XNA we can load only models with extension .X and .FBX. FBX exporters are quite common in modeling software and we can download the convertor to the other format [4].

[4]We used kW X-port downloadable at http://www.kwxport.org/ - a free plug-in for various versions of Autodesk 3ds Max

To place either manually created model or loaded model into our application properly, there are several requirements that must be met: the front side of the car must be aligned with the X axis and the front view (in direction of Z axis) must fit well on the track profile it is supposed to ride on (Figure 5.22). Since it is not always possible (or easy at least) to save model and load it so that its position stays as we defined it, we can define the necessary transformations while creating a new cart right in the program. These are translation, scale and rotation along Y axis, performed in this order.

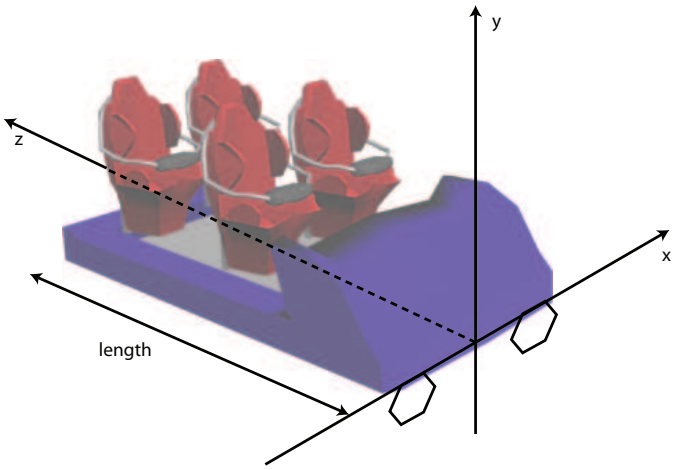


Figure 5.22: A properly placed car model.

Length

By length we mean the difference between minimum and maximum Z coordinate of all the vertices car is made of (Figure 5.22). We need this information to be able to compute the position of each car in a train, which is discussed further in Section 5.6.2.

The situation is far simpler with manually edited vertices. The length is easily found by looping through all the vertices, finding the minimum and maximum Z coordinate and subtracting them. It is more complicated with loaded models though because of the differences between the formats and modeling software.The problems lie mainly in swapped axes (for example Autodesk 3ds Max has Z axis pointing up by default) or automatically adjusting the model's size when exporting.

For getting the length of a model we could make use of the method to get a bounding sphere of a given mesh that is available in XNA. Bounding spheres are very simply represented – by a vertex representing its center and a number representing its radius. However, this will result in just an approximate length. The sphere must contain all the vertices in the mesh's vertex array and as we can see on Figure 5.23, its diameter is actually bigger than the real model's length. While this difference might be negligible at some models, we will rather not use this option.

The way to compute the exact length of the model is by accessing its vertices directly [11]. First we need to get bone transforms of the model. For example, when exporting model

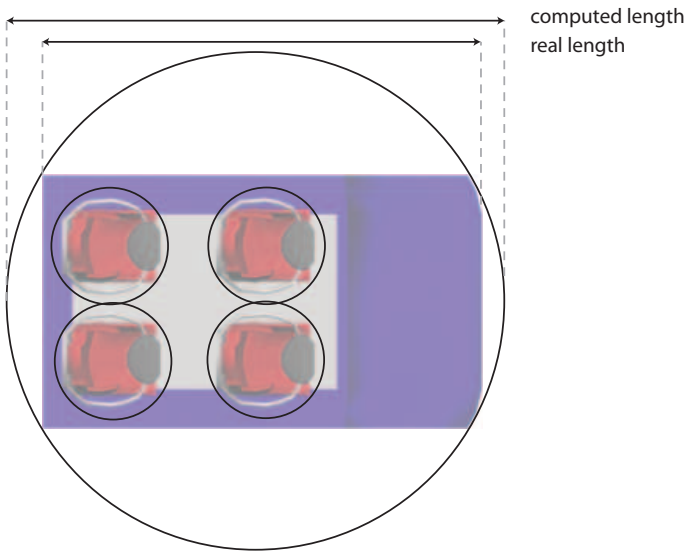


Figure 5.23: A top view of a roller coaster car and edges of the bounding spheres covering model's meshes.

from 3ds Max into FBX, this is the transformation that swaps axes and scales the model. Then we need to look at all the vertices for each model's mesh separately and transform each of them by the bone transformations and apply the Y axis rotation (defined by user), so that we work with the correct dimension. At this point we can finally access the correct Z coordinate value. Once we loop through all the vertices of all the model's meshes and get the desired minimum and maximum coordinate, we can finally count the length as the difference of those two values. At the very end we must not forget to multiply this length by the scale value (also given by user).

While this way should be a reliable way to get the model's length, we leave the option to define its length manually open. The whole process of exporting a model and loading in XNA is quite problematic and the behavior differs between modeling tools, models properties, exporting software, export settings or XNA content loading settings.

5.6.2 Putting the train on the track

When positioning the train on the spline, we are basically just positioning its cars. Since we set the rule that all the car models' front edge must be aligned with X axis, we will refer to the car position as the place where its front edge is. On a straight track the positioning is very simple: each car's position is the previous car's position plus its length plus the gap (gap is explicitly user defined) except for the first car which is at initial train position (Figure 5.24. We use exactly the same transformation matrix derived from the frame of reference at given point of the spline as described in Section 5.5.5.

However, at curved track the situation is far more complicated. Figure 5.25 shows what happens if we simply use the frame of reference of the point where the car starts. We can see that the back of the car is in the air although the cars are properly aligned with the frame.

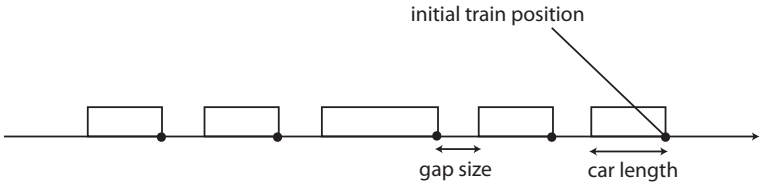


Figure 5.24: A train with different lengths of cars on a straight segment. The dots represent the positions of each car.

