Oliver Kayser-Herold, Andreas Keese, Markus Krosche, Martin Krosche, Oliver Pajonk

# Achterbahn-Editor/-Simulator

**Documentation for the software development internship at the Institute for Scientific Computing**

**Institute for Scientific Computing**

 Carl Friedrich Gauss Faculty

**Technical University of Braunschweig**

Version: 1.0.3 für das Sommersemester 2009¨

Revision: 2284

Erstellt am 30. Marz 2009¨

Braunschweig, 2009

## Table of contents

## table of contents i

## 1 Introduction 1

## 2 About the document¨ 1

## 3 project description 1

## 4 Modeling of the route 2

## 4.1 Basics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

## 4.2 Cubic B´ezier curves . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

## 4.3 Calculation of control points . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

## 4.4 Entire Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

## 5 Modeling of dynamic behavior 7

## 5.1 Conservation of Energy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

## 5.2 Possible model extensions¨ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

## 5.3 Geometry derivations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

## 5.4 Time Integration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

## 5.5 Initial values for the time integration¨ . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

## 6 Runge-Kutta method for time integration 10

## 6.1 Euler's Polygonal Method . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

## 6.2 The fourth-order Runge-Kutta method . . . . . . . . . . . . . . . . . . . . . 11

## 6.3 Further procedures¨ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

## 7 The simulation algorithm 13

## 7.1 Sample implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

## 8 XML files 14

## 9 General information 15

## literature

## 1 Introduction

## The Institute for Scientific Computing presents two linked project topics for the software development internship (SEP) 2009. On the one hand, a CAE simulator is to be developed that simplifies a roller coaster, but simulates it mathematically correctly and visualizes it three-dimensionally (sub-project roller coaster simulator). On the other hand, a CAE editor is to be implemented for the comfortable, engineering-style construction of roller coaster curves (sub-project ¨ roller coaster editor“).

## This document contains the project description, a detailed explanation of the mathematical basics of modeling and simulation, a brief description of the document exchange format and general information on the project. It thus forms the basis for the overall project; special information on the two sub-projects can be found in separate documents [4, 5]. In addition, it is essential that both groups also familiarize themselves with the other's task description in order to understand the overall project and to be able to work towards a common goal.¨

## Additional information on the organization can be found at¨ http://www.wire.tu-bs.de and the corresponding information page of the Institute for Software Systems Engineering (SSE),¨ http:

## //www.sse-tubs.de/teaching/ss09/sep.

## 2 About the document¨

## This document contains many hyperlinks, usually marked in blue. Use this first to get additional help if you have questions or simply to get further and more detailed information.

## 3 Project Description

## Everyone knows the classic roller coaster that can be found at fairs or similar events. In most cases, the roller coaster car is pulled by a drive over an initial incline to an initial height. From there, the car continues to run independently on the specified path under the influence of gravity. Physically, this is an interplay between potential energy (at the higher points of the orbit) and kinetic energy (at the lower points of the orbit). This interplay is controlled by the course of the track.

## In our SEP, two CAE tools are to be developed for this type of fairground attraction: an editor, which can be used to conveniently design routes, and a visualizer, which can display these routes in three dimensions and carry out a simulated roller coaster ride on the route .¨

The visualization and the editor require a mathematical representation of the trajectory, ie the geometry of the trajectory. In this practical course, the course of the track is composed of individual track elements. The individual line elements are described by cubic polynomials. The resulting space curve piecemeal composed of polynomials is called a spline. With their help, the typical elements of a roller coaster such as loops and steep curves can be easily implemented. The exact procedure is explained in detail in section 4 on the next page.¨

The position and shape of the individual track elements is defined by the user in the editor and stored in an XML file, which the simulator then reads. A suggested format is given in section 8 on page 14. In addition to the course of the route, further parameters such as the initial position and the initial speed of the train and the strength of the gravitation are required for the physical model. These and possibly other simulation parameters¨ are to be read in from another XML file. The format of this file is also explained briefly in section 8 on page 14.¨

