Physically based motion of point masses on NURBS surfaces in real-time

Hendrik Bugdoll
Physically based motion of point masses on NURBS surfaces in real-time

Hendrik Bugdoll
Physically based motion of point masses on NURBS surfaces in real-time

Hendrik Bugdoll

This work shows the possibility of simulating and visualizing the motion of a point mass respectively a system of point masses on NURBS surfaces under the effect of gravity in real-time. After an introduction to the Lagrange formalism the complexity of the resulting equations of motion is revealed on some examples of parametric surfaces. Then the equations of motion are derived for a generic parametric surface and after that efficiently applied on NUBRS surfaces. The theoretical aspects are practically realized by a simulation program, which is briefly introduced. Benchmark results of test runs are presented. The approach is usable for scientific visualization purposes as well as for teaching purposes.

1 Introduction

Mostly, one has a certain intuition whether the motion of an object seems to be realistic or not. Even a relation between complex abstract formulas and the motion (also reverse) is hardly noticed. Yet, the power of imagination of many persons is insufficient for motions under idealized conditions like frictionlessness, because such phenomena can only marginally be observed in our natural environment. Here, the interdisciplinary scientific visualization combines aspects of computer science, physics and mathematics. Scientific visualization fulfills the expectations of making a difficult issue more clear in this context.

There is also a trend in teaching to communicate more and more content in a multimedia way. Not much is won by scaring off students in theoretical physics with too complicated formulas. Therefore, the personal motivation of the students should be increased by descriptive pictures or series of pictures like animations.

The goal of this work is the visualization of the constraint (or bounded) motion of a point mass respectively a system of point masses under the effect of gravity on static surfaces. Therefore, NURBS (Non-uniform rational B-splines) surfaces are supposed to maximize the user flexibility in the way of surface modelling at runtime. Thus we want to realize a real-time 3D animation based on a physically correct computer simulation.
2 Related Work

The formalism established in 1788 by J. L. Lagrange is contained in many textbooks about theoretical physics. Sometimes the equations of motion are derived for simple surfaces, e.g. for a sphere by way of illustration. Kuska showed in [Kus97] how it is feasible to symbolically compute the equations of motion for surfaces of low degree and visualize pictures of the resulting trajectories with the aid of the computer algebra system Mathematica. The simulation and visualization of the motion on an elliptical paraboloid was shown in [ES04]. A few websites contain Java applets visualizing the motion on simple surfaces with rudimentary graphics. In [Bug09] a real-time capable animation of a single point mass on selected cubic free-form surfaces was presented. This was only done for cubic Bézier surfaces and special cases of them. Due to the fact that the amount of control points and thus the adaptability to a desired geometry depends on the degree of the surface, these surface classes are less flexible and not universally usable. However, methods for symbolic computing at runtime provided by [BFK02] were necessary thereto.

In contrast to general particle simulations (e.g., see [Dro07]) we concentrate on physically based motions of point masses on a specified surface.

3 Theoretical Background

The motion of an object, in physics heavily idealized as a point mass or a system of point masses, can generally described by mathematical equations called equations of motion (or movement equations). They form a functional association between the acting forces and the object’s position respectively their first and second time derivatives. In our context the acting force is the gravity \( g \) in negative \( z \)-direction. So the gravity expectedly points downwards.

In the last centuries several fundamental approaches were developed for this purpose; emphasized are the Newtonian and the Lagrangian mechanics. For recent related publications see [Bli07] and [Bli10]. They are based on the Whittaker’s method. We will not cover them in this work, but rather base our investigations on the classical Lagrangian approach.