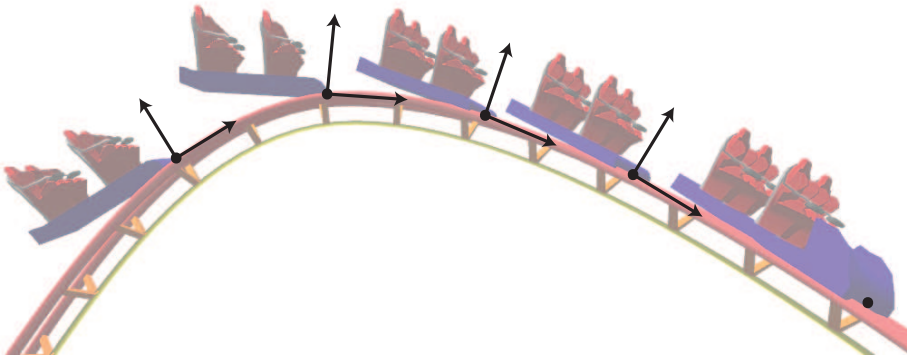


Figure 5.25: Cars positioned on the spline using the frame of reference of the front edge point. The arrows represent its tangent and up vector.

We have to adjust the frame of reference so that the front and the back edge of the car fit on the track. To do that, we first need to find where the back edge position on the spline is. We know how far the two points are from each other – it is the length of the car. But we need the real distance, not distance along the spline. We get the point by iterating with a very small step along the curve backwards, checking the distance in each step and returning the point that trespass it first.

As Figure 5.26 shows, the desired frame of reference is computed from these two points. The tangent is the vector lying on the line connecting them. The up vector is computed by interpolating their up vectors since they already include the track's roll – a linear interpolation at 0.5 will give us the up vector approximately corresponding to the desired car's orientation (if the track has any roll, the four points where the wheels should be do not lie in one plane, therefore it is impossible to position the car perfectly). At this point we also need to recalculate the binormal so that the axes of the new frame of reference remain perpendicular and normalize all of them.

Now when positioning the other cars of the train we must get their position as the back point's position plus the gap, because the curve distance is obviously bigger than real car's length. To be absolutely precise, we should compute the position after the gap size the same way we compute the position of the back point, but since the algorithm is more complex and the difference is negligibly small, we will just leave the curve distance here. The result can

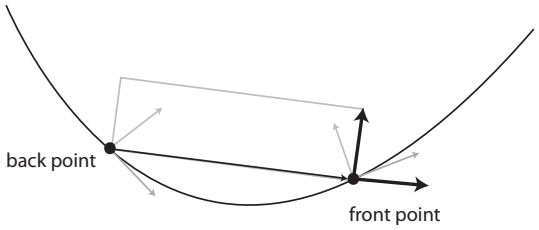


Figure 5.26: Frames of reference at front and back point of the car. The desired tangent and up vector are computed from these two.

be seen in Figure 5.27.

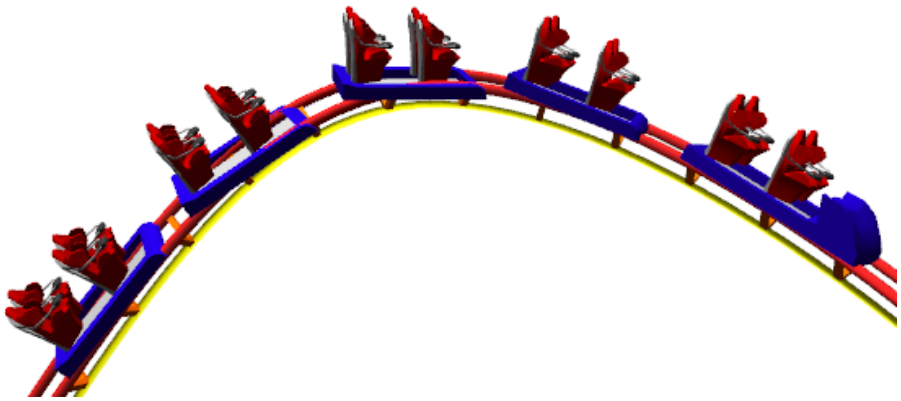


Figure 5.27: Cars placed on the spline based on the position of the front and the back edge.

So far we expected the train to be somewhere on the track. But if the roller coaster track is not circuit, it might fall off. Studying what happens to the train then is outside the scope of this thesis, but we still need to take care of this special situation. Since we take into account both the front and the back edge of each car, we need to check both of them. If one of these points is not on the spline, we simply skip the car in all the other computations (like physics) and it will not be drawn at all.

5.6.3 Train movement

In this section we will look at the implementation details of train movement, which is based on the physical model described in Section 4.2. Then we will discuss the special segment types and how can their behavior be included in this model.

Constant speed

To be able to implement the actual physics, we need to make the train move with constant speed first. We cannot simply use constant step in parameter t in each segment, because

distance between the two points acquired using t does not correspond to the same Euclidean distance. Take, for instance, spline of two straight segments, where the first is 1 m and the second is 10 meters long. The step of $0.1t$ will correspond to 0.1 m on the first segment and to 1 m on the second, where the train would move 10 times faster. The second problem is that even if the segment's length is the same, it is not ensured that the t will be evenly distributed through the segment. This matter has already been discussed in Section 5.5.2 where we used this to our advantage.

To achieve the constant speed, we need to be able to calculate the length of a spline, segment or just their part.

The length of a curve is computed numerically. We start at a certain point with the distance 0 and iterate through the curve with a very small fixed step of t – we create very small seglines that will not be drawn. Since segline is a straight line, we can easily compute its length as the distance of two vertices v_1 and v_2:

$$d = \sqrt{(v_2.x - v_1.x)^2 + (v_2.y - v_1.y)^2 + (v_2.z - v_1.z)^2} \tag{5.2}$$

and add this to the length we have calculated so far. This way we get an approximate length of the curve. The smaller the step, the more accurate the result will be.

Note that it is not necessary just for the speed, but we use this algorithm for example to draw the track sleepers. There are small variations, for example at sleepers we look for a point that is at certain distance of another point. These are all just modifications based on the same principal – approximation of the length as a sum of distance of very small seglines.

Special segment types

By special types we mean any type with special a mechanism that is not a simple track. These are normal station, shuttle station, boost, brake and lift. They affect the physical model in their own way and at certain moments of the computations.