In abstract terms, the roller coaster is a continuous dynamic system. At any point in time, the state of the train is determined by its position in space - the position vector - and its speed - the speed vector. In the course of the journey, this condition changes continuously due to the forces acting on the train. Within the scope of the practical course, only the gravitational force that accelerates the train towards earth is taken into account. Additional forces, such as air resistance, friction on the rail or energy supply from a drive, remain unconsidered for the time being - these increase the complexity of the modelling. Mathematically, the dynamic behavior of the train is represented by an ordinary ¨

Differential equation (ODE for ordinary differential equation) describes

which is derived and explained in detail in section 5 on page 7.¨

The calculation of the actual temporal behavior of the orbit, which is due to the geometry and the physical modelling, is done by a numerical solution method. This takes over the so-called time integration of the differential equations and is explained in detail in section 6 on page 10

**4 Modeling of the route**

As the first ingredient for the roller coaster in the computer we need the mathematical description

”

of the trajectory. Since we want to develop CAE tools, this should correspond to reality with sufficient accuracy. This specification gives us some geometry requirements.

In order to keep the forces that act on the passengers during a roller coaster ride within an acceptable range, the route of a roller coaster must meet certain requirements

fulfil: since we want to model a real rollercoaster route, the route must be without¨ holes“.

" be. This means that the trajectory must necessarily be closed - especially at the support points. Mathematically, this is a so-called continuity requirement. Likewise, for physical reasons, the speed can only change continuously: the first derivation of the route must therefore also be constant

The first roller coasters fulfilled only these necessary conditions. One consequence was that passengers often had problems in their spine after a trip. The reason for this was the discontinuity in the second derivation of the route, which from a physical point of view corresponds to the acceleration and thus to the forces acting on the passengers. Sudden changes in these forces caused the injuries. The second derivative should also be continuous.

In addition, too abrupt changes in direction, such as a 90° turn with an extremely small radius, would inevitably lead to serious injuries because the forces acting on the passengers become too great. So we have to make sure that the second derivative is not only continuous, but that its absolute value is also bounded. However, we can only verify this during the simulation – we cannot enforce it directly in the model.¨

In this practical course, a sequence of cubic polynomials is used for the routing. These are twice continuously differentiable everywhere and thus avoid the problems described above. They are therefore also used in real roller coaster construction. We'll be a special

Insert type of these polynomials: cubic B´ezier curves. These curves are named after the French engineer¨ Pierre Etienne B´ezie´ r, who used the curves in automobile construction. But they were developed in 1959 by Paul de Casteljau. These curves are easy to implement and have other advantages, as we shall see.

**4.1 Basics**

Mathematically, the route of any three-dimensional roller coaster is given as a set of functions that describe the development of the x,y, and z coordinates of the route as a function of a scalar parameter, here called s (remember the space curves from the Analysis):



A closed path must also fulfill the following condition:

[

Figure 1: Representation of a parameterized curve in space. The parameter s runs through the interval a..b and is mapped into three-dimensional space.

Such a closed space curve is shown in Figure 1 on the previous page. Since general functions x,y and z, which describe a path with loopings and curves, are difficult to find, an approximation of these original functions by simpler functions makes sense. In this practical course, the function between n nodes in three-dimensional space (x1,...,xn) is approximated by n cubic B´ezier curves, where one

support point is given by¨



parameter intervals

Figure 2: Representation of a parameterized, piecewise curve with nodes in space.¨ The parameter s runs through the interval 0..send1 and is mapped into the three-dimensional space between two nodes. At the support point, the function changes to and the parameter s runs through the interval send1..send2, etc.

**4.2 Cubic Bezier curves**

A d-dimensional cubic B'ezier curve has the following form:



where p0 to p3 are points in d-dimensional space. The scalar value s is the parameter. More information about this type of curves can be found e.g. in [6].

A closer look reveals that the space curve for s = 0 lies exactly at p0 and for s = 1 exactly at p3. In between we have a continuous function. If we now imagine p0 and p3 as two consecutive vertices of the coaster, xi and xi+1, we have an easy-to-use description for the coaster spline: we use for each of the sections the in Figure 2 are indicated on the previous page, such a B'ezier curve, denoted by Bi(s) in the following.