3.1 Lagrangian mechanics

It is possible to model a motion tied on constraints with the Lagrangian mechanics. We suppose the bounded motion on a surface as a constraint, which is modelled by a constraint force \( \mathbf{Z} \in \mathbb{R}^3 \) performed by the surface. This is directly done in the Lagrange’s equations of the first kind for
an implicit surface \( f(x, y, z) = 0 \) by
\[
m\ddot{x} = F + Z = F + \lambda \nabla f,
\]
where \( m > 0 \) denotes the constant mass of the point mass, \( F \) the force exerted by the gravity and \( \nabla f \) the gradient of function \( f \). The Lagrangian multiplier \( \lambda \) has to be determined.

Lagrange’s equations of the second kind
\[
\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = 0, \quad L = E_{\text{kin}} - E_{\text{pot}}, \tag{1}
\]
are not based on an implicit surface equation. They require a parametric form of the surface equation and describe the bounded motion with \emph{generalized coordinates}. Two of such real coordinates \( u \equiv q_1, v \equiv q_2 \) allow for specifying the motion on a parametric surface \( C(u, v) \subset \mathbb{R}^3 \). These coordinates are time-dependent and so \( \dot{q}_i = \frac{dq_i(t)}{dt}, \quad \ddot{q}_i = \frac{d^2q_i(t)}{dt^2} \). Take note that the point mass leaves the surface at no time. This is why we can interpret the point mass’ position \( x = (x, y, z) \in \mathbb{R}^3 \) as a point of the surface in the following sections.

The \emph{Lagrangian} \( L \) is the difference between kinetic energy \( E_{\text{kin}} = \frac{m}{2} \| \dot{x} \|^2 \) and potential energy \( E_{\text{pot}} = mgz \), where \( g \) is the gravity. The sum of kinetic energy and potential energy is a conserved quantity in this system.

The main advantage of Lagrange’s equations of the second kind is the motion’s description by only two instead of three equations of motion. They are even less complex for many surfaces. In the case of surfaces of revolution (generated by rotating a plane curve about the \( z \)-axis) one equation of motion and one constant expression are sufficient, because the conservation of angular momentum implies \emph{cyclic coordinates} with \( \frac{\partial L}{\partial \dot{q}_{\text{cyclic}}} = 0 \). For further information on theoretical aspects see [Kus97] and [ESHD05].

Furthermore, this approach guarantees that the following two essential conditions are fulfilled during the entire simulation: the position of the point mass is a point \( x \in C(u, v) \) and the velocity vector \( \dot{x} = \frac{dx}{dt} \) is a vector in the tangent space of the surface in point \( x \).

### 3.2 Some examples of parametric surfaces

To illustrate the complexity of the occurring equations we exemplary investigate the following two parametric surfaces. The parametric equations for a hyperbolic paraboloid is given by \( x = (u, v, uv) \). Applying (1) to this simple example yields the equations of motion \( \ddot{u} = v\zeta, \quad \ddot{v} = u\zeta \) with \( \zeta := -\frac{q+2q}{(1+u^2+v^2)} \). In most cases the complexity of the equations of motion increases with the complexity of the parametric equations. Exceptional cases are for instance symmetries in the surface equations, which may reduce certain terms to zero.
A vastly more complicated example is given by the surface we called wave-torus. Its parametric equations
\[
x = (R + \psi \cos v)(\cos u, \sin u, \psi \sin v)
\]
with an oscillating tube radius \( \psi := r + a \sin(2bu) \) and thus four constant parameters \( R, r, a, b \in \mathbb{R} \) yield two complex and barely comprehensible equations of motion
\[
\ddot{u} = \frac{a b \cos(2bu)(4 \cos v(R + \psi \cos v) + 2a(1 - 2b^2 \cos(2bu)) \sin(2bu))\dot{u} \dot{v} + 8a \psi(2R + \psi \cos v) \sin v \dot{u} + 4ab \cos(2bu)(-g \sin v + \psi^2)}{a^2 + 2a^2b^2 + a^2b^2 + 4R^2 + a^2(2b^2 - 1) \cos(2bu) + 4ar \sin(2bu) + 8R \psi \cos v + 2 \psi \cos(2bu)}
\]
\[
\ddot{v} = -\frac{g \cos v + (2R + \psi \cos v) \sin v \dot{u}^2 + 2ab \cos(2bu) \dot{u}}{\psi}
\]
The derivation can be done through nasty work by hand or with the aid of computer algebra systems, but it is very tedious and must be done for every new surface. At this point it should be apparent that the equations of motion are independent of mass \( m \), which is canceled during computation. Figure 1 shows a hyperbolic paraboloid and a wave-torus plus typical motions on these surfaces.