Brake Brake slows the train's speed down. Its function applies until the train reaches the maximum speed defined by the user. In reality brake is basically a mechanism that creates a powerful friction force that acts against the direction of the motion. We can apply it in our application the same way – when the car is on a brake segment, we will add a special friction to the natural one. The strength of the brake is defined by the value of friction coefficient which is also defined by the user. Note that by setting a very high coefficient we can achieve a behavior we cannot in reality (i.e. instantly stopping the train). The brake can slow the train down no matter what direction along the track it is moving.

Lift Lifts are used to carry the roller coaster train on top of the first largest hill and gather the energy for the whole ride - it increases a potential energy by increasing the object's height. We represent lift as a segment that ensures the train's positive minimum speed (negative would mean the same speed but opposite movement direction) which can be defined by a user. The limitation of this implementation in our application is that the minimum speed is set no matter how heavy the train is, while in reality lifts are made to carry a train of some realistic weight and would not be able to lift anything significantly heavier.

Boost Boost is a part that speeds the train up with a powerful force. They are usually used at the start of the track right after the station to give the train speed for the ride - it increases a kinetic energy by increasing the speed - instead of lifting it up on the initial hill first. This segment type adds a user defined positive force (it boosts only forward) to the other forces affecting the car at that moment causing the train speed to increase. The higher the force, the higher the resulting acceleration will be.

Station A station is more complicated than the previous three segment types, because it combines them together. When the train is entering the station, it acts like a brake and slows it down. After it is slowed to the required speed, it behaves like a lift and carries the train to the end of the station to prepare for another ride - the train waits until the passengers get in and out. Once it is ready to leave, the station's behavior changes to either lift (in a normal station case) or boost (shuttle station case).

To implement the station properly, we need to implement a variable telling us what the current behavior of the segment is. At initialization, we set it to brake. Then we set the current behavior depending on the train position and speed as described.

The physical parameters the user can set for a station is derived from what it can act like – brake and lift for both and boost for shuttle station. Since we do not have any passengers to wait for, our train waits just a short fixed amount of time once it reaches the end of the station.

Friction direction problem

As already described, friction acts against the movement speed. However, we cannot simply give it a negative value, because our trains can even go backwards. For example, if they do not have enough energy to reach over the top of a hill, they start moving backwards. With a fixed negative value of friction force, it would actually speed the train up when going backwards, which is undesired. Or if the train stopped on a straight track only due to the friction force, it would start going backwards.

When we set the direction of friction force depending on the current train's speed (or to 0 if the speed is 0), the train will hardly ever stop because the chance of speed being exactly 0 is very low. Instead, the train would start oscillating between low positive and negative values of speed and so would the friction.

To be able to stop the train completely, we need to define when the train stopped only due to the friction force. We detect this in a moment when the train's movement direction changes. If it changes and the absolute value friction force is higher than the absolute value of current gravitation force affecting all of train's cars, we set the new speed to 0.

5.7 Editor

Next to the simulator, the editor creates the second part of the application. While it may seem to be a half of the application, it actually mostly just uses the functionality we have implemented so far. The difference is that this time it is used by a common user instead of programmer; therefore the functionality must be wrapped in user friendly and understandable tools.

5.7.1 User interface

Creating the user interface (UI) is a very important part of an application. Quality of interface has a big impact on the program's popularity. It also usually is the main factor for a user, who is deciding between several applications of the same function, and can play even bigger role than the functions they offer.

The designing of the user interface should go through several stages, or even better – iterate through them over and over until it is perfect. Sometimes it is called a spiral development (Figure 5.28). We analyze the situation, design a suitable interface, create a prototype, test it, analyze the results and correct the mistakes in the new design. Then the process repeats over and over and the user interface is getting more and more perfect.

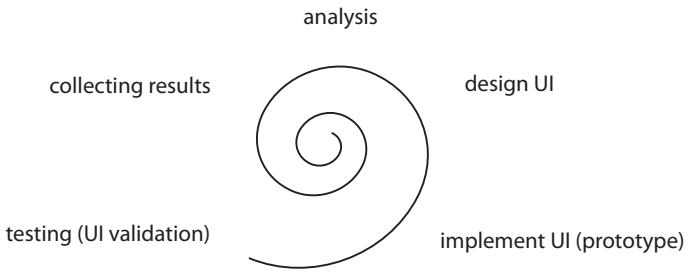


Figure 5.28: The spiral development of UI.

One of the most important rules when designing a user interface is actually very comfortable for us – we should not try to create new control tools than the users are used to from other programs, unless they are way better. Since the perfection of the interface is not the main goal of our application, we can rely on using the standard methods and inspire ourselves in already existing software of similar function, as described in Section 3. We therefore skipped many of the recommended stages such as creation of prototypes and testing. The UI creation consists just of the analysis (which we have already done in Section 4.3), design, and implementation.

Designing the user interface

The basic form of a UI design is a paper mock-up. A mock-up is usually the first stage of designing a UI. Although it is called paper mock-up, it does not need to be on paper, it rather suggests its complexity. This stage consists in a simple draft describing where and how the functions will be available to a user.

Inspired by numerous 3D modeling applications, our editor will consists of the window where the user can see the edited track and move around in 4 kinds of views: perspective view with free walking and three orthographic views – top, side and front. The technical details are discussed in Section 5.4.1. The second part of the UI will be a side panel, which contains information and setting controls that we cannot fit into the view window (Figure 5.29).

View window The view window will cover the most of window's space. A user will see the spline he is editing. However, the vertices of the roller coaster style are recalculated every

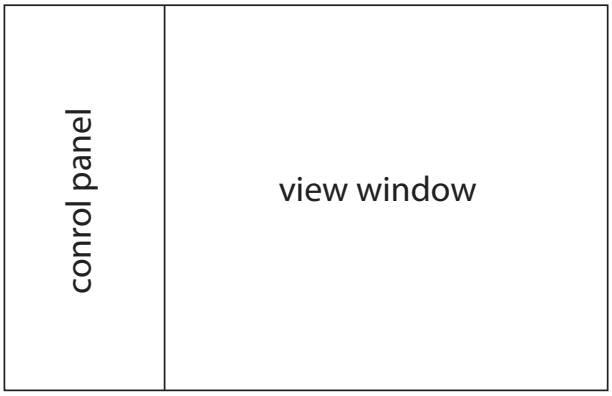


Figure 5.29: An application window divided into the view window and the control panel.

time the spline is changed, which is obviously happening quite often, and it would slow the application down a lot. For this reason, we leave this preview option open, but create an extra visualization: a schematic representation. It consists of colored lines representing the track shape and the up vectors and colored cubes representing the key points and auxiliary points of the segments (Figure 5.30). This information gives the user all the information he needs for editing the roller coaster while the response time of the application stays short.