For the complete route description through this sequence of B'ezier curves, the ¨ is now missing

Calculation of the intermediate support points"¨ p1 and p2 for all sections (these are often¨ con" ”

called troll points”) – because only the n support points¨ xi are given (see formula 2 on the previous page). How these control points can be calculated from the support points is described in the next section

**4.3 Calculation of Control Points**

The control points p1 and p2 are calculated from the support points of the space curve (x1,...,xn) by solving a linear system of equations (LGS). These are known from the basic lectures in mathematics, especially linear algebra. A brief reminder: such a system has the form in general

ax = rhs,

where A is a matrix, rhs is a vector of known values, and x is a vector of unknown values. In our case, the known values are the vertices¨xi, while the control points¨ are the unknown values.

Using the formula for cubic B´ezier curves (Formula 3) and the continuity requirements described in the text above, we can set up our LGS. But first we define, as required above, p0 := xi and p3 := xi+1. In addition, we name p1 := ai and p2 := bi, since we need different values for the control points for each section of the spline.¨

For the further procedure we still need the first two derivations of Equation 3:



This gives the values of the B´ezier curve function and its first two derivatives at the ends of the interval s ∈ [0..1]:

Bi(0) = x I (6)

Bi(1) = xi+1 (7)

B0i(0) = −3xi + 3ai (8)

B (9)

B00i (0) = 6xi − 12ai + 6bi (10)

B. (11)

The continuity requirements described above can be directly translated into the following equations:¨

B0i(1) = B0i+1(0) (12)

B. (13)

Since this is a closed roller coaster curve, we have two more equations:

B (0) (14)

B. (15)

If one now inserts the corresponding equations 8 to 11 into the four conditions 12 to 15 (and makes the necessary index shifts), a uniquely solvable system of 2¨n·d linear equations for the 2¨n·d unknowns ai results and bi This is brought into a matrix-vector form and solved using standard methods such as the Gauss algorithm.

**4.4 Entire Algorithm**

In summary, the following algorithm results:

• Read in the xi,yi,zi coordinates of the nodes of the orbit.¨

• Prepare the matrix A of the LGS. The equations resulting from conditions 12 to 15 are encoded in this”.

”

• Determine from the coordinates xi the control points ai and bi for the different sections of the route by solving the LGS.¨

After that, one has a set of polynomials for each trajectory coordinate, which describe the course of the trajectory in a certain interval. These polynomials are required both for 2D/3D visualization in the simulator or editor and for later physical simulation of the roller coaster itself.¨

The Gauss algorithm can be looked up in [7] if it is no longer familiar from basic mathematics lectures. However, it is strongly recommended to use a fully implemented solution method for solving a system of linear equations.

**5 Modeling of dynamic behavior**

The physical model describes the roller coaster car as a single mass point. Normally the state of a mass point in space can be completely described by six values. These are the position (in all three spatial axes) and the speed (in all three spatial axes). Since this is a dynamic behavior, the position and the speed are time-dependent, i.e. the time (usually denoted by t) appears as a parameter of the six values.

With our roller coaster, however, the number of degrees of freedom is naturally reduced from six to two due to the geometry specified by the spline: the current position of the roller coaster on the curve and the current speed of the roller coaster in the direction of the curve.

**5.1 Conservation of Energy**

To describe the temporal behavior of our roller coaster, the appropriate differential equation (DE) must be set up. For this we use a very simple law of conservation, already known from school physics: the law of conservation of energy of Newtonian mechanics:

E = T + V = const. (16)

It states that in this model the total energy of the system (E), which is the sum of kinetic energy (T) and potential energy (V), remains constant. This fits our model requirements; we want to take these energies into account and, as mentioned above, neglect all other forces (friction, etc.). So let's write down the formulas for potential and kinetic energy:

 (17)

Here m denotes the mass, ˙p the instantaneous velocity of this mass, p the instantaneous position of the mass and g the effective force field, i.e. the gravitational force in our case. The small dot above p is the notation for a derivation with respect to time, which has been used since Isaac Newton: the instantaneous velocity ˙p is the derivation of the instantaneous position p with respect to time. After a little thought and a few attempts to remember what you were doing in physics class, you will quickly be able to verify this statement.¨

In order to get from this abstract Equation 17 to a differential equation that we can use, we have to think about what is still unclear in the equation. These are the velocity ˙p and the position p, which we add to our geo- ¨ introduced in Section 4

"must translate metry": we must incorporate what is momentary "" in our geometry

Position" and current speed" mean specifically. We'll start this off with a simple ” Insertion.