4 Equations of motion for a generic parametric surface

Considering much more complex geometries (extending those of section 3.2), leads to more and more unhandy parametric equations and equations of motion. Here it would seem appropriate to use reliable spline techniques, which provide a good local control during the modelling stage even with low degree of the surface.

We now initially derive the equations of motion for a generic parametric surface. Starting from the simplest form, we write the parameter notation with unknown functions \( x(u, v), y(u, v), z(u, v) \)
H. Bugdoll:

for the three Cartesian coordinates as

\[ \mathbf{x} = \mathbf{x}(u, v). \]  

(2)

Thereby the first time derivatives are \( \dot{\mathbf{x}} = \dot{u} \frac{\partial \mathbf{x}}{\partial u} + \dot{v} \frac{\partial \mathbf{x}}{\partial v} \) and the Lagrangian \( L \) used for the computation of (1) is consequently

\[ L = \frac{m}{2}(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) - mgz. \]

After simplification, the first Lagrange’s equation of the second kind concerning generic coordinate \( u \) is given by:

\[
\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{u}} \right) - \frac{\partial L}{\partial u} = \ddot{u} \left( \frac{\partial^2 \mathbf{x}}{\partial u^2} + \frac{\partial^2 \mathbf{x}}{\partial v^2} + \frac{\partial^2 \mathbf{x}}{\partial u \partial v} \right) + \dot{u}^2 \left( \frac{\partial^2 \mathbf{x}}{\partial u^2} + \frac{\partial^2 \mathbf{x}}{\partial u \partial v} \right) + 2\dot{u}\dot{v} \left( \frac{\partial^2 \mathbf{x}}{\partial u \partial v} \right) - g \frac{\partial z}{\partial u}
\]

\[ = \ddot{u}_{1} \gamma_{1} = 0. \]

In analogous manner, it yields with appropriate \( \alpha_{2}, \beta_{2}, \gamma_{2} \) for the second equation concerning generic coordinate \( v \) (after observing that \( \alpha_{2} \equiv \beta_{1} \)) to \( \ddot{u}_{2} + \ddot{v}_{2} + \gamma_{2} = \ddot{u}_{1} + \ddot{v}_{2} + \gamma_{2} = 0 \). These coupled equations can be rewritten to a simple system of linear equations which allows us to get the desired (explicit) equations of motion in the form \( \ddot{u} = \ddot{u}(u, v, \dot{u}, \dot{v}, g) \) and \( \ddot{v} = \ddot{v}(u, v, \dot{u}, \dot{v}, g) \):

\[
\ddot{u} = \frac{\beta_{2} \gamma_{1} - \beta_{1} \gamma_{2}}{\beta_{1} - \alpha_{1} \beta_{2}}, \quad \ddot{v} = \frac{\alpha_{1} \gamma_{2} - \beta_{1} \gamma_{1}}{\beta_{1} - \alpha_{1} \beta_{2}}.
\]

(3)

where \( \alpha_{1}, \beta_{1}, \beta_{2}, \gamma_{1}, \gamma_{2} \) are independent of \( \ddot{u} \) and \( \ddot{v} \) as defined before. Hence, for computing the generic solution (3) fifteen partial derivatives are necessary. These are the three-component vectors \( \frac{\partial \mathbf{x}}{\partial u}, \frac{\partial \mathbf{x}}{\partial v}, \frac{\partial^2 \mathbf{x}}{\partial u^2}, \frac{\partial^2 \mathbf{x}}{\partial v^2}, \frac{\partial^2 \mathbf{x}}{\partial u \partial v} \). Thereby the tangent vectors \( \frac{\partial \mathbf{x}}{\partial u}, \frac{\partial \mathbf{x}}{\partial v} \) of the surface in point \( \mathbf{x} \) are easily given.