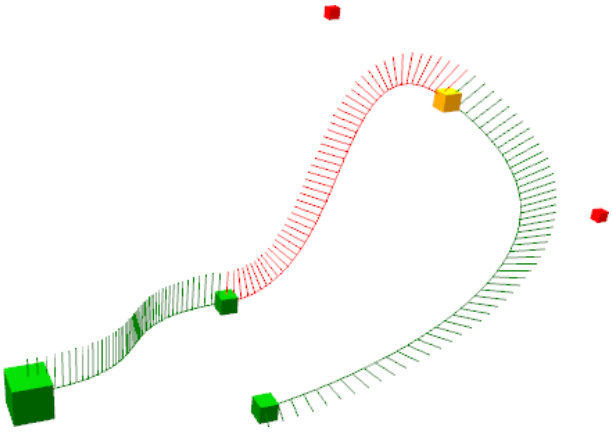


Figure 5.30: A schematic representation of a spline. The biggest cube represents the first key point of the spline. The middle sized cubes are the other key points and the small cubes are the auxiliary points.

The key points are selectable by a mouse click. Once they are selected, their auxiliary points also appear and are also selectable. The user can move them with a method called drag and drop – the point position is moving with the cursor while the mouse button stays pressed. If the user moves one of the auxiliary points, the other one moves as well. That is because the spline continuity (described in Section 4.1.1) must be preserved.

By selecting a key point, the user also automatically selects an adjacent segment, where the point is its ending point (with one exception at unclosed circuits where the first key point

selects the first segment of the spline). This is because the roll of the segment appears to user to be the roll of its ending point.

All of the selections are represented by colors of the cubes and lines, and are easily modified. Clicking in the empty space deselects everything.

Control panel At this point we should divide all the tools into logical groups. These groups will be visually isolated and help the user orient better in our application or find the tool he is looking for.

- Roller coaster settings will display all the information and settings valid for the whole roller coaster. These are for example roller coaster style, name or initial train position.
- Segment settings will show just the information about the selected segment. This group can be further divided into:
  - useful information, such as the dimension of the segment or the position of the selected point,
  - roll editing tools,
  - segment type settings, where the user can set the type of segment and the adjustable physical values.
- Spline modifying tools, which will contains the functions related to modifying the spline's shape (e. g. adding and removing segments).

Implementing the user interface

In this Section we will focus on how to implement picking and drag and drop function. We are not describing the other UI features since they are fully provided by Winforms framework and require no special attention.

Picking Picking [12] is a method that provides the user to select an object by clicking on it. Whenever the mouse button is pressed, all the selectable objects are looped through and checked whether they were hit or not. There are special data structures to optimize the complexity of this algorithm (such as BVH - bounding volume hierarchy), but our application is very simple and contains too few selectable objects, to make use of these.

All the selectable objects must have a bounding volume – a volume with as simple representation as possible, that surrounds the object's geometry as accurately as possible. The most basic bounding volumes are bounding box or bounding sphere (Figure 5.31), which are sufficient for our purposes. For advanced bounding volumes and their use, see [6]. These volumes are used for computing whether the ray created by a mouse click intersects them or not, which is also the reason for the necessity of simple representation.

The only selectable objects in our applications are cubes representing points (key points or auxiliary points). Each of them has its own bounding volume – we choose bounding sphere because it is simpler. A sphere is represented by its center and its radius. It does not need

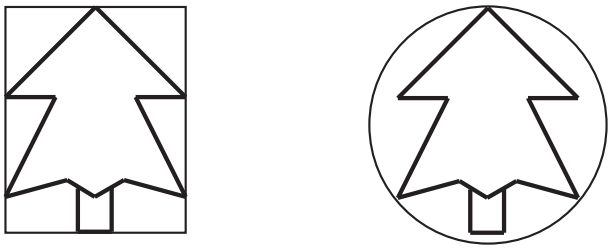


Figure 5.31: A tree model surrounded by a bounding box and a bounding sphere

to be rotated (because rotations do not affect a sphere at all), we only need to translate the center into the point's position and set the radius according to the cube's size.

The ray is calculated when the mouse button is pressed. Each vertex in the world is transformed by view, projection and viewport transformations. This time though we need to invert the process. We have the position of a mouse in the viewport and need to know the two sets of world coordinates that define the ray. XNA's method Unproject, provided with the viewport location (with a Z coordinate being 0 for the close coordinate and 1 for the far coordinate) and the three matrices performs the inverse transformation for us. The ray is then represented by the near coordinate and vector pointing in the direction of the far one (Figure 5.32).

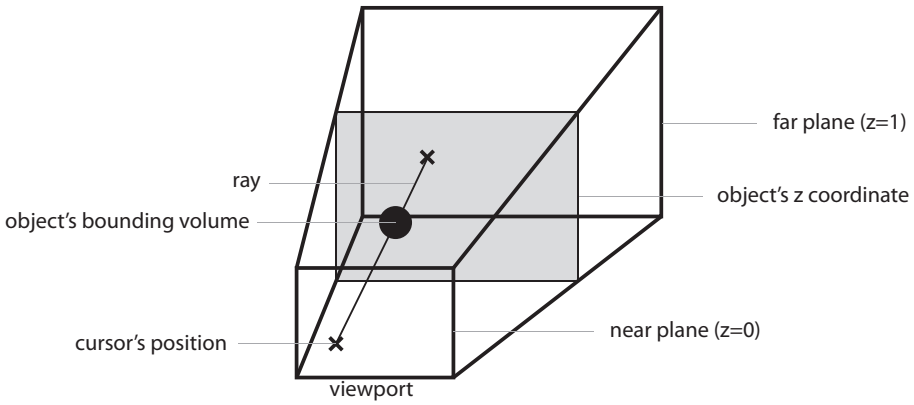


Figure 5.32: A scheme of a view frustum, bounding volume of an object and ray defined by two vertices (represented by crosses on near and far plane).