The course of the trajectory in three-dimensional space was described in Section 4.1 by a vector-valued function ~x parameterized with s. The roller coaster train is always on exactly this curve, so its position is described by a certain value of s for every point in time t. In other words: to describe the current position of the train, s becomes time-dependent. We mark this with the usual notation ¨ s = s(t). However, with this we have exactly fulfilled the requirement for the current position p in our physical model (equation 17) and write:¨

p = ~x(s(t)) =: ~x. (18)

Here ~x is a shorthand notation for the position vector ~x(s(t)) to make the following equations clearer. We also omit the notation of the time dependence of s if necessary. However, these two simplifications must not be forgotten, since they are very important for the further calculations!

In order to make the notation correct, we now also have to replace g with a vector ~g, which is defined as follows:



(In other words: gravity acts exclusively in the z-direction). Substituting Equations 18 and 19 into Equation 17 we get the following:



(Caution: note the chain rule – let's remember our abbreviation:¨ ~x = ~x(s(t))!)

 (20)

Here ~xs denotes the derivative of ~x with respect to the variable s and ~xT denotes the transposition of the vector ~x. The transposition is notationally necessary because we are now partly working with dot products of vectors.

Equation 20 can easily be rearranged and you get a simple differential equation for the instantaneous speed of the mass point (roller coaster train) on the trajectory:

.  (21)

We could actually use this for our further procedure, but unfortunately it has two possible forms: a positive and a negative one. And it is not clear to decide which of the two solutions we need. So we have to change something in the differential equation to make it uniquely solvable.

By looking at Equation 20, we see that our problem is represented by the square

”

the current speed is caused. If we derive this equation in time, this problem will disappear. So:



(Observe chain rule and product rule)



(observe chain rule)

. 

We rearrange this equation again, this time according to ¨s:

 . (22)

The energy is a constant by the law of conservation 16. The derivative of a constant is known to be 0, which gives us:

. 

By shortening ˙¨ s and the mass11 m we get the final form of our ODE:¨

 . (23)

In contrast to our first ODE (equation 21), this has a unique, albeit slightly more complicated form. It allows us to calculate the instantaneous acceleration (¨ s) to calculate.

**5.2 Possible model extensions¨**

The model described above can easily be expanded to include so-called source and sink terms, which can then model additional forces such as drive and friction.¨

The central observation that allows such an extension is that in this case the total energy of the system is no longer a constant. So if you take equation 22 on the previous page and replace E˙ with a function that describes the desired change in total energy, you have already extended the model.

For example, the equation¨

E˙ = −c |~x˙|2 (24)

model something like air friction: as is well known, this increases with the square of the velocity. The drag coefficient12 of the roller coaster and the density of the air would then flow into the positive constant c. Further extensions can be made in the same way - these then appear in Equation 24 as additional summands.¨

**5.3 Geometry derivations**

In order to be able to calculate equation 23, we still need the first and second derivation of the geometry with respect to the parameter s. But these are exactly the equations that we already needed to set up the system of equations in section 4.3 on page 5 (equations 4 and5).

**5.4 Zeitintegration**

Die DGL 23 gibt an, wie man aus gegebenen Werten (rechte Seite der Gleichung) die Beschleunigung ¨*s* berechnen kann.

Was wir jetzt brauchen ist ein Verfahren um aus dieser momentanen Beschleunigung die Position des Zuges auf der Achterbahnkurve zu einem bestimmten Zeitpunkt zu berechnen – denn das ist ja genau das, was wir für die Darstellung des Zuges ben¨ otigen. Solche Verfahren gibt¨ es naturlich schon; es sind Integrationsverfahren f¨ ur gew¨ ohnliche Differentialgleichungen. Sie ap-¨ proximieren numerisch mit Hilfe eines Startwertes für¨ *s*(*ti*) und einer Differentialgleichung ˙*s* den Funktionswert zum Zeitpunkt *ti* + *δt*. Durch wiederholtes Anwenden des Verfahrens kann man so eine Naherungsl¨ osung f¨ ur¨ *s*(*t*) für jedes Vielfache¨ *t* von *δt*, ausgehend von einem Startwert, berechnen.