Notice that the symmetric form of (3) is caused by the fact that the generic parametric form itself (2) is symmetric in \( u \) and \( v \).

5 Application on NURBS surfaces

Free-form surfaces in general and NURBS surfaces in particular are a popular and convenient surface modelling technique in computer graphics and computer-aided (geometric) design.

The solution presented in the previous section is applicable for any parametric surface for which the required partial derivatives are computable. Now we use this approach for NURBS surfaces, as is shown in the following passage.
A NURBS surface in vector notation is modelled by

$$\mathbf{x} = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}(u,v) \mathbf{P}_{i,j}$$

with the (piecewise) rational B-spline basis functions

$$R_{i,j}(u,v) = \frac{N_{i,p}(u)N_{j,q}(v)w_{i,j}}{\sum_{k=0}^{p} \sum_{l=0}^{q} N_{k,p}(u)N_{l,q}(v)w_{k,l}}$$

as blending functions, control points $\mathbf{P}_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j}) \in \mathbb{R}^3$ and additional weights $w_{i,j}$. The (piecewise) B-spline basis functions $N_{i,p}(u)$ and $N_{j,q}(v)$ with degrees $p$ and $q$ are defined using the well-known Cox-de Boor recursion formula, see [PT97]. The time derivatives $\dot{N}_{i,p}(u)$ and $\ddot{N}_{i,p}(u)$ of $N_{i,p}(u)$ can also be written as a recursion; analogous for $N_{j,q}(v)$:

$$\dot{N}_{i,p}(u) = \frac{p}{\tau_{i+p} - \tau_i} N_{i,p-1}(u) \quad \dot{N}_{i,p}(u) = \frac{p(p-1)}{(\tau_{i+p} - \tau_i)(\tau_{i+p-1} - \tau_i)} N_{i,p-2}(u) - \left( \frac{p}{\tau_{i+p} - \tau_i} + \frac{p}{\tau_{i+p+1} - \tau_{i+1}} \right) \frac{p-1}{\tau_{i+p} - \tau_{i+1}} N_{i+1,p-2}(u) + \frac{p(p-1)}{(\tau_{i+p+1} - \tau_{i+1})(\tau_{i+p+1} - \tau_{i+2})} N_{i+2,p-2}(u).$$

Starting from (4) the first time derivative is

$$\dot{\mathbf{x}} = \sum_{i=0}^{n} \sum_{j=0}^{m} \left( \frac{\partial R_{i,j}(u,v)}{\partial u} \dot{v} + \frac{\partial R_{i,j}(u,v)}{\partial v} \dot{u} \right) \mathbf{P}_{i,j}$$

and the required partial derivatives are

$$\frac{\partial^{\eta+\mu} \mathbf{x}}{\partial u^{\eta} \partial v^{\mu}} = \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{\partial^{\eta+\mu} R_{i,j}(u,v)}{\partial u^{\eta} \partial v^{\mu}} \mathbf{P}_{i,j}$$

where $1 \leq \eta + \mu \leq 2$. To compute the partial derivatives of $R_{i,j}(u,v)$ the use of the quotient rule is needed, which makes the derivatives more complicated.