The ray and properly transformed bounding sphere are all the information we need to know in order to detect whether they intersect or not. Again, XNA has a method that computes this for us. We cannot select more points at a time – only the first hit point in the loop is selected and all the others set as not selected (this is needed for deselecting and drawing the cubes with the proper color).

Drag and drop Drag and drop is a method that moves an object with the mouse movement when it is selected and dragged and stops moving it when it is dropped. In our case, the

drag means that user holds the mouse button and drop that he releases it.

When moving an object, we need to know what its depth on the viewport is first, so that we can keep it and the object follows the mouse's movement accurately. This is achieved by transforming the object's position by view, projection and viewport matrix (XNA's method Project) and getting the Z coordinate, which is, at this point, between 0 and 1.

The movement (drag) is defined by the change of cursor's position on the viewport – current mouse position and the position in previous Update function call. Since these two positions, with the Z coordinate calculated above, are positions on viewport, we need to perform the inverse transformation (described in Section 5.7.1) to get the two world coordinates. Their subtraction is the desired vector defining the offset of the selected object.

5.7.2 Editing tools

Since the editor just uses the functions that have already been implemented in other parts of the applications, there is not much to cover in this Section. However, a few tools deserve a special attention.

Initial train position

Although the initial train position would normally be at the end of the station, the application does not restrict a user from creating a roller coaster with more stations. The safest way to set this position is to let the user set it himself.

The position is given in a real distance and is limited to the spline's length. While the UI element allows entering negative value, program recalculates it in order to keep it always inside this limit. The same happens with editing the spline and its length decreasing. Editor always ensures that the initial train positions remains valid.

Editing support position

Every segment can have one support pillar. Its position is defined as parameter t of the curve creating the segment, and thus belongs to interval $\langle 0, 1 \rangle$. We chose this definition because of the more intuitive behavior while changing the shape and the length of the segment. E.g. pillar at t=1 will always be at the end of segment, t=0 at its start and the values in between will vary depending on the segment's shape.

Roll snap

While the up vectors behave intuitively most of the time, somteimes, they differ from the ones the user would intuitively expect – the up vectors expected to point straight up can be slightly inclined. For this purpose, we enabled a feature to snap the up vectors either to straight up or straight down direction. This feature automatically calculates the roll of the segment needed for the end point's up vector pointing in the desired direction.

The calculations are drafted on Figure 5.33. Using dot product, we calculate the angle α between the vector pointing straight up and vector a being the up vector that we want to straighten. To determine whether we want to add or subtract this angle from the roll,

we calculate the cross product, which results in a vector either in the direction of tangent or in the opposite direction. To cover some numerical inaccuracy we calculate another dot product of this cross product result and tangent, which returns positive number if they point in the same direction from the plane given by vectors a and up, negative number if they point in different directions.

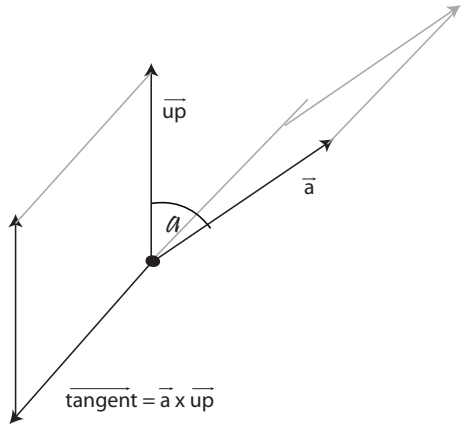


Figure 5.33: A scheme of rotating vector $\vec{a}$ in the position of $\vec{up}$.

Adding new segment

New segment is always added to the end of the spline (the feature is disabled if the spline is a closed circuit). It is created based on the last key point and its adjacent auxiliary points. As shown in Figure 5.34, the new key point's position is calculated as A + 2.5·$\vec{d}$, with $\vec{d}$ being the vector pointing from the last key point to the last auxiliary point (the one prepared for the new added segment).

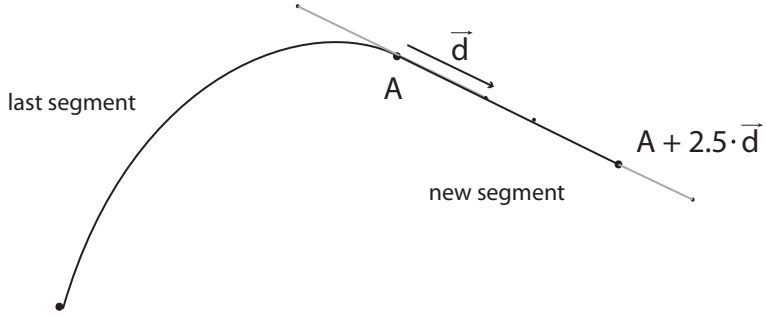


Figure 5.34: Last segment of the spline and a new segment attached to it.

Finishing circuit

This tool automatically inserts a segment between the first and the last key points. All the key points and their auxiliary points are already prepared, so we do not actually need

to calculate anything regarding its shape.

However, we must adjust the roll, so that the up vector of the last and the first segment match. For this we use the same procedure as used in snapping up vectors in Section 5.7.2.3. We only do not snap to straight up or straight down, but to the starting up vector of the spline.

Attaching existing spline

This tool covers the requirement of attaching prefabricated parts to our roller coaster. The prefabricated parts are nothing else than another spline saved as a normal roller coaster. Of course, both the attached spline (further referred to as spline B) and the one it is being attached to (spline A) must not be closed. The attached spline must be transformed according to the end of the edited one – in other words, their frames of reference must match. An example can be seen in Figure 5.35.

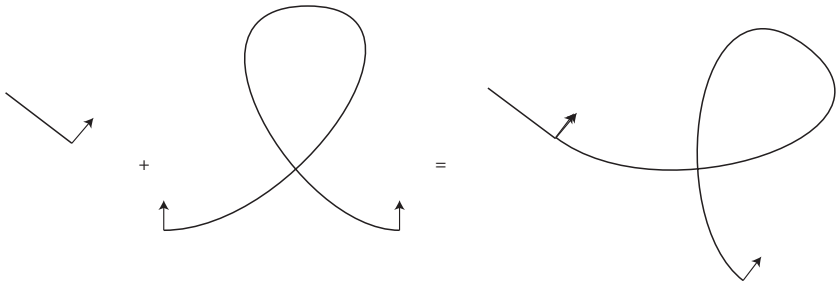


Figure 5.35: An example of attaching a loop at the end of spline. The frames of reference (here represented just by the up vectors) must match.