11Die Masse des Zuges kurzt sich tats¨ achlich aus der DGL raus. Man denke an das Experiment mit Blei und¨ Feder im Vakuum – da keine Luftreibung vorhanden ist fallen beide gleich schnell.

12Hierbei handelt es sich um den beruhmten¨ *cw*-Wert.

Da unsere DGL allerdings eine zweifache Zeitableitung beschreibt – mathematisch benannt: sie ist eine DGL zweiter Ordnung“ – benotigen wir einen kleinen Trick um ein solches Verfahren¨

”

anwenden zu konnen. Wir schreiben unsere DGL zweiter Ordnung in ein System von zwei DGLs¨ erster Ordnung um:



Ultimately, this means nothing more than applying the integration procedure twice to the original ODE for s.

**5.5 Initial values for the time integration¨**

For the time integration of the ODE system¨ 25 we still need initial values¨ s(0) = s0 and ˙s(0) = s˙0. Let's recall the meaning of s: s(t) is the position of the coaster train at time t and ˙s(t) is the speed - but in parameter space, not in 3D space.

We can also best specify the start position in the parameter space, since an absolute ¨

position in 3D space only causes problems: it is not immediately clear whether a position is "legal" at all, i.e. whether it is also located on a spline piece. So we can choose as a starting position e.g.:¨

s0 = 0. (26)

This then means that we want to start at the beginning of the first spline piece. Other values are also possible, e.g. s0 = 2.5 would mean that we want to start halfway through the third spline segment

The situation is different for the initial velocity ˙s0. This cannot be set intuitively in the parameter space, but very well in the 3D space. This can best be illustrated by the units that occur: in 3D space, the unit is e.g. meters per second, whereas “in the parameter space the unit would be spline pieces per second and thus independent of the concrete dimensions of the orbit itself. For this reason we want as the initial velocity |x˙ 0| in the direction of the trajectory.

However, the differential equation still needs ˙s0 as a starting value, which we have to calculate:¨

|x˙ 0| = |xs,0 s˙0|

= |xs,0||s˙0|

⇒ |s˙0| = |x˙ 0|/|xs,0|. (27)

With these initial values (|x˙ 0| and s0) we can start the time integration.¨

**6 Runge-Kutta method for time integration**

As mentioned above, a numerical method is required to solve the differential equations. The idea of these methods can be understood most easily using the so-called ¨ Euler method, which is to be described here as an example. However, it is not used in the internship itself; a Runge-Kutta process is used there (more on this later)

**6.1 Euler's Polygonal Method**

The task when solving a differential equation is to find an unknown function x(t) of which only a few properties (usually the first derivative, ˙x(t)) are known. This is rarely analytically possible, so numerical methods are used. The simplest method is to approximate the unknown function with a polyline. At the starting point, the function value x(0) = x0 (the so-called initial value) and, of course, the slope of the function are known – this results from the differential equation. Now you walk a bit along the straight line that has the same slope as the function. This point forms the next initial value.

Given are: x˙(t) = f(t,x), x(0) = x0.

Then an approximation of the desired function¨ x(t) can be determined incrementally using the following algorithm:

x(δt (i + 1)) = x(δt i) + δtf(i δt,x(i δt)).

The parameter δt is commonly referred to as the time step. Figure 3 shows what a solution of the differential equation f˙ = f looks like using this method. Figure 4 on the next page shows the solution with a smaller time step δt. You can clearly see the improvement. It can even be shown that the solution of this algorithm for δt → 0 converges to the real solution of the differential equation. Due to its simplicity, however, this algorithm has disadvantages in terms of stability and accuracy. Therefore, a different algorithm will be used for the practical course, which in principle works the same way.

0

0.5

1

1.5

2

2.5

3

3.5

4

0

10

20

30

40

50

60

Loesung mit Euler,

δ

 t = 1.00

Analytische Loesung: f(t)=e

t

Figure 3: Euler's polyline method for the differential equation f˙ = f with the initial value f0 = 1 and δt = 1.0. The analytical solution is f(t) = et.