Let $\frac{A}{C} = A_{i,j}(u,v) := R_{i,j}(u,v)$, we exemplary take a closer look at $\eta = 1, \mu = 0$:

$$\frac{\partial \mathbf{x}}{\partial u} = \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{\partial R_{i,j}(u,v)}{\partial u} \mathbf{P}_{i,j} = \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{\partial A_{i,j}}{\partial u} \mathbf{P}_{i,j} \quad =: \mathbf{A}_\Sigma$$

$$= \left( \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{\partial A}{\partial u} \mathbf{P}_{i,j} \right) \mathbf{C} - \left( \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{\partial A}{\partial u} \mathbf{P}_{i,j} \right) \frac{\partial C}{\partial u} = \left( \frac{\partial A_{\Sigma}}{\partial u} - \mathbf{A}_\Sigma \frac{\partial C}{\partial u} \mathbf{C} \right) \frac{1}{\mathbf{C}}.$$

The last simplification is done for implementational reasons. As is easy to see, that the reciprocal and the partial derivative of $C$ have to be computed only once in the three-component vector. For $\eta + \mu = 2$ the quotient rule necessarily has to be applied twice.
Since the recursive computation of the B-spline basis functions needs a significant effort, an optimization is preferable. One can detect the following properties described in [PT97]: (i) not all $N_{i,p}(u)$ are needed because of the finite support, (ii) many $N_{i,p}(u)$ are computed multiple times (compare multiple incoming edges in recursion direction in Figure 2) and (iii) many $N_{i,p}(u)$ are zero as per definition of the recursion base case. It seems natural to avoid redundant evaluations of these functions.

Regarding (i), a linear or binary search in the knot vector $\tau = (\tau_1, \ldots, \tau_{n+p+1})$, $\tau_i \leq \tau_{i+1}$, yields $i_{max}$ with $u \in [\tau_{i_{max}}, \tau_{i_{max}+1})$ and $i_{min} = i_{max} - p$. Regarding (ii) and (iii), the recursion is dissolved and every $N_{i,p}(u)$ is only computed once per a specific $u$ in reversed direction; analogous for $N_{j,q}(v)$. The time derivatives $\dot{N}_{i,p}(u)$ and $\ddot{N}_{i,p}(u)$ can be obtained within the computation of the blending functions for $p - 1$ respectively $p - 2$.

With these considerations the following computation order is obvious:
1. compute $i_{\text{min}}, i_{\text{max}}$ and $j_{\text{min}}, j_{\text{max}}$.

2. compute the blending functions $N_{i_{\text{min}},p}(u), \ldots, N_{i_{\text{max}},p}(u)$ and $N_{j_{\text{min}},q}(v), \ldots, N_{j_{\text{max}},q}(v)$ plus their first and second time derivatives as shown in Figure 2 and save them temporarily,

3. compute efficiently $A \Sigma, \frac{\partial^{n+\mu} A \Sigma}{\partial u \partial v}, C, \frac{\partial^{n+\mu} C}{\partial u \partial v}$ in one double for-loop. Notice that many terms in the double sums can be reused in one loop iteration,

4. compute $\frac{\partial^{n+\mu} \mathbf{x}}{\partial u \partial v}$ and thereof $\alpha_1, \beta_1, \beta_2, \gamma_1, \gamma_2$, whereby all data for solving the equations of motion are present.

Therefore, $N_{i,p}(u)$ and $N_{j,q}(v)$ have to be continuously double differentiable. As a consequence, the minimal degree for our simulation purpose is $p_{\text{min}} = q_{\text{min}} = 3$, because $N_{i,0}(u)$ and $N_{j,0}(v)$ are not continuous.

The equations of motion for B-spline surfaces are easily obtained by using B-spline basis functions $N_{i,p}(u)N_{j,q}(v)$ instead of the rational B-spline basis functions $R_{i,j}(u,v)$ in (4). Equations of motion for Bézier surfaces are derived in analogous manner with Bernstein polynomials $B_{i,n}(u)B_{j,m}(v)$. In these cases the quotient rule is no longer required. Of course B-spline surfaces can also be obtained by setting all weights $w_i = 1$ and moreover Bézier surfaces with additional adjustment of the knot vectors.