As in many other cases, we will use the frames of reference defined by tangent, binormal and up vector, to perform the transformation. First, we offset all the points in the spline B so that its starting key point is in the center of coordinate system – (0, 0, 0). Then we convert all of the points' coordinates in universal coordinates x, y, z by expressing them in the coordinate system defined by the frame of reference. As shown in Figure 5.36, we use dot products.

To transform the spline B so that it starts with the same frame of reference as the one at the ending key point of the spline A, we use the algorithm described in Section 5.5.5. At the end we offset the spline B so that the key points to be merged (the last key point of spline A and the first key point of spline B) have the same position. Finally, all the segments are ready to be added to spline A.

Preserving the continuity

As described in Section 4.1.1, the desired spline continuity is ensured by the rule saying that the auxiliary points attached to the shared key point of two adjacent segments must lie in one line with the key point position. We need to preserve this rule in editor when the

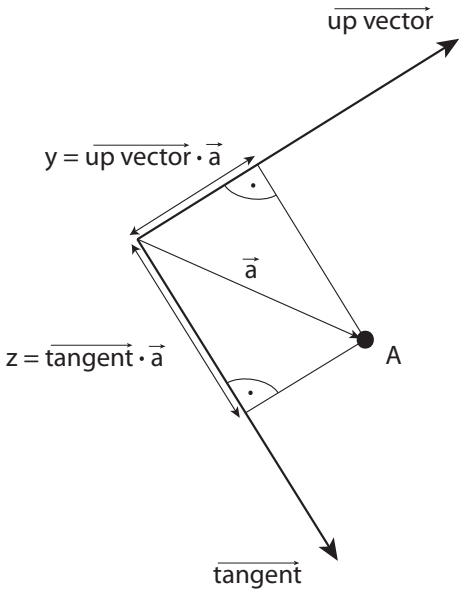


Figure 5.36: Expressing the coordinates of vertex A in coordinate system given by the frame of reference (here represented just by up vector and tangent).

user is changing the position of one of the auxiliary points. Whenever the point's position changes, we must recalculate the position of the other auxiliary point.

The process is shown in Figure 5.37, where A represents the point the user is moving with and B represents the other one. First, we calculate the distance d between key point and point B, because it is the value we want to keep. If the user wants to keep C1 continuity, we calculate the distance between key point and point A instead.

Because the three points (key point, A, and B) must lie on the same line, the new point B will lie on a line defined by key point and a vector a. To preserve the length, we normalize this vector and multiply it with d, which results in the final position of point B.

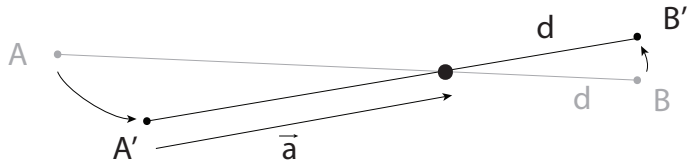


Figure 5.37: of point B's position according to the change of the position of point A.

Surface level

We implement an extra restriction that will ensure that the track will always stay above the defined surface level. We need to check this in all the tools that modify the spline's shape. We do not allow positioning any key point below the surface, we modify newly added

segment so that it stays above and we show an error message if an attached spline reaches below this Y coordinate. The attached spline is not modified because it would deform the shape which we usually want to preserve when adding a predefined piece.

The implementation has the limitation that it only checks for the key points' position while any of the coordinates between them may reach below the surface. This may happen in editor, but as described in Section 5.5.8, in simulator we are making sure that the entire track is above ground.

Chapter 6

Result

We demonstrate the result of the applications on Figure 6.1 and 6.2. We tried to model the real Blue Fire roller coaster in Europa park, Germany according to pictures and videos. This is a shuttle roller coaster which contains various complicated elements such as loop, twist or zero-G roll which makes it a suitable roller coaster to test the application on.

We created the roller coaster style according to the real style of Blue Fire, both its shape and its colors. We also designed the less detailed versions of this style for displaying lower levels of detail.

Both roller coasters have the height of 38 meters and reach the maximum speed of 100 km/h. We managed to edit all the elements and copy the train behavior including the two stations, boost and brake.

Figure 6.3 shows a sample of an on-ride camera including the HUD displaying the current train speed, maximum speed reached during the ride and the current g-force in a vertical direction.

Figure 6.4 shows the progress of editing Blue fire roller coaster in the editor. We can see the final editing controls and the schematic view of a detail of the track.

Figure 6.1: Photo of Bluefire, Europa park, Germany [8].

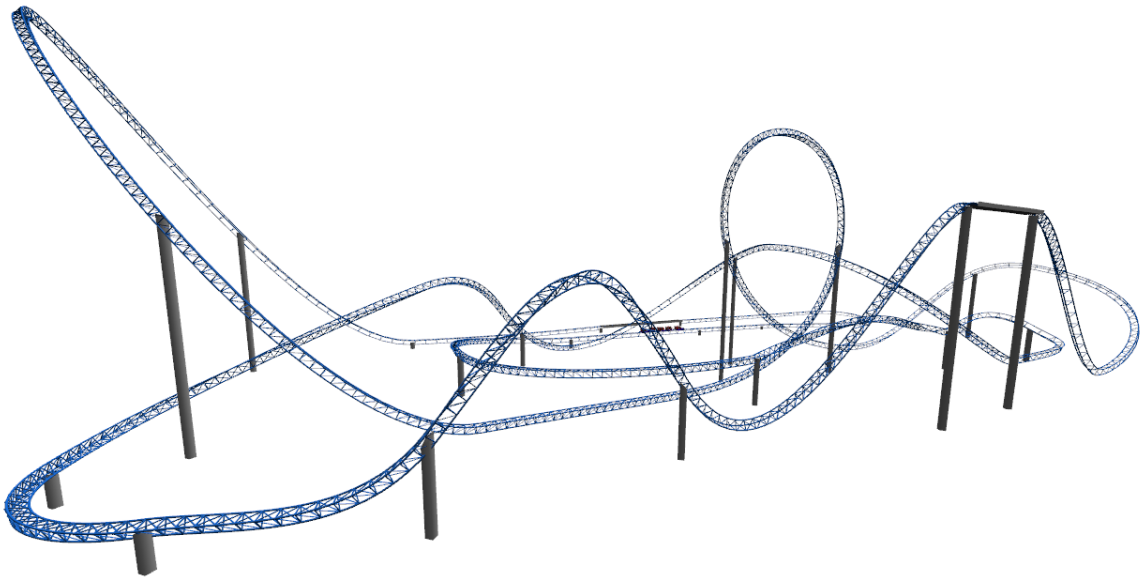


Figure 6.2: Blue Fire modelled in the editor based on photographs, videos, and personal experience. The skybox is temporarily disabled for better visibility of the track itself.