**6.2 The fourth-order Runge-Kutta method**

The fourth-order Runge-Kutta method (see e.g. [7]) first calculates some intermediate values, from which the final value for the next time step is then determined. This increases the stability and accuracy. This is paid for with an increased computational effort”

wall, since function values of the differential equation now have to be calculated four times per time step instead of ¨

0

0.5

1

1.5

2

2.5

3

3.5

4

0

10

20

30

40

50

60

Loesung mit Euler,

δ

 t = 0.10

Analytische Loesung: f(t)=e

t

Figure 4: Euler's polyline method for the differential equation f˙ = f with the initial value f0 = 1 and δt = 0.1. The analytical solution is f(t) = et.

only once as in the Euler method. Figure 5 on the next page shows the solution of the ODE f˙ = f using the 4th order Runge-Kutta method and the same time step size as in Figure 4. The numerical solution can no longer be distinguished from the analytical one in this example.

The fourth-order Runge-Kutta method is given by the following set of simple equations:

k1 = f(ti,xi)

k2 = f(ti + (1/2)δt,xi + (1/2)δtk1)

k3 = f(ti + (1/2)δt,xi + (1/2)δtk2)

k4 = f(ti + δt,xi + δtk3)

xi+1 = xi + (δt/6)(k1 + 2k2 + 2k3 + k4).

Here f is the differential equation to be integrated and i was introduced as an index for time.¨

Note: if this method also shows stability problems, the time step δt was chosen too large and must be reduced.¨

**6.3 Further Procedures¨**

There are also methods that automatically increase and decrease the step size during integration depending on an estimated integration error - these are so-called adaptive methods, which we do not want to discuss further here. You are of course free to use such a method anyway; Keywords here are Runge-Kutta-Fhlberg

method” [2] and Dormand-Prince method” [1] – both are so-called embedded Runge

Kutta procedure".

Since the time step width is chosen adaptively with these methods, one can imagine that this leads to problems with the visualization: ideally, this requires just as many time steps per second as it displays frames per second. To achieve this one can

Use interpolation method, which between the calculated time steps in certain

0

0.5

1

1.5

2

2.5

3

3.5

4

0

10

20

30

40

50

60

Loesung mit Runge−Kutta Ordnung 4,

δ

 t = 0.10

Analytische Loesung: f(t)=e

t

Figure 5: The fourth-order Runge-Kutta method for the differential equation f˙ = f with the initial value f0 = 1 and δt = 0.1. The analytical solution is f(t) = et. In this example, there is apparently no difference between the numerical and the analytic solution.

way interpolated and thus makes it possible to make any time steps. See for example [3].

**7 The simulation algorithm**

With the geometry from Section 4, the physics from Section 5 and the solution method from Section¨

6, all the ingredients for the roller coaster on the computer are now complete. After the route”

If the description was read from the XML file, the control points ai and bi for all route sections must be determined and saved in a suitable data structure.

Only one variable s(t) should be used to describe the position of the train on the track. The route section and the position on this route section must therefore be determined from this variable. Since each segment has exactly one parameter range of s ∈ [0..1], the segment results as the integer part of s(t) modulo the number of segments, n. Consequently, the parameter within the segment is exactly the floating point part of s (t). The solution of the differential equations starts at a certain initial position s(0) with an initial velocity ˙s(0). From there, a Runge-Kutta method is then used to integrate the differential equation 23 on page 9. The parameters of this differential equation then change in the course of the simulation as soon as s(t) changes its integer value - i.e. as soon as the section changes.¨

A relatively small time step may have to be chosen for the time integration. It is therefore advisable to design the program from the outset in such a way that the visualization is decoupled from the time integration. Possibilities here are, for example, a determinable divider between the integration and visualization steps or the interpolation method for Runge-Kutta methods already mentioned in section 6.3 on the previous page.¨

**7.1 Example Implementation**

Since the descriptions in this document are, despite everything, quite complex and therefore naturally difficult to understand, the participants have a fully-fledged, very well commented and structured prototype of the roller coaster simulation at their disposal. It is written in MATLAB, a programming language specifically for numerical mathematics. However, this language is quite easy to read, so the prototype can be used very well as a template for your own implementation.