6 Implementation

Now we present the practical realization by a simulation program.

6.1 Workflow of the simulation program

Figure 3 shows the four-stage workflow we have embedded in the simulation tool. For every new surface the following procedure is executed:

1. obtain the parametric surface equations by user choice of predefined surfaces or compute them after interactive manipulation of control points,

2. therefore the equations of motion are determined,

3. simulate the motion (one single step or one step with several intermediate steps),

4. visualize and goto 3 (until user abort).
The first and second step are hardcoded in the program, the expressions have only to be filled with numerical data. In this connection a computation of the tangent vectors for i.a. constructing tangent plane and normal vector is done. Step three and four are continuously passed through during the animation, so it yields an on-the-fly simulation.

Additional and in view of highly complex equations of motion and slower computers there is a strict separation of simulation and visualization implemented, too:

3. simulate the motion (all simulation steps),

4. visualize (all simulation steps).

The simulation stage is performed for a particular finite period as a preprocessing in that case. Hence, the real-time capability of the visualization no longer depends on the simulation speed. But there is also one mentionable disadvantage. With preprocessing we lose the opportunity to change simulation parameters like gravity, step size, etc. during the simulation. And there is no more chance for an interactive simulation – possible in the on-the-fly mode – where the user can manipulate the motion by changing the length and direction of the velocity vector via keyboard interaction combined with a chase camera mode.

Coulomb friction can be considered in a simple way. After every simulation step the point mass’ velocity is damped by a factor in relation to the coefficient of friction and to the step size.
6.2 Symbolic methods

The solution (3) also can be used for arbitrary parametric surface equations inputted by the user with symbolic methods at runtime. This was already described in [Bug09]. With the aid of the free C++ library GiNaC [BFK02], maximum flexibility in respect to the parametric surface equations was reached. GiNaC extends the programming language C++ to new data types for symbolic expressions and methods for symbolic computing (e.g., see [Vol06]). The disadvantage of a missing expression simplifier was fixed there through an optional linkage to Mathematica for utilizing its powerful simplifier at runtime. Here, this approach can be applied in a similar manner. Contrary to [Bug09] now the symbolic computation of the equations of motion could be omitted. After the partial derivatives are symbolically computed they only have to be inserted in (3). If a certain partial derivative is zero, GiNaC allows to simplify operations with zero, while a general expression simplifier is not available.

6.3 Scientific computing and visualization

The equations of motion form a system of ordinary differential equations (ODE) as in many other physical problems. They can not be solved exactly except for planes or surfaces which are not curved in the direction of gravity. We utilize a numerical solution procedure of the well-known and approved Runge-Kutta methods (see [PTVF07] and [ESHD05]). Therefore, a reduction from a second order system to a first order system is required. So the quantity of ODEs is duplicated. As an iterative method for approximating the solution of the equations of motion the common fourth-order Runge-Kutta method (RK4) with double precision arithmetic is used. In addition, there are also an adaptive version of the RK4 with optimized step size, the Euler’s method (RK1) and the improved Euler’s method (RK2) implemented.

Furthermore, certain initial conditions are needed for the (numerical) solution of the ODEs. The user is required to set the following two initial conditions for this initial value problem: the initial position $\mathbf{x}_0$ and the initial velocity $\dot{\mathbf{x}}_0$. This is done by intuitive mouse (or keyboard) interaction combined with an additional picture-in-picture camera, which looks in the direction of the normal on the current initial position and is oriented by the tangent vectors of the surface. The user changes $u_0, v_0, \dot{u}_0, \dot{v}_0$ at a starting time $t_0$, thereby $\mathbf{x}_0$ and $\dot{\mathbf{x}}_0$ are determined. So the two conditions at the end of section 3.1 are fulfilled, too.

The simulation program and its comfortable graphical user interface (GUI) is shown in Figure 4. The current problem is visualized by drawing (a section of) the surface as a parametric plot.
plus control points and control polygon during the modelling stage, infinitely small point masses as visible particles and finally their trajectories as linestrips or tubes via OpenGL. There are several plotting settings available for the surface and trajectories. For optimizing the rendering performance we use standard techniques like OpenGL display lists.

Additionally, a few technical details should be given: The portable stand-alone simulation program is usable for Windows and Linux platforms. It is developed in C/C++ for high performance and uses the fast and slim FLTK library for GUI creation. Common graphics hardware with standard OpenGL support is sufficient.

![Simulation program](image)

**Figure 4: Simulation program.**

### 7 Results

Because of the difficulty for describing complexity results in this field we use just frames per second (fps) and absolute times as measure for the speed of our implementation (of course depending on the hardware). Regarding the on-the-fly mode, Table 1 shows that the simulation performance is relatively independent of the degrees $p$ and $q$ (in case of useful degrees) and also independent of the specific structure of the NURBS surface. It is in evidence that the count of control points has an impact on the simulation speed. The reason for this fact is the rendering of the surface itself. Since the speed of the OpenGL pipeline is apparently concealing the pure computation speed in on-the-fly mode, the preprocessing times are more clear and show that the geometric structure of the NURBS surface has no impact on the preprocessing speed. The number of control points has an impact to a limited extent because of the optimizations from section 5. Only $(p + 1)(q + 1)$
12

control points are used in the computations. Indeed, the degrees have a big influence on the preprocessing speed.

<table>
<thead>
<tr>
<th>Surface</th>
<th>Preprocessing mode (time)</th>
<th>On-the-fly mode (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperbolic paraboloid</td>
<td>0.01 s</td>
<td>259</td>
</tr>
<tr>
<td>Wave-torus</td>
<td>0.03 s</td>
<td>185</td>
</tr>
<tr>
<td>NURBS surface (16 control points, ( p = q = 3 ))</td>
<td>0.32 s</td>
<td>272</td>
</tr>
<tr>
<td>NURBS surface (225 control points)</td>
<td>0.48 s</td>
<td>154</td>
</tr>
<tr>
<td>NURBS surface (low degree: ( p = q = 3 ))</td>
<td>0.34 s</td>
<td>208</td>
</tr>
<tr>
<td>NURBS surface (higher degree: ( p = q = 7 ))</td>
<td>1.01 s</td>
<td>206</td>
</tr>
<tr>
<td>NURBS surface (planar)</td>
<td>0.48 s</td>
<td>208</td>
</tr>
<tr>
<td>NURBS surface (complex structure)</td>
<td>0.48 s</td>
<td>205</td>
</tr>
</tbody>
</table>

Test environment: Intel Xeon 2.8 GHz, 4 GB RAM, NVIDIA GeForce 7600GS, no FSAA, 1920×1200 Pixel, Windows XP Prof.

Table 1: Simulation speed of the motion of a single point mass. If not otherwise specified, the default settings for the NURBS surfaces are 100 control points and \( p = q = 4 \). The step size is 0.005; RK4 is used. The preprocessing time is for computing 10 virtual seconds of the motion.

<table>
<thead>
<tr>
<th>Surface</th>
<th>Preprocessing mode (time)</th>
<th>On-the-fly mode (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count of point masses ( \rightarrow )</td>
<td>1 100 200 400</td>
<td>1 100 200 400</td>
</tr>
<tr>
<td>Hyperbolic paraboloid</td>
<td>0.01 s 4.16 s 8.64 s 19.98 s</td>
<td>259 97 53 28</td>
</tr>
<tr>
<td>Wave-torus</td>
<td>0.03 s 6.44 s 13.14 s 29.21 s</td>
<td>185 78 47 25</td>
</tr>
<tr>
<td>NURBS surface</td>
<td>0.44 s 46.96 s 94.28 s 202.81 s</td>
<td>207 31 17 9</td>
</tr>
</tbody>
</table>

Table 2: Simulation speed of a single point mass in comparison to systems of point masses. The test conditions are the same as in Table 1. The NURBS surface has 64 control points and \( p = q = 4 \).

As one can see in Table 2, the performance logically decreases by increasing the count of point masses. A doubling of the amount approximately halve the frame rate. The preprocessing time scales very well with the count of point masses. For comparison, the measurement results of the surfaces presented in section 3.2 are listed, too.

All measurements were generated with a small step size of 0.005 for the RK4 routine. This results in a very small error of \( \approx 10^{-9} \). The simpler Euler’s method is proved to be unsuitable with no conversation of energy. The improved Euler’s method is sufficient in some cases with small step sizes.

In the on-the-fly mode intermediate steps without graphical output noticeably decrease the numerical error. This is purchased by only a small decrease in the simulation speed. In addition, the emulation of friction does not show a loss of performance.

Figure 5 illustrates typical trajectories on a NURBS surface. The velocity coloring and the acceleration coloring of the trajectories give more information about the temporal progress of the
motion than in static pictures with standard coloring (compare Figure 5b with 5c and 5d).

Figure 5: Bounded motion of a point mass on a NURBS surface with 100 control points, which is the result of the modelling stage (a). The trajectories in (b) are computed in two simulation runs – without (light blue trajectory) and with (orange trajectory) emulation of friction. The first one is colored in respect to the velocity (c) respectively acceleration (d) of the point mass by color interpolation between green and red.

8 Conclusion and future work

By use of Lagrange’s equations of the second kind in combination with the approved modelling technique NURBS, a suitable and efficient approach and thereof a simulation tool were developed. Hence, the simulation and visualization of physically correct motions of point masses on NURBS surfaces are realizable in real-time and so usable in real-time applications. Therefore, the motion on arbitrary curved surfaces – exactly modelled by NURBS surfaces – or motions on complex large-scale surfaces (e.g., a landscape) are possible, although the degree of the surface is kept relatively low.

As a further extension of this work it is planned to investigate the impact of optimization techniques (like culling or level of detail) to handle such large-scale surfaces. Small systems of point masses without interaction between the particles are possible in real-time even with moderate
H. Bugdoll: CPUs. This is e.g. useful for direct comparison of several trajectories based on slightly different initial conditions. In particular a GPU-based parallel numerical solution using CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language) is in progress to cover the motion of large systems of point masses.

The achieved solution is usable for scientific visualization purposes as well as for teaching purposes. With this method, all advantages and benefits of NURBS can be used. So the examination of the point mass’ motion on a very large range of surfaces is achievable without recomputation of the equations of motion for another surface. Through emulation of friction more realistic conditions can be considered which are very crucial in game physics.

In addition, the computation of locally shortest paths between two points on the surface called geodesics is given almost without much extra effort. It can be realized by setting the gravity to zero. The initial value problem has to be transformed into a two-point boundary value problem, which implies an appropriate solver. The shooting method or relaxation could be an opportunity for this purpose [PTVF07]. A boundary problem as well is the computation of closed trajectories (start point is identical with end point). This is very interesting under physical aspects.

Moreover, improvements in the rendering quality of the surface and trajectories are beneficial. The GPU-based visualization or raytracing of NURBS surfaces is a current active research field (e.g., see [AGM06]). With this, one could follow the actual trend to let the GPU or multiple GPUs doing both graphics and physics calculations.

Acknowledgement

We would like to thank Christian Stussak for fruitful discussions.

References


Information about the Author(s)

Hendrik Bugdoll
Institut für Informatik
Universität Halle
06099 Halle, Germany
E-Mail: hendrik.bugdoll@informatik.uni-halle.de
WWW: http://www.informatik.uni-halle.de/arbeitsgruppen/computergrafik/staff/bugdoll/