Figure 6.3: An on-ride view from the first car of the train, taken just before entering the twist.

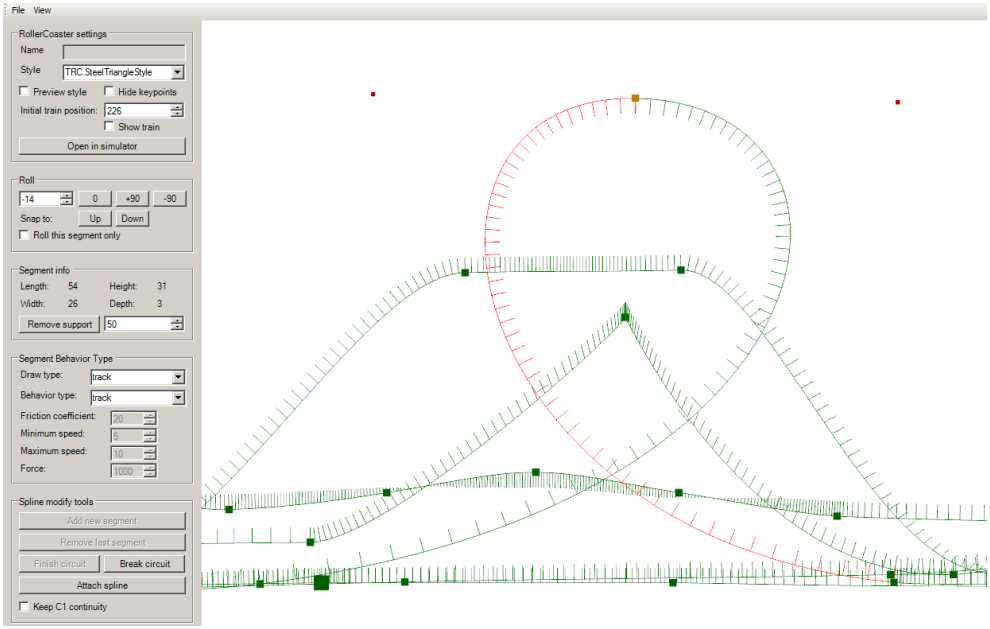


Figure 6.4: Print screen of a schematic view of a part of Bluefire in the editor. Orthographic view is currently used.

Chapter 7

Conclusion

In this thesis we created a roller coaster editor where the user can design most of the existing roller coasters. The editor provides all the tools required for this task and user can freely change the roller coaster style. The simulator uses functional physical model for the train movement, including the correct behavior of all the special types of segments. The train's cars appearance and weight is editable. Application is easily extendable by new roller coaster styles or types of cars.

7.1 Future work

Roller coaster editor can go through years of development and still not provide everything any user might want. While we created a sufficient and well working application, there are still many handy features missing:

- Extra spline modifying tools in the editor: especially the ability to select and move more key points at once, split a segment into two or being able to set a fixed absolute roll in certain points of the segment that will not change by further modifications.
- Advanced track supports. We have implemented only the basic support pillars that do not respect the track's orientation or lack any realistic geometry for the user to choose from.
- More realistic environment could be achieved by adding sounds, editing the height of surface and enabling inserting of trees and other various models that support the visual theme of a roller coaster.
- The application should check roller coaster validity. The track must not intersect, must have some maximum curvature and other restrictions.
- Virtual ride experience could be greatly improved by providing a stereoscopy support.

Bibliography

[1] Samuel R. Buss. *3D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, 2003.

[2] Andrew J. Hanson. *Visualizing Quaternions*. Morgan Kaufmann, 2006.

[3] Sadri Hassani. *Mathematical Physics, A Modern Introduction to Its Foundations*. Springer-Verlag, New York Inc., 1999.

[4] Jiří Žára; Bedřich Beneš; Jiří Sochor; Petr Felkel. *Moderní počítačová grafika*. Computer Press Brno, 2004.

[5] Wendy Stahler. *Beginning Math and Physics for Game Programmers*. New Riders Publishing, 2004.

[6] Gino van den Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann, 2004.

[7] David Luebke; Martin Reddy; Jonathan D. Cohen; Amitabh Varshney. *Level of Detail for 3D Graphics: Application and Theory*. Morgan Kaufmann, 2002.

[8] Published fan photos - blue fire panorama. [online]. `http://www.epfans.com/up-down/panorama_22_03_09.jpg`.

[9] First person camera xna tutorial on msdn. [online]. `http://msdn.microsoft.com/en-us/library/bb203907.aspx`.

[10] Game state management xna sample. [online]. `http://creators.xna.com/en-US/samples/gamestatemanagement`.

[11] Xna bounding box from a model tutorial. [online]. `http://www.toymaker.info/Games/XNA/html/xna_bounding_box.html`.

[12] Picking xna sample. [online]. `http://creators.xna.com/en-us/sample/picking`.

Appendix A

User manual

A.1 System requirements

- Operating system MS Windows Vista Service Pack 1; Windows XP Service Pack 3; Windows 7.
- A graphics card that supports DirectX 9.0c and Shader Model 1.1.
- Installed Microsoft XNA Game Studio 3.1.

A.2 Installation instructions

No installation is required. The application can be run by executing the file RC.exe.

A.3 Controlling the application

A.3.1 Menu navigation

Key	Action
<enter>	select
<esc>	cancel/exit
<down arrow>, <up arrow>	move selection
<left arrow>, <right arrow>	change settings

Table A.1: Menu navigation

A.3.2 Controlling the editor

Key	Action
<left mouse click>	select
<right mouse click>	switch to camera rotating mode
<mouse movement>	rotate camera
<w>	move forward/zoom in
<s>	move backwards/zoom out
<a>	move left
<d>	move right
<q>	move up
<r>	move down
<esc>	exit

Table A.2: Editor controls

Legend for the editor panel in Figure A.1

1. Roller coaster name (will be set automatically according to the file name).
2. Roller coaster style – the way the track looks like.
3. Turning the style preview on and off (editing with preview turned off is recommended for the fast application feedback).
4. Hide/show the cubes representing the key points.
5. The initial position front edge of the first car of the train when starting the simulation in a real distance (in dm).
6. Hide/show the first car of the train.
7. Shortcut that will directly open currently edited roller coaster in simulator. Please note that the roller coaster will not be saved to hard disk automatically.
8. The roll of the selected segment in degrees.
9. Snap tool will compute and set the roll necessary for the up vector of the end of the selected segment to point straight up or down.
10. Local roll toggle – if selected, the modified roll will only be applied locally and the following segments' roll will not change.
11. Set roll to 0∘.
12. Add 90∘ to the current segment's roll.
13. Subtract 90∘ from the current segment's roll.
14. The dimensions of the selected segment (in meters).
15. Adds or removes the selected segment's support.

16. The position of the support in parameter t (t=0 at the start of the segment, t=1 at its end).
17. The geometry type to draw the selected segment with.
18. The behavior of the selected segment.
19. Additional settings for selected behavior type in basic units.
20. Adds a new straight segment at the end of the spline.
21. Removes the last segment of the spline.
22. Inserts the last segment that closes the circuit based on the first and currently last segment.
23. Breaks the connection between the last and the first segment of the circuit spline.
24. Opens a file open dialog and attaches selected spline to the currently edited spline. Note that neither roller coaster can be a closed circuit.
25. Keeps the distance of two auxiliary points and the key point equal (useful tool for editing symmetric parts of a roller coaster).

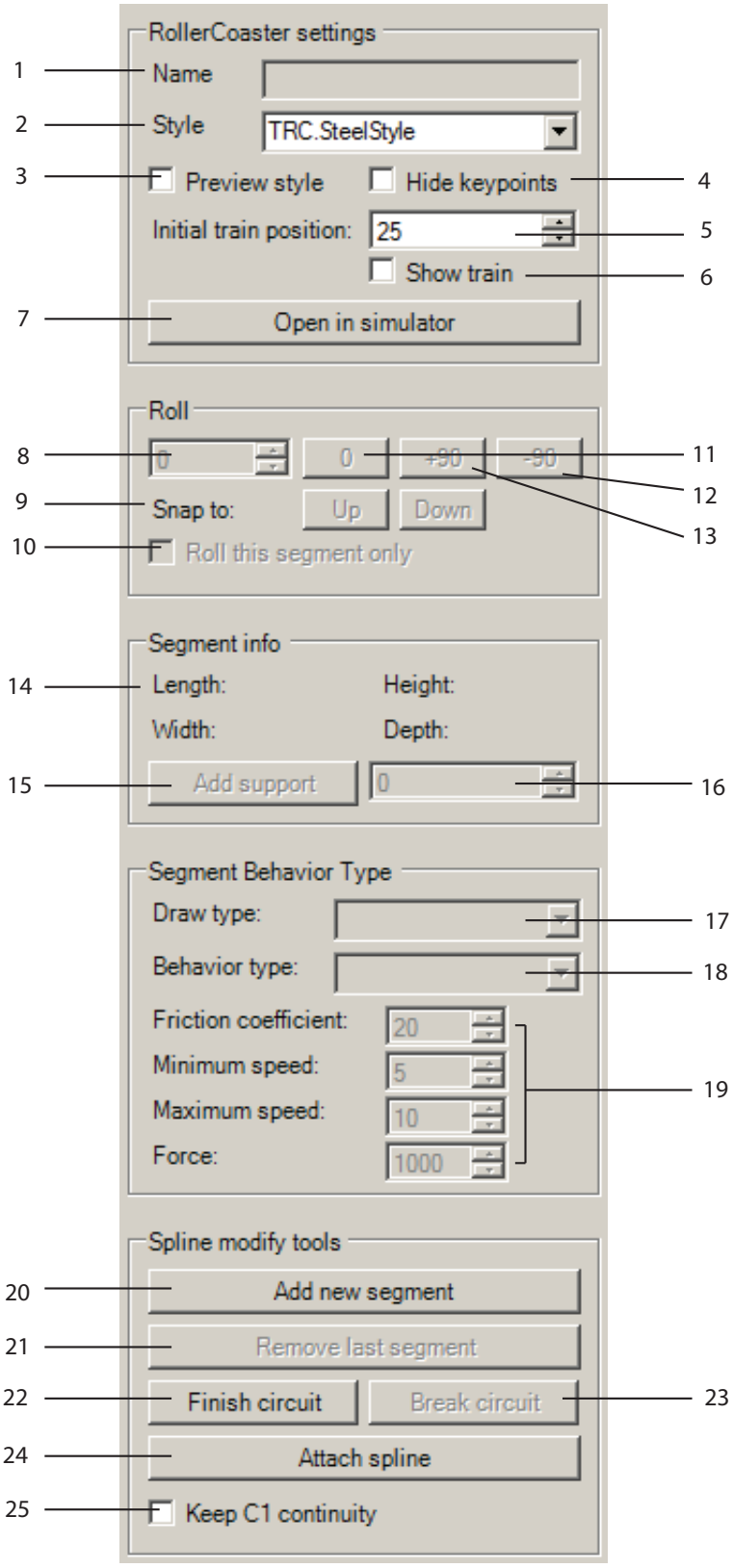


Figure A.1: The editor panel.

Appendix B

DVD Content

The following folders are to be found on the attached DVD:

`DVD`	
`|-- bin`	contains an executable file
`|-- src`	source files of the application
`|-- thesis`	
`|  |-- src`	source files of this text in LaTeX
`|  \ -- thesis.pdf`	PDF version of this text
`\-- index.html`	DVD guide