The prototype contains all the important procedures described in this document:¨

• The geometry

• Calculation of the B´ezier control points for all sections from the support points

• The differential equation

• The time integration method

• A (very simple) visualization

• Several routes for testing (but not in XML format)

So download this prototype and give it a try. MATLAB is available on the university's terminal computers, but can also be purchased for a small fee from the computer center and installed on your own computer.

**8 XML files**

While the support of ASCII file formats used to cost the programmer quite a bit of time, today XML files can be used with off-the-shelf XML parsers. The advantage of human readability and easy editing of ASCII files is retained. We want to use these advantages for ourselves in the SEP and serialize the route descriptions and the simulation parameters in an XML format.

The XML files for the route description should look something like this:¨

<?xml version="1.0"?>

<bahn>

<stuetzpunkt>

<nummer> ... <\nummer>

<xcoord> ... <\xcoord>

<ycoord> ... <\ycoord>

<zcoord> ... <\zcoord>

<\stuetzpunkt> ...

<\bahn>

However, this XML format is not yet complete - due to requirements for the visualization (see [4]), further data are needed here (keyword: yaw vector). So here rules¨

Need for discussion and clarification between the groups.¨

There is also a file containing the parameters for the simulation:¨

<?xml version="1.0"?>

<simparameter>

<anfangsbedingungen>

<s\_null> ... <\s\_null>

<xdot\_null> ... <\xdot\_null>

<\anfangsbedingungen>

<physik>

<grav> ... <\grav>

<\physik>

<\simparameter>

The exact specification of the file format must be done by all groups in cooperation, since it represents the interface between the projects. Formal descriptions using XML schemas lend themselves to this, since these can be checked automatically and can therefore also be tested.

**9 General information**

In this internship, the main focus is clearly on software engineering. This document makes heavy use of mathematics; the reason for this, however, is to make the necessary basics accessible to you in the most digestible form possible. In the internship itself, good inter- and intra-team work, a well thought-out software design, the structured implementation of the algorithms and, last but not least, the quality of the documentation are more important.

It is recommended that groups take care of dividing the program into modules well in advance. Irrespective of the complexity of the task, self-designed extensions are possible and even desirable after consultation with the HiWi or the supervising assistant, as long as they do not restrict the required functionality. If anything is unclear¨ the HiWi should always be asked first if possible!¨

Dates are announced either by email or on the associated institute website. Further information and dates are given on the website of the Institute for Software¨ Systems Engineering.

At the end we wish all participants a lot of fun with the internship and look forward to many rapid roller coaster rides at the end of the semester.

**Literature**

1. Dormand, J. R. und P. J. Prince: *A family of embedded Runge-Kutta formulae*. Journal of Computational and Applied Mathematics, 6(1):19–26, Marz 1980.¨
2. Fehlberg, E.: *Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf W¨armeleitungsprobleme*. Computing, 6(1):61–71, Marz 1970.¨
3. Gladwell, I., L. F. Shampine, L. S. Baca und R. W. Brankin: *Practical Aspects of*

*Interpolation in Runge-Kutta Codes*. SIAM Journal on Scientific and Statistical Computing, 8(3):322–341, 1987.

1. Kayser-Herold, Oliver, Andreas Keese, Markus Krosche, Martin Krosche und

Oliver Pajonk: *Teilprojekt Achterbahnsimulator -Aufgabenbeschreibung für das Softwareent-¨ wicklungspraktikum am Institut für Wissenschaftliches Rechnen¨* , Marz 2009.¨

1. Krosche, Martin und Oliver Pajonk: *Teilprojekt Achterbahneditor - Aufgabenbeschreibung für das Softwareentwicklungspraktikum am Institut f¨ ur Wissenschaftliches Rechnen¨* , Marz 2009.¨
2. Prautzsch, Hartmut, Wolfgang Boehm und Marco Paluszny: *B´ezier and B-spline techniques*. Springer Verlag, 2002.
3. Schwarz, Hans Rudolf: *Numerische Mathematik*. B.G. Teubner, 1997.