Virtual Mechanics

Simulation and Animation of Rigid Body Systems

Hartmut Keller, Horst Stolz, Andreas Ziegler, Thomas Bräunl
Univ. Stuttgart, IPVR, Computer Vision Group,
Breitwiesenstr. 20-22, D-70565 Stuttgart, Germany
e-mail: braunl@informatik.uni-stuttgart.de

Abstract: The animation system *AERO* is presented in this paper. It allows the creation of virtual environments from scenes with simple three-dimensional geometrical objects (sphere, cylinder, box, point, plane). Objects can be linked to each other by a variety of methods (rod, spring, damper, joint). Realistic object movements are achieved by a simulation procedure, where forces can be applied to the objects in addition to gravity, friction, and air resistance. Collisions between objects are simulated using either the "penalty method" or the "analytical method", described in the text. Wire frame animations can be generated in real time, depending on scene complexity and computer system performance. *AERO* also generates scene description sequences, which can be used as input for ray tracing programs to generate photorealistic animations as batch jobs, a task that is also called "physically based modelling". *AERO* has a number of special features, such as stereo graphics output, camera control, and even the mounting of the camera to an object in the scene.

Keywords: simulation, animation, physically based modelling, rigid body systems, virtual mechanics, ray tracing.

1. Introduction

Computer animation has been becoming increasingly important in recent years. Of particular interest is animation that results from simulation. In the area of mechanics, reality is reduced to relatively simple abstractions, such as the velocity and position of a mass at a single point, in order to reduce the computational complexity. However, if one gives up this simple abstraction and considers real objects with actual dimensions, the problem becomes increasingly more complicated. In the new scenario there is rotational velocity and acceleration and multiple objects can collide at different angles. In addition, bodies touching each other no longer exert point forces, instead they exert forces over entire areas. These complex systems are no longer expressible in closed form, instead they must be simulated iteratively. For example, how do several balls behave when being dropped from different heights onto an uneven floor? When and how do the balls collide (see figure 1) ?

AERO ("Animation Editor for Realistic Object motion") is an animation system for the visualization of such problems. In *AERO* a virtual world is simulated, in which all defined bodies move according to the operative physical laws. This system allows the design and execution of physically based computer animations. While the motion of objects in other animation systems often has to be specified by the user, *AERO* at-

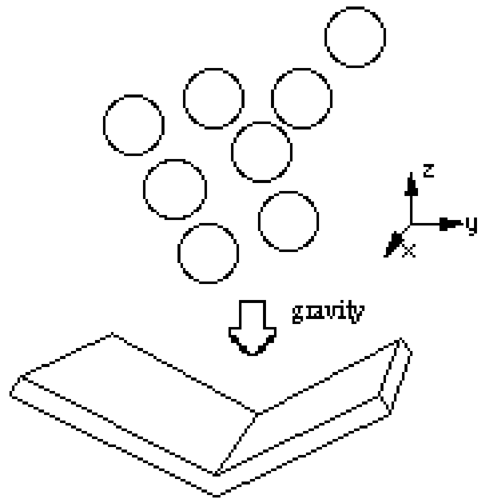

Figure 1: Example of a simulation

tempts to achieve realistic animation solely through the computational application of the laws of mechanics.

A scenario containing objects and applied forces is produced with the help of an editor for 3D-data, which was developed during the project. After starting the simulation, the movement and interaction of objects can be observed. Particularly interesting is the option to make changes in the scenario at any point during execution and either continue or restart the simulation. The laudable goal of generating photo realistic animation in real time is not achievable on current workstations. Therefore, an implementation was chosen which allows the presentation of the animation sequences at three levels:

1. **Interactive mode**: The scenes are produced, calculated and presented in an interactive user interface. The viewing of a simulation sequence can be stopped, wound forward and backwards and altered under user control just like with a video mixer. It is possible to react to the events of the simulation through this interactive intervention. For example, if an external force is to be applied at the exact moment of a collision, one must first know the exact moment of the collision. In *AERO* the sequence could be started and simulated until the collision occurred. At this animation frame, the sequence would be stopped and the new force applied.
2. **Precomputed mode**: It is possible with particularly complex scenes containing many objects and complicated interdependencies that the computation of the individual sequence takes a relatively long time. Therefore, it is possible to calculate a sequence off-line (in batch mode) and have it displayed at a later time.
3. **Rendering (full calculation)**: Due to time constraints, the previous two modes allow only a very simple graphical wire-frame representation. In order to achieve photo realistic results, enormous computational expense and therefore considerable time is required. In this mode, a sequence with full details (light, shadows, surfaces, etc.) can be calculated off-line. *AERO* generates source files for an external ray tracing program. The result, basically a sequence of rendered images, can be displayed as animation on the computer screen using special presentation programs (e.g. ImageMagic or MPEG compression) or stored on videodisc or videotape.

2. Creating a Virtual Environment

2.1 Scene Editor

After starting *AERO*, the user enters the scene editor (see figure 2). The editor allows to enter three-dimensional objects, forces and connections. However, since only two-dimensional media are available for input (such as keyboard and mouse) and output (screen), the input occurs through four views. These are (clockwise from bottom left): a view from above (XZ view), a view from the front (XY view), a view from the right (YZ view), and a 3D view diagonally from above to the front and the right.

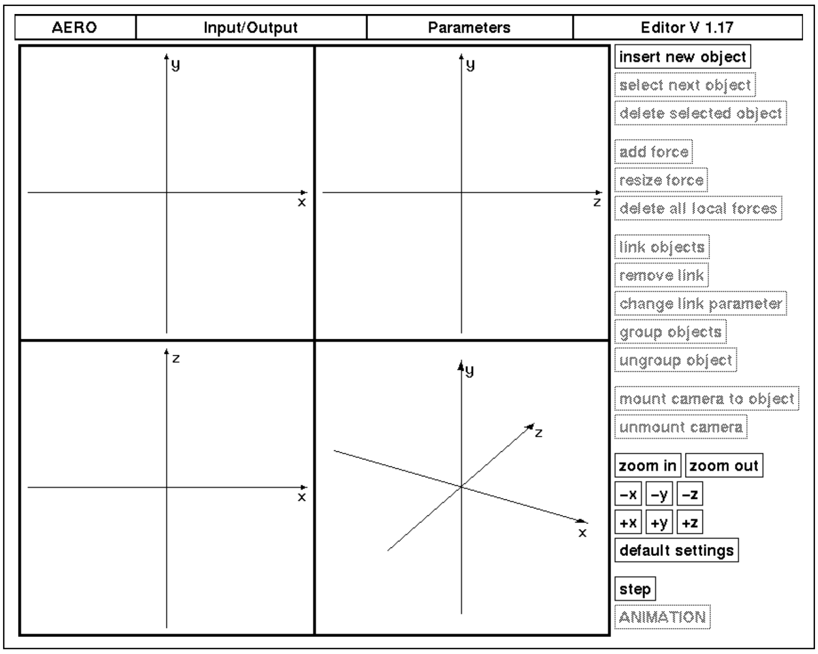

Figure 2: Editor window

Objects can be selected and placed in the different views using the menus provided. Various fundamental bodies may be selected: spheres, cylinders, cuboids[1], planes and fixed points. Object parameters, such as size and color, can be altered using a special object window (see figure 3). By selecting different materials, the user can influence physical characteristics, such as density (thus determining the mass and weight of an object) or coefficient of friction and elasticity.

Objects can be connected to one another. Two objects are selected for every connection and the type of connection is specified. Available are rigid connections, rods, springs, dampers and joints, where additional parameters may be specified (for example minimum active separation and spring constant for the spring connections). Multiple connections between two bodies are also possible. For example, one could construct a cuboid with multiple springs connected to a plane at different locations or a shock ab-

1. Cuboids are right parallelepipeds. Basically, these are three-dimensional boxes in which the height, width and length can differ from one another, but all surfaces have right angles at their corners.

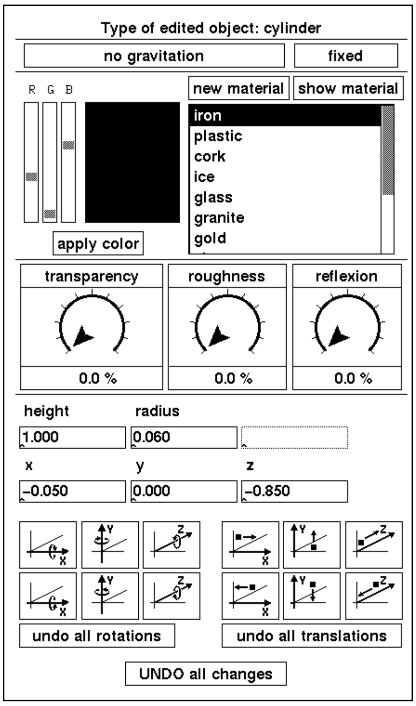

Figure 3: Object settings

sorber realized through a spring and a damper. Complex objects can be constructed this way. The editor window in figure 4 displays a scenario with a number of bodies and connections between them.

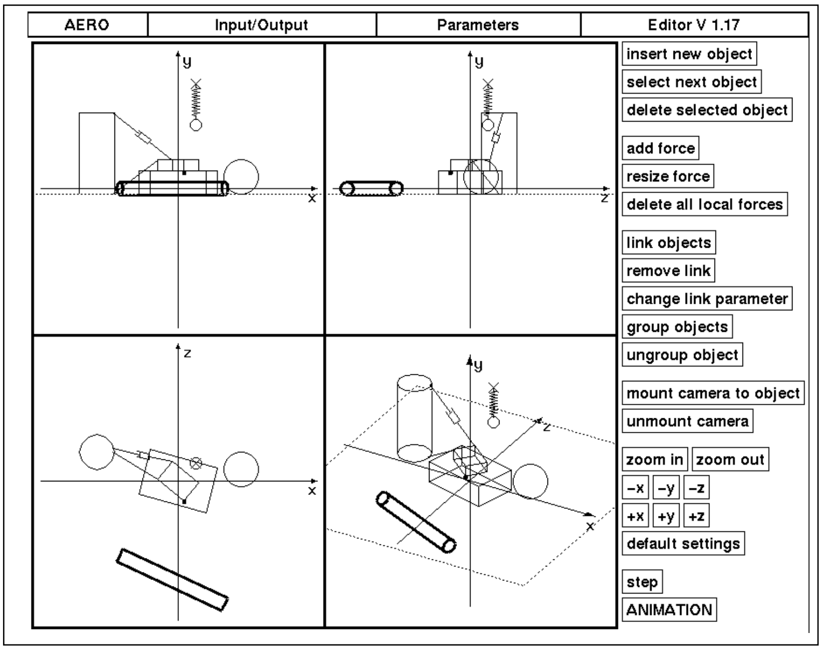

Figure 4: Editor window with objects

In addition, forces can be specified. The user can accelerate or induce rotation of bodies in a defined direction using forces. Forces are specified in three different ways:

- local to a single body
  (for example, a force constantly applied tangentially to a sphere in order to create a rotational moment)
- relative to another body
  (a force that is always exerted in the direction of another body, allowing *following*)
- absolute with respect to the reference frame
  (for the unidirectional acceleration of an object)

For the application of forces, a certain time interval has to be specified.

2.2 Animation

In figure 5 an impression of a wire frame animation sequence is given. An animation

Figure 5: Images from animation sequence

can be started using a scenario entered via the methods described above. At that point, a new window will be opened, in which the animation occurs (see figure 6).

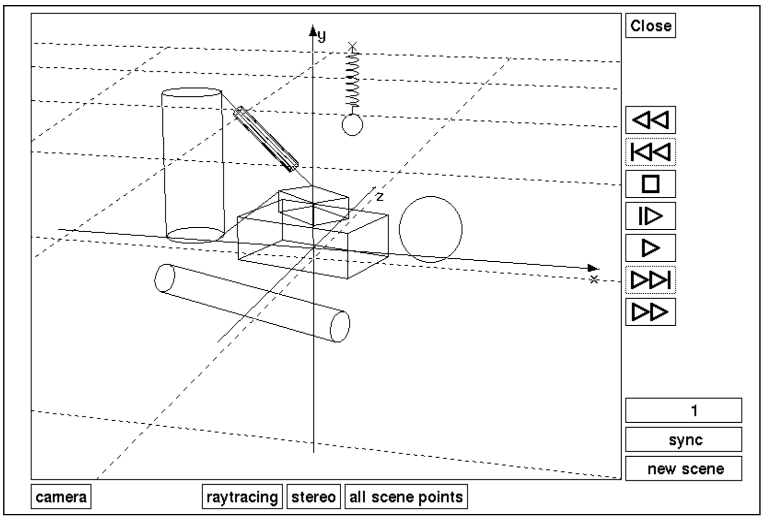

Figure 6: Animation window

At the right border of the animation window are buttons resembling those of a video recorder (see figure 7). These have a very similar functionality in the *AERO* system.

The start button is used to begin the viewing of an animation. The other buttons can be used to influence the viewing of the animation accordingly.

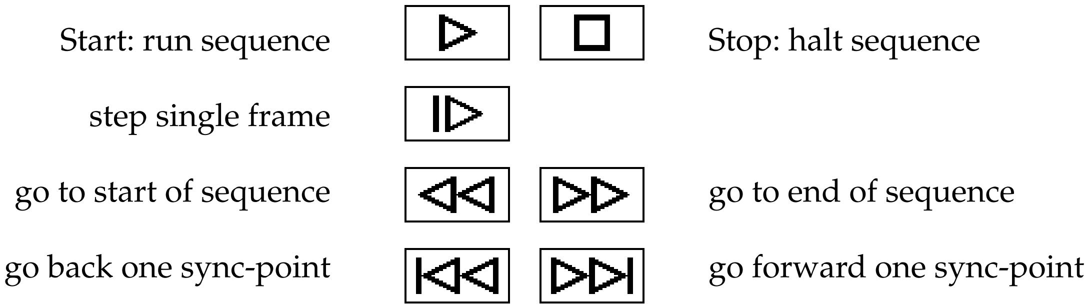

Figure 7: Animation functions

The position of the (virtual) camera in the animation window can be specified, in order to view the animation from a different location. This camera position can be altered at any point during the animation playback.

An uncommon concept in the *AERO* system are synchronization points (*sync points*). Only the start state and current state of the animation are maintained during its playback. All playback functions use state transition functions that convert the current state into the following state. Unfortunately, the calculations cannot simply be reversed in order to achieve the corresponding reverse playback. However, one should still be able to reverse the animation in small steps. This is where sync points are useful. The state of the animation at any point can be declared a sync point. This means that the playback control stores this state and the user can jump back to it at any time. Since any number of sync points can be declared, the sequence can be divided up as finely as necessary. If one wants to reverse the animation by a small amount, one simply jumps back to the previous sync point and inches forward image by image until the desired point is reached. The scene can be altered at any point in time. This is similar to a sync point, however, the transition to the next state does not occur via the state transition functions. Instead, the new scene information is adopted at this *scene sync point*. If one returns to the editor by closing the animation window, the scenario can be altered at that point in the animation. Film cuts in the animation can be generated this way.

The generation of output for the ray tracer can be engaged during the animation. In this case, a new file is generated for every image frame, which contains the current state in the form of the scene description language of the applicable ray tracing program.

2.3 Presentation of Objects

AERO bodies are presented differently in the different modes (see figure 8). In the edtor, it is necessary to be able to determine accurately the distances between objects. A perspective contortion would be out of place and therefore a parallel projection is used for presentation. In addition, connections are displayed only as symbols for the purpose of clarity.

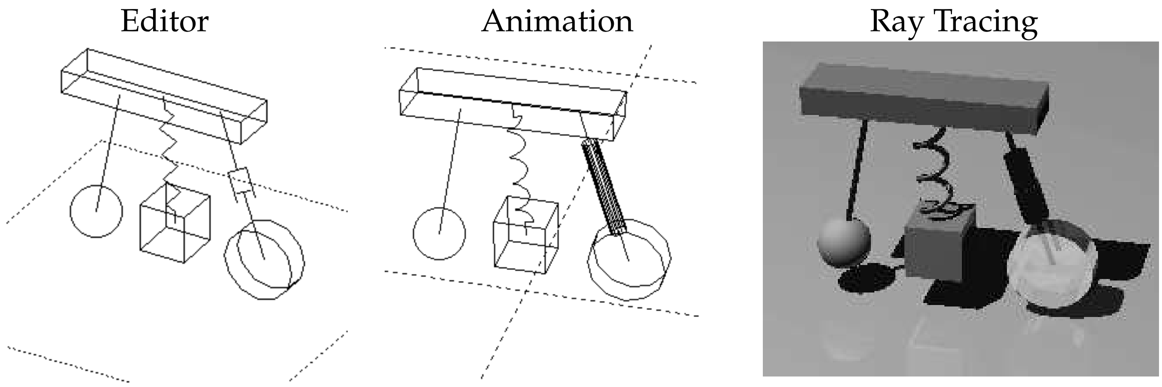

Figure 8: Representation of various objects and connections

The presentation in the animation window of *AERO* should occur as realistically as possible. Therefore, a perspective projection was selected. In addition, the connections have their correct shapes. A spring connection is shown as a spiral spring and a damper is cylinder-shaped.

If photo realistic output via the ray tracer is selected, the objects are rendered with a material-dependent surface texture. In this way, a body made of wood will have a wood grain texture. Glass bodies are clear and light refraction is also taken into account.

2.4 Stereo Image Display

Images are two-dimensional due to the fact that they are displayed on a surface and the third dimension, depth, is missing. Humans, can perceive things three-dimensionally, viewing images with two eyes from slightly different viewpoints. The brain can deduce distance information from the differences, or more generally the brain *senses* depth. In order to achieve a three-dimensional effect for an animation sequence, each of the eyes must be provided with the two-dimensional image that it would have seen in reality (images which differ slightly in their viewpoint). This is achieved in 3D films by using two cameras mounted side by side with eye separation on a tripod. The second image can be generated by a rendering tool just like the first, except that the second virtual camera is about 6-7cm to the side of the first and has a slight inward convergence in order to compensate for the parallax of human eyes.

One common method of providing each eye with separate images is the so called red-green 3D image representation (see figure 9). The observer wears a pair of glasses, whose right lens is green and left lens is red. The image for the right eye is presented in red simultaneous to the image for the left eye, which is green. Due to the color filtering of the lenses and the fact that green and red are complementary colors, both eyes view only the image intended for them. However, only a pseudo-monochrome presentation is possible. This 3D presentation method is implemented in the animation window of *AERO* as wire frame graphics.

Another more convenient way of viewing stereo images are 3D displays. A liquid crystal display (LCD) in front of the CRT monitor alternately polarizes the light coming from the tube in two opposing directions. In order to prevent flicker, these displays have to use twice the regular image refresh rate. Images corresponding to the

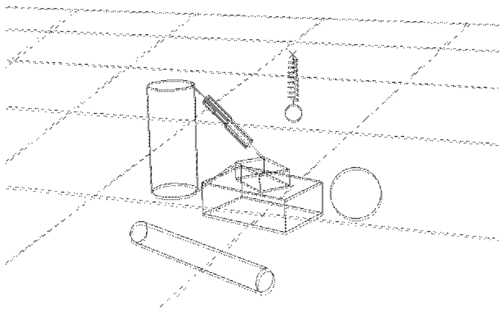

Figure 9: Stereoscopic red/green presentation of animation

right or left eye are displayed in synchronization with the direction of polarization. If the observer wears a special pair of glasses with accordingly polarized lenses, then each eye sees exactly one image in full color. Due to the alternating polarization of the display, the observer sees alternately one image in the left eye and then the other image in the right eye. If the image alternation is fast enough, the switching will not be noticed and a three-dimensional effect is achieved. *AERO* allows the generation of such images when used together with a ray tracing rendering system.

3. Simulation Model

The purpose of a simulation model is a computable representation of mechanical processes, which approximates reality accurately enough under the desired conditions. On the one hand, the response time in the interactive system should be as short as possible, and on the other hand as many physical effects and laws as possible should be taken into consideration in order to insure realistic movement. In order to compute object movements over time, so-called *motion equations* have to be established. These provide a mathematical relationship between the motion variables like position, velocity and acceleration, and the applied *forces*.

Another important point is the modelling of physical characteristics. The restriction to rigid spatial *bodies* comes from the well founded and tested knowledge of those systems which can be applied. There are also models which allow for elastic, deformable bodies. However, since the bases of these models are finite element methods, they are not workable in a time critical system such as *AERO*. Therefore, liquids and gases are also not modelled. Only a rudimentary *air resistance* is provided. In order to prevent objects from passing through one another, either the impulse of a *collision* is applied or the forces of *touching* objects are applied. Before this *collision processing* can be undertaken, *collision recognition* has to be done. This includes computation of the contact section between the objects and further data required for collision processing. Constraints[1] define the coupling between bodies in the form of *connections*. These are input into the motion equations as forces just like gravity and friction. Finally, user specific forces can be applied in order to set bodies in motion.

1. These are usually equations depending on the position of the body and having the form $\varphi(x_1, \ldots, x_n) = 0$.

3.1 Position and Orientation of a Body in the Virtual Environment

To start with, the user must define the position and orientation of a body in three-dimensional space. A vector[1] $\vec{r}_{OS}$ from the origin O of the fixed space coordinate system K to the center of gravity S is sufficient to describe the position of a body. Euler parameters (Quaternions) are used to describe the rotation between the K′ coordinate system, which is fixed with respect to the body, and the fixed space coordinate system K. K′ is derived by rotating K clockwise around the rotational axis $\vec{u}$ by the angle Υ. Using this scheme, the four Euler parameters $q = (q_0, q_1, q_2, q_3)$ are calculated according to:

$$q_0 = \cos\frac{\Upsilon}{2}$$

$$(q_1, q_2, q_3)^T = \vec{u}\sin\frac{\Upsilon}{2}$$

One of the nice features of quaternions is the simple summation of rotations through

$$q \times \upsilon = (q_0, \vec{q}) \times (\upsilon_0, \vec{\upsilon}) = (q_0 \cdot \upsilon_0 - \vec{q} \cdot \vec{\upsilon}, q_0 \cdot \vec{\upsilon} + \upsilon_0 \cdot \vec{q} + \vec{q} \times \vec{\upsilon}) \ .$$

Matrix A provides a translation from vector $\vec{r}'$ in coordinate system K′ to vector $\vec{r}$ in coordinate system K:

$$A = \begin{bmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1 q_2 + q_0 q_3) & 2(q_1 q_3 - q_0 q_2) \\ 2(q_1 q_2 - q_0 q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2 q_3 + q_0 q_1) \\ 2(q_1 q_3 + q_0 q_2) & 2(q_2 q_3 - q_0 q_1) & 2(q_0^2 + q_3^2) - 1 \end{bmatrix}$$

Using this matrix, we can derive the relationships $\vec{r}' = A \cdot \vec{r}$ and $\vec{r} = A^{-1} \cdot \vec{r}'$. The inverse matrix A^{-1} is obtained by simply reversing the rotational axis $\vec{u}$. This is achieved by replacing (q_0, q_1, q_2, q_3) with $(q_0, -q_1, -q_2, -q_3)$ in the matrix above. A detailed description of Euler parameters, Euler angles and Bryant angles can be found in [Wittenburg 77]; the relationship between two rotations is described in [Shoemake 85].

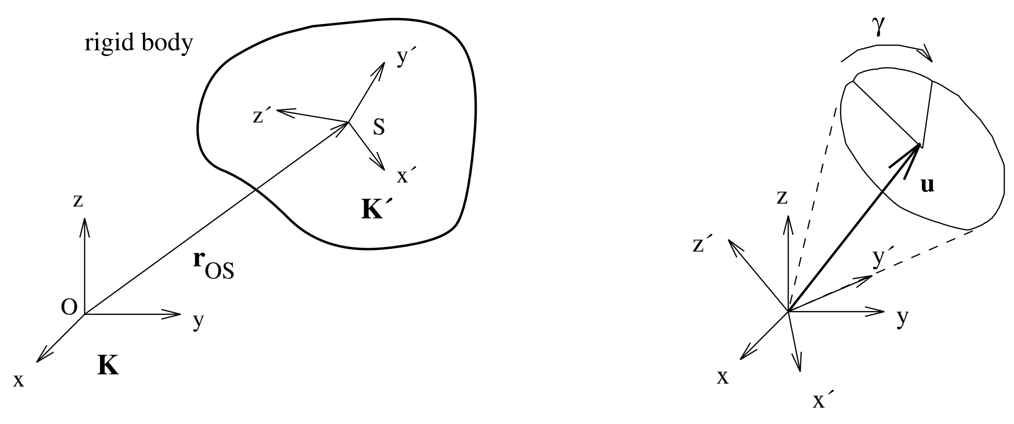

Figure 10: Position of a body and rotation between two coordinate systems

1. In figures, vectors are printed in bold type face. Vectors with an apostrophe are with respect to the K′ coordinate system.

The following formula, using the matrix above, defines the location of a point fixed with respect to a body:

$$\vec{r}_{OP} = \vec{r}_{OS} + \vec{r}_{SP} = \vec{r}_{OS} + A^{-1} \cdot \vec{r}'_{SP}$$

In addition, the following holds for the velocity $\vec{\upsilon} = \dot{\vec{r}}$ of the point P:

$$\vec{\upsilon}_P = \vec{v}_S + \vec{\omega} \times \vec{r}_{SP}$$

Here $\vec{\omega}$ represents the angular velocity vector, whose direction specifies the rotational axis about the center of gravity. $|\vec{\omega}|$ represents the angular velocity about this axis.

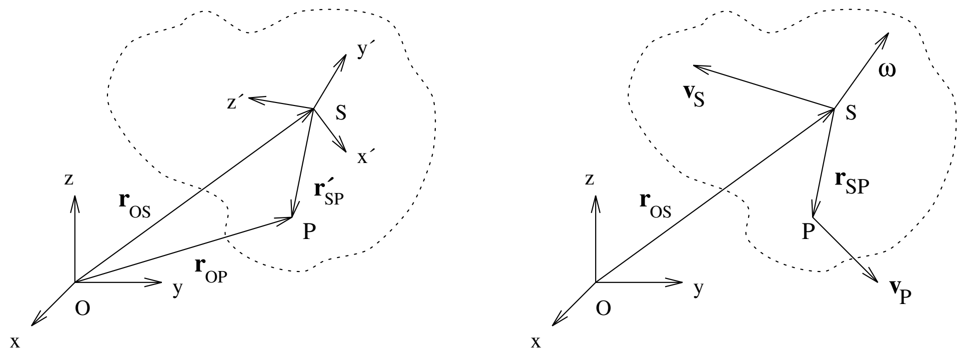

Figure 11: Position and velocity of a point fixed with respect to a body

3.2 Rigid Body Systems

A model using six fundamental bodies was implemented in an effort to provide basic building blocks, with which the user can experiment. These bodies have geometric forms and physical characteristics corresponding to real materials. In order to minimize computational complexity, a system of rigid or non-deformable bodies should be used. These systems are known in mechanics as rigid body systems. The use of a few simple but generalized spatial forms should avoid the necessity of approximating complex objects with polygonal surfaces and thereby reduce the difficulty of collision processing.

A body has several different *attributes*: mass[1], physical size and material characteristics. Each of the attributes has impact on the simulation. For instance, motion due to the application of force is only possible when the mass is known. Only bodies with physical size can collide. Finally, the material characteristics influence collision, contact and friction between bodies. Not all of the bodies in figure 12 have all of the attributes. Except for the immovable point and point mass, all of the bodies have physical size and therefore material characteristics as well. However, an infinite plane has no mass, since it is only a two-dimensional object. This means that planes cannot move. Since the immovable point also has mass zero, it provides a fixed point from which other bodies can be influenced via connections. The point mass can move, however it is not influenced by physical size, i.e. from collisions or contact with other objects.

1. Specifically, mass m and inertia tensor J, where the value of the latter for common spatial forms can be found in any technical reference book.

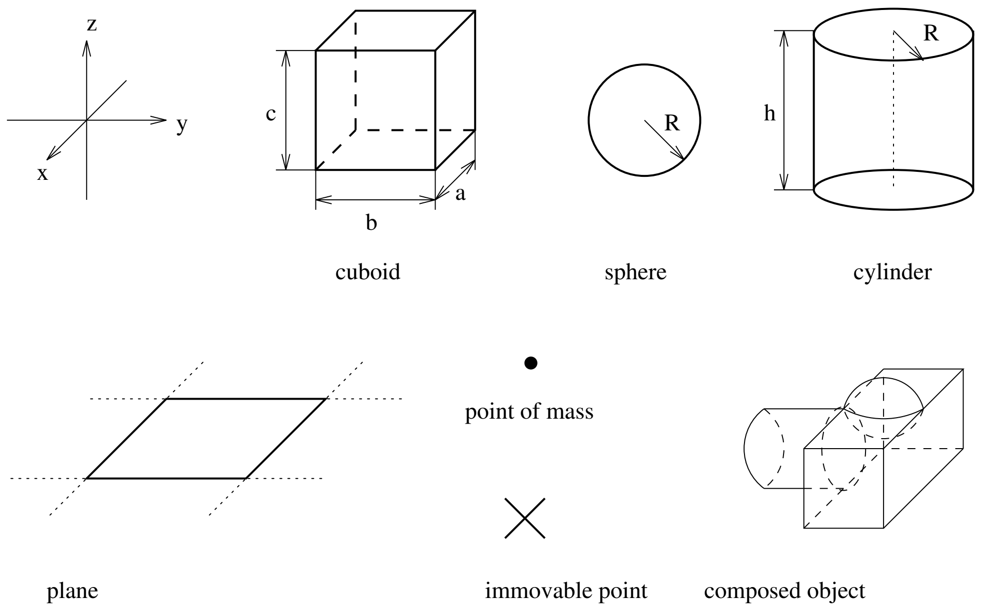

Figure 12: Body types

Compound bodies can be composed from point masses, cuboids, spheres, and cylinders. These allow the construction of complex rigid bodies. The body parts, which are fixed with respect to each other, are simulated with their corresponding material characteristics. In addition to the three attributes described above, there are others which control visual presentation such as color, texture, etc. However, these attributes are irrelevant for the calculation of the state transition.

3.3 Motion Equations

By choosing the center of gravity as reference point for a fixed body, the resulting equations for momentum and spin take on their simplest form. Together they represent the motion equations for a body.

$$m \cdot \ddot{\vec{r}}_{OS} = \sum_{i=1}^{n} \vec{F}_i \qquad \text{Linear Momentum}$$

$$J_S \cdot \dot{\vec{\omega}}_S = \vec{M}_S + \sum_{i=1}^{n} (\vec{r}_i \times \vec{F}_i) \qquad \text{Angular Momentum}$$

Where $\vec{F}_i$ are the forces applied to the body of mass m, $\ddot{\vec{r}}_{OS} = \dot{\vec{\upsilon}}_S = \vec{a}_S$ is the acceleration of the mass at the center of gravity and $\dot{\vec{\omega}}_S$ is the rotational acceleration. The matrix J_S is the so-called inertia tensor and represents the spatial mass distribution about the center of gravity. It is comparable to the inertial characteristics of the mass m in the momentum equation. For instance, the rotational velocity of pirouetting ice skaters is smaller with their arms extended due to a larger rotational moment than with their arms retracted.

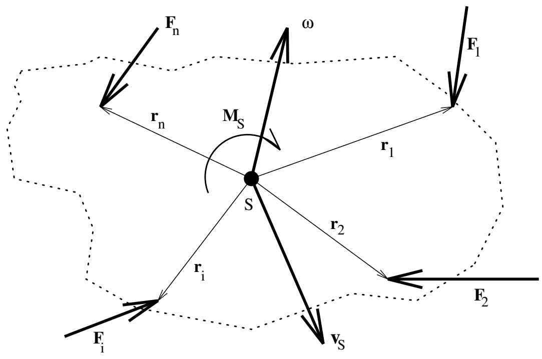

Figure 13: Forces exerted against a rigid body

Since the unknown $\vec{r}_{OS}$ is represented by its derivative with respect to time, these equations are known as a system of differential equations. This system can be solved using *numerical integration*.

3.4 Connections

A connection links a point A of one body with a point B of another body via a mathematical relationship. In this way, a force[1] corresponding to the type of connection is applied at the connection points.

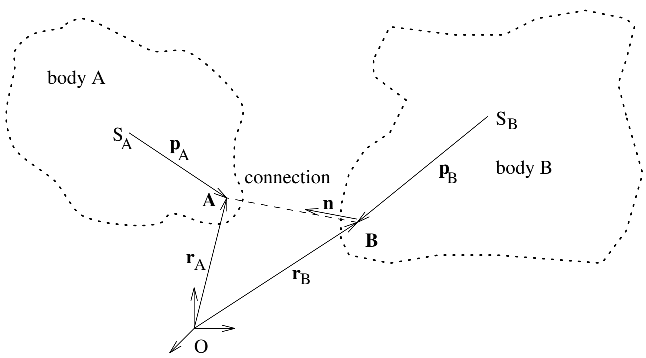

Figure 14: Connections between two bodies

Springs, dampers and rods, which consist of a combined spring and damper members, were selected as connection types due to their universal applicability, well-known functionality and simple implementation. By setting the rod length to zero, a ball-and-socket joint can be approximated.The force equations are calculated at each point in time and the results are fed into the motion equations.

It should be made clear at this point, that a rod consisting of a spring and damper must have its length altered in order to exert a force. This is a type of shock absorber. However, by setting the appropriate parameters, one can simulate a rod with sufficient accuracy.

1. $\vec{n} = \vec{r}_A + \vec{r}_B$ is the normal vector between points A and B with $|\vec{n}| = 1$ and $x = |\vec{r}_A - \vec{r}_B|$ is the distance between the two points. The velocity is given by: $\dot{\vec{r}} = \dot{\vec{r}}_S + \vec{\omega} \times \vec{p}$.

A general model for coupling different bodies can be achieved using constraints. [Witkin 90], [Barzel et al. 88] and [Isaacs 87] give analytical methods for the more general connection types, including the implementation of fixed rods and joints. These methods are commonly used for technical and scientific purposes[1] and are based on analytical mechanics.

These methods were not selected for two reasons:

- If contact or frictional forces must also be analytically calculated, the resulting systems of equations are virtually unsolvable.
- The rigid connections would also impart momentum during collisions, which would make the collision processing even more complex.

3.5 Adding External Forces

In order to manipulate bodies over time, constant forces can be applied at a fixed point on a body during specified intervals. The direction of the force can be specified either in the coordinate system of the body, such as $\vec{F}'_1$, in fixed space coordinates, such as $\vec{F}_2$, or by giving another point on a different body, such as $(\vec{r}'_4, \vec{F}_3)$. Smooth, accelerated translational and rotational motion can be generated using these three types of forces.

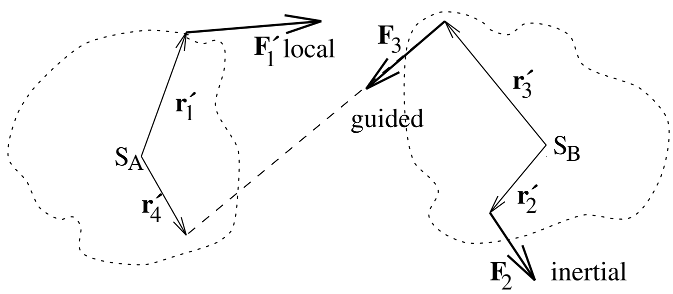

Figure 15: Types of force

For example, rotational motion can be achieved, as shown in figure 16, by the application of two equal forces directed within the coordinate system of the body. The translational acceleration of the two forces annihilate each other, leaving the resulting moment: $\vec{M} = 2 \cdot \vec{r}' \times \vec{F}'$.

3.6 Collisions

A generalized representation of a collision between two bodies is shown in figure 17. In addition to the collision normal $\vec{n}$ and the penetration depth Υ, the determination of the normal velocity of the two bodies at the collision point P is important.

$$\upsilon_N = \vec{n} \cdot (\vec{\upsilon}_1 + \vec{\omega}_1 \times \vec{p}_1 - \vec{\upsilon}_2 - \vec{\omega}_2 \times \vec{p}_2)$$

1. Due to the fact that they allow exact calculation of constraint forces. For more information, see [Haug 84].

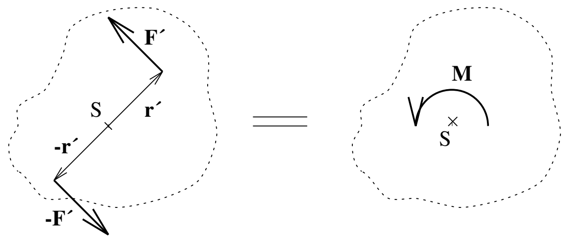

Figure 16: Generation of rotational moment using two forces

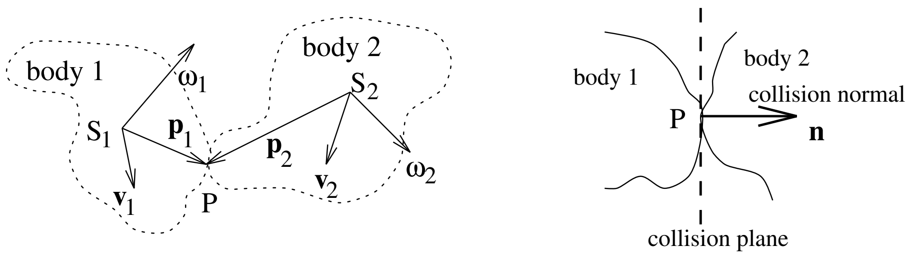

Figure 17: Collision between two bodies

If $\upsilon_N = 0$, then P is a point of contact (see next section). In the case of $\upsilon_N < 0$, the two bodies are separating. A collision only occurs if $\upsilon_N > 0$.

The collision value ε represents the material behavior of the bodies involved:

- $\varepsilon = 1$: elastic collision -- a ball thrown against the wall returns with the same velocity (υ_{N1} and υ_{N2} exchange values).
- $0 < \varepsilon < 1$: partially elastic collision -- incidental velocities are distributed according to $(\upsilon_{N1} - \upsilon_{N2}) = \varepsilon(\upsilon'_{N1} - \upsilon'_{N2})$.
- $\varepsilon = 0$: non-elastic collision -- incidental velocities of the two bodies are the same after the collision.

Two routines for collision handling have been implemented:

Collision Processing via Springs

When a collision occurs, a very stiff spring is inserted at the point of contact. The strength of the spring is proportional to the penetration depth. The spring constant c depends on the materials of the colliding bodies. The collision elasticity of the materials is also taken into account.

$$\vec{F} = \begin{cases} \Upsilon c \vec{n} & \text{for } \upsilon_N \geq 0 \quad \text{(approaching)} \\ \varepsilon \cdot \Upsilon c \vec{n} & \text{for } \upsilon_N < 0 \quad \text{(separating)} \end{cases}$$

The disadvantage of this method are the large computational costs arising from the high values of the spring constant c increasing the time granularity of the integral solution process. If c is selected too small, the bodies will penetrate each other depending on their mass and velocity and may even pass through each other.

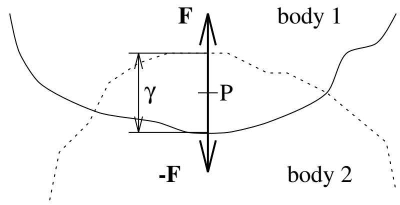

Figure 18: Collision processing via springs

Analytical Collision Handling

The analytical method determines the new velocities $(\vec{\upsilon}'_1, \vec{\omega}'_1, \vec{\upsilon}'_2, \vec{\omega}'_2)$ after the collision from the current translational and angular velocities $(\vec{\upsilon}_1, \vec{\omega}_1, \vec{\upsilon}_2, \vec{\omega}_2)$ by applying momentum and spin conservation principles (no energy is lost).

$$m_1\vec{\upsilon}'_1 = m_1\vec{\upsilon}_1 + \vec{R}$$

$$m_2\vec{\upsilon}'_2 = m_2\vec{\upsilon}_2 - \vec{R}$$

$$J_1\vec{\omega}'_1 = J_1\vec{\omega}_1 + \vec{p}_1 \times \vec{R}$$

$$J_2\vec{\omega}'_2 = J_2\vec{\omega}_2 - \vec{p}_2 \times \vec{R}$$

In order to ensure that the collision impulse $\vec{R}$, which is exchanged between the bodies during the collision, is exerted in the direction of the collision, the following restrictions are used in addition:

$$\vec{R} \cdot \vec{e}_1 = 0$$

$$\vec{R} \cdot \vec{e}_1 = 0$$

$$\vec{n} \cdot (\vec{\upsilon}'_2 + \vec{\omega}'_2 \times \vec{p}_2 - (\vec{\upsilon}'_1 + \vec{\omega}'_1 \times \vec{p}_1)) = \varepsilon \cdot \vec{n} \cdot (\vec{\upsilon}'_1 + \vec{\omega}'_1 \times \vec{p}_1 - (\vec{\upsilon}'_2 + \vec{\omega}'_2 \times \vec{p}_2))$$

This system of linear equations with 15 unknowns[1], $\vec{\upsilon}'_1, \vec{\upsilon}'_2, \vec{\omega}'_1, \vec{\omega}'_2$ and $\vec{R}$ can be solved with standard methods such as the Gauss algorithm. Altogether the analytical method is faster than the method using springs because it only needs to be carried out once at the point of collision and not over a long period of time. The path of a sphere falling to the ground from a height of 0.05m can be seen in figure 19.

Based on [MacMillan 60], [Moore et al. 88] indicates that one can take both friction as well as the propagation of collision impulses through connected bodies into account. [Wang et al. 92] describes the case of friction for two-dimensional applications and [Wittenburg 77] dedicates a chapter to collision processing for systems containing compound bodies. However, the critical problem of handling multiple simultaneous collisions at one body remains therein unsolved.

1. This can easily be reduced to 13 unknowns since $\vec{R}$ runs parallel to $\vec{n}$.

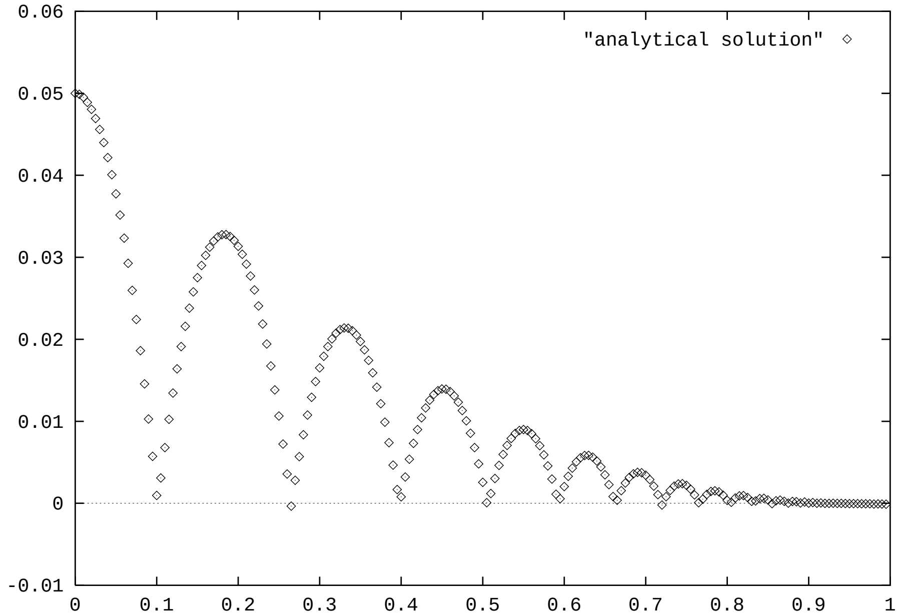

Figure 19: Path of a sphere falling to the ground with $\varepsilon = 0{,}81$.

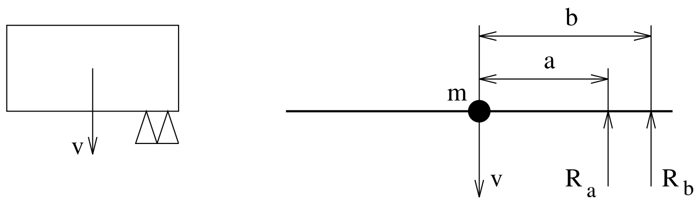

Figure 20: Cuboid with two asymmetrical points of collision

Consider a cuboid, such as the one in figure 20, that experiences collisions at two different places simultaneously. Simplified, the following results from the collisions:

$$m\upsilon' = m\upsilon + R_a + R_b$$

$$Jw' = Jw - R_a - R_b$$

$$\left.\begin{array}{l} \upsilon' + w'a = 0 \\ \upsilon' + w'b = 0 \end{array}\right\} \text{since} \quad a \neq b \Rightarrow w' = 0$$

This means that the cuboid would not begin to rotate, even though the collisions are asymmetrical. As long as there is no easily applicable physical model for multiple, simultaneous collisions, one has to handle simultaneous collisions serially. The ***AERO*** system carries out collision processing on points with $\upsilon_N > 0$ until all points have separated from each other with $\upsilon_N \leq 0$. However, a maximum of $2k$ steps are carried out, where k is the number of collisions occurring.

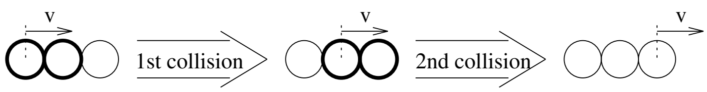

Figure 21: Propagation of collision impulse with three spheres

This form of collision propagation has the disadvantage, that if a cuboid falls flat onto a level surface, it begins to rotate on an axis parallel to the surface as shown in figure 22. For this reason, the user can select for collision handling either the analytical method or the spring method (which does not have the problem shown above), whichever is best suited for the intended simulation.

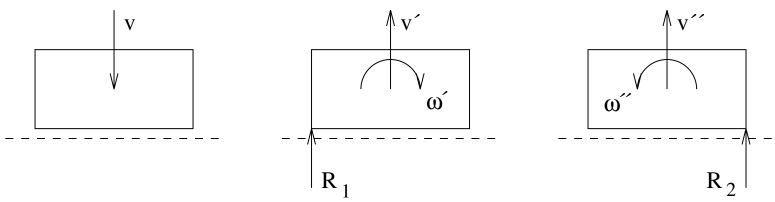

Figure 22: Cuboid falls flat onto plane

3.7 Handling of Physical Contact

Theoretically, one could apply the collision handling methods discussed in the previous chapter to points of contact having $\upsilon_N \approx 0$ as well. In fact, this is exactly what happens when the user selects the spring method for collision handling. In contrast, the analytical method of collision handling is too computationally complex to use with every state transition. Instead, the "penalty method" is used, which has a better transient behavior than the spring method. Every point having $|\upsilon_N|$ less than a specifiable maximum contact velocity $\upsilon_{N\max}$ will have a force applied opposing the direction of penetration. This force is proportional to the penetration depth Υ and proportional to the velocity along the contact normal $\vec{\upsilon}_N$. The elasticity factor c and the damping factor d are available as parameters for every type of material. Extended transient periods as well as strong oscillation upon contact can be avoided during the contact process through a suitable selection of d. figure 23 shows the settling of an iron sphere in meters. The horizontal time axis is in units of seconds.

The analytical handling of contact forces, as described in [Baraff 89], was not implemented. However, it certainly represents a worthwhile goal, because it avoids the problem of different penetration depths of bodies as shown in figure 24. In addition, it would allow the consideration of constraint forces. A subroutine for solving LCP[1] problems using *Lemke's Complementary Pivot Algorithm* was implemented as well and may be included in an extended version.

1. Linear Complementarity Problem: find the vectors w and z for $w - Mz = q \wedge w \geq 0 \wedge z \geq 0 \wedge w_i z_i = 0 \quad \forall i$, where the matrix M and the vector q are given. For more details, see [Murty 88].

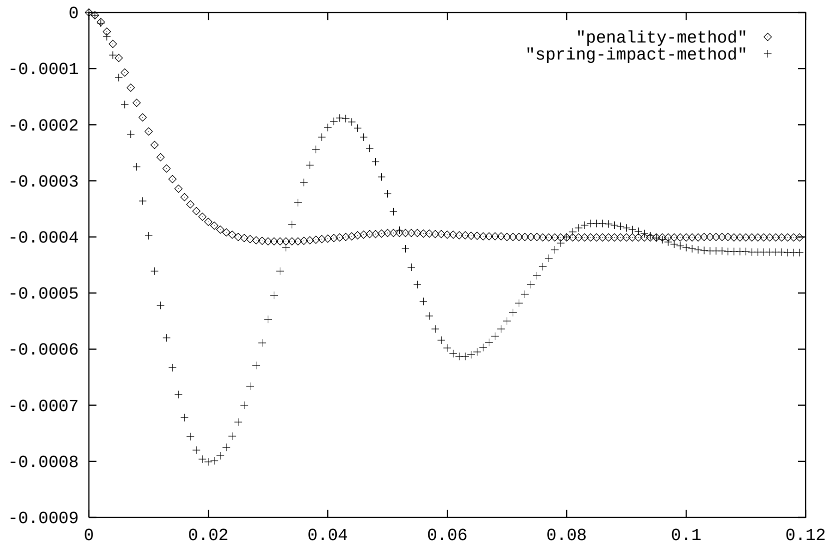

Figure 23: Comparison of methods for settling a sphere in to a plane

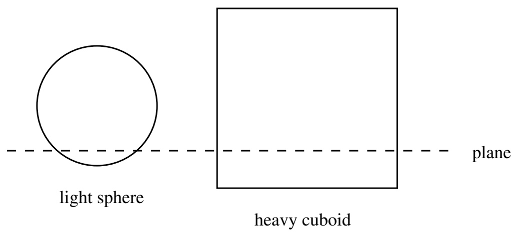

Figure 24: Contact handling using the penalty method

3.8 Friction

When two touching bodies move relative to one another, friction creates a force opposing the direction of movement.

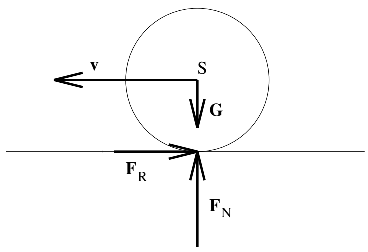

Figure 25: Dynamic friction induces rotation in a sliding sphere

This type of friction is known as dynamic friction. The force created by the dynamic friction $\vec{F}_{R0}$ can easily be calculated from:

$$\vec{F}_{R0} = -\frac{\vec{\upsilon}_T}{|\vec{\upsilon}_T|} \cdot \mu_0 \cdot |\vec{F}_N|$$

F_N is the force of contact which was calculated in the previous section. The coefficient of dynamic friction μ_0 is a value in the range $[0\ldots1]$ and depends on the pair of materials in contact. As has been seen before, implementing dynamic friction is no problem. Static friction, on the other hand, must be approximated. The force of static friction $\vec{F}_R$ usually applies only when the tangential velocity at the point of contact $\vec{\upsilon}_T = 0$ and holds the body at a standstill. In addition, $\vec{F}_R$ always lies within the plane of contact and is never larger than the force of contact $\vec{F}_N$, or more exactly: $|\vec{F}_R| \leq \mu \cdot |\vec{F}_N|$.

In *AERO*, the restriction $\vec{\upsilon}_T = 0$ is lifted and all points of contact are handled with $\vec{\upsilon}_T \leq \upsilon_{Tmax}$. The dynamic friction formula is used for points of contact having a tangential velocity above the maximum υ_{Tmax}. Instead of having an unknown direction of application, the force of static friction is applied opposing the direction of movement within the plane of contact. This results in the following approximation:

$$\vec{F}_R = -\frac{\vec{\upsilon}_T}{\upsilon_{Tmax}} \cdot \mu \cdot |\vec{F}_N|$$

It should be apparent, that this static friction will not bring a body to an absolute standstill. One can only select the smallest possible parameter values so that no visual affects are apparent in the animation. The problem here is that this would require the integration method to use very fine steps. Actually, the contact forces and frictional forces should be placed in a large system of equations in order to achieve an analytical solution, which would provide the correct. As shown in [Baraff 91], such mixed systems of equations are difficult to solve with current mathematical methods.

3.9 Gravity and Air Resistance

In order to prevent bodies from behaving as if they were in zero gravity in outer space, a constant force of gravity g is applied everywhere in the negative y direction. Naturally, a value for g differing from 9,81 m/s^2 can be selected in order to move the simulation to the surface of another planet. Every body also has an extra attribute "gravitation" that allows one to turn off the gravity for just that particular object.

Another option is allowing the force of gravity to apply somewhat randomly with a specified percentage. This has the advantage, that special cases such as stacking spheres on top of one another do end with the expected collapse. On the other hand, it should be pointed out that replacing constant gravity with a random function having an average value g also has unwanted side effects. This will lead to uncontrolled energy changes in free falling bodies. The energy of a body is calculated as:

$$E = E_{\text{kin}} + E_{\text{pot}} = \frac{m}{2}\vec{\upsilon}^2 + \frac{1}{2}J\vec{\omega}^2$$

The random g directly influences $\dot{\vec{\upsilon}} = \vec{a}$. For a pendulum, which changes potential energy into kinetic energy, the energy changes would lead to changes in the height and period of the swing. Another side effect is that stationary objects sitting on the ground are more easily brought to sliding.

In addition to the force of gravity, a force of air resistance can be applied opposing the direction of travel. This force increases with the square of the velocity and includes a body specific air resistance factor c_w. The attack surface A is calculated as a constant from the dimensions of the body. The formula used in *AERO* does not reflect true flow behaviour, but serves quite well as a simple way for generating a resisting force:

$$\vec{F}_{Air\mathrm{Resistance}} = -c_w \cdot A \cdot |\vec{\upsilon}| \cdot \vec{\upsilon}$$

3.10 Collision Detection

In order to engage the collision handling process, collision detection must recognize the penetration of one body within another and calculate the data required for the collision or contact handling methods. Collisions are recognized by checking each body against every other body for overlapping regions. Those pairs of bodies that are in contact are collected in a list.

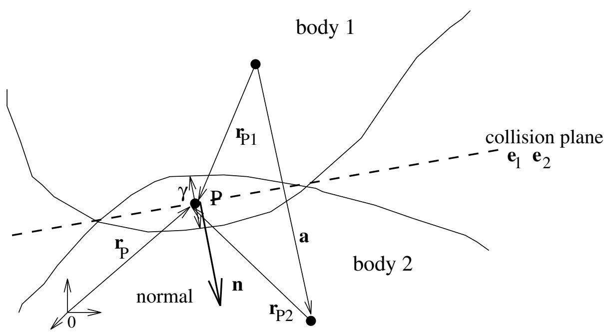

Figure 26: Collision detection

The assumption is made, that only a few of the points of contact need to be taken into consideration. For instance, when two cuboids are in contact, only the corners of the overlapping region are calculated. These points are then used for the application of contact and frictional forces as well as for the collision impulse.

The parameters relevant for a collision are the collision point $\vec{r}_P$ and the vectors from the bodies' centers to that point: $\vec{r}_{P1}$ and $\vec{r}_{P2}$. Also important are the collision normal $\vec{n}$ and the penetration depth Υ. It is necessary, that a suitably large countering force be applied to a point P so that the penetration depth Υ is reduced. Exactly this behavior is required of the contact methods.

In spite of the fact that geometrically simple bodies were selected, $\frac{n}{2} \cdot (n+1) = 10$ routines are requires to handle the $n = 4$ bodies (cuboid, sphere, cylinder and plane). Unfortunately, even with these simple fundamental bodies, the collision tests proved to be computationally expensive. Therefore, due to run time considerations, the cylinder collision routines have not been imlemented yet. Cylinders are approximated by cuboids and handled using the cuboid collision routines.

4. Computational Methods

4.1 Integration of the Motion Equations

Since the motion equations form a system of differential equations, solving them via numerical integration is appropriate. Most integration methods are only applicable for systems of the form:

$$\dot{\vec{x}} = \vec{f}(t, \vec{x}) \wedge \vec{x}(t_0) = \vec{x}_0$$

The process will calculate the new state $\vec{x}(t+h)$ from the old state $\vec{x}(t)$. However, the motion equations represent a system of differential equations of the second order.

$$\ddot{\vec{r}} = \frac{1}{m}\sum_i \vec{F}_i$$

$$\dot{\vec{\omega}} = J^{-1} \cdot (\vec{M} + \sum_i (\vec{r}_i \times \vec{F}_i))$$

In order to construct a system of the first order, the following assumptions are made:

$$\dot{\vec{r}} = \vec{\upsilon}$$

$$\dot{\vec{q}} = g(\vec{q}, \vec{\omega})$$

Here, the main reason for using the Euler parameters to describe the orientation of a body is revealed. In contrast to Euler angles, which use three serially executed axis rotations, Euler parameters are more easily integrable. The following equation holds:

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = g(\vec{q}, \vec{\omega}) = \frac{1}{2}\begin{bmatrix} 0 & -w_1 & -w_2 & -w_3 \\ w_1 & 0 & w_3 & -w_2 \\ w_2 & -w_3 & 0 & w_1 \\ w_3 & w_2 & -w_1 & 0 \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

The state vector $\vec{x}$ contains for each body the position $\vec{r}$, the orientation $\vec{q}$, the linear velocity $\vec{\upsilon}$ and the angular velocity $\vec{\omega}$. The right side of the equation above corresponds to the problem dependent function vector $\vec{f}(t, \vec{x})$. The integration method is responsible for accumulating or summing the accelerations $\ddot{\vec{r}}$ and $\dot{\vec{\omega}}$, the velocities $\dot{\vec{r}}$ and $\vec{\omega}$, and the position $\vec{r}$ and orientation $\vec{q}$.

4.2 Integration Method

The selection of an integration method was suggested by [Heinzel 92] and involves an imbedded Runge-Kutta procedure[1] with error control and interpolation of intermediate states. The method of the order $s = 7$ calculates from the intermediate values

1. This is known as RK5(4)7FM in [Dormand et al. 80] and [Heinzel 92].

$$\vec{k}_o = \vec{f}(t, \vec{x}(t))$$

$$\vec{k}_i = \vec{f}(t + c_i \cdot h, \vec{x}(t) + h \cdot \sum_{j=0}^{i-1} a_{ij}\vec{k}_j) \qquad i = 1, \ldots, s-1$$

the new state as

$$\vec{x}(t+h) = \vec{x} + h \cdot \sum_{i=0}^{s-1} b_i \vec{k}_i$$

The step size h is selected through the error control according to the desired accuracy. By selecting another state value $\vec{x}'$ of a lower order using

$$\vec{x}'(t+h) = \vec{x} + h \cdot \sum_{i=0}^{s-1} b'_i \vec{k}_i$$

one can get an approximation of the error between the values for t and $t+h$ from $\delta = \|\vec{x} - \vec{x}'\|$. If the accuracy ε is specified, then whenever $\delta > \varepsilon$ holds, the state $\vec{x}(t+h)$ must be discarded and recomputed with a new step size h_{new}, to get more accurate results. A suitable formula for the new step size is:

$$h_{\text{new}} = 0{,}9h \left(\frac{\varepsilon}{\delta}\right)^{\frac{1}{p+1}}$$

Here, $p = 4$ is the order of the state equation for $\vec{x}'$, which is one less than that of $q = p + 1 = 5$. In an actual implementation, it is useful to constrain the step size to a specific interval in order to avoid unacceptably long response times. By using interpolation based on the previously generated intermediate values $\vec{k}_0, \ldots, \vec{k}_{s-1}$, the system can efficiently generate intermediate states $\vec{x}(t + \sigma h)$ within the interval $t + \sigma h \in [t, t+h]$. The state changes within this interval are approximated by using a polynomial of degree $d = 5$. This polynomial's term coefficients are

$$\vec{\alpha}_0 = \vec{x}(t)$$

$$\vec{\alpha}_j = h \sum_{i=0}^{s-1} \beta_{j-1,i} \vec{k}_i \qquad j = 1, \ldots, d$$

Computing an intermediate state occurs without evaluating the function $\vec{f}(t, \vec{x})$ by using:

$$\vec{x}(t + \sigma h) = \sum_{j=0}^{d} \alpha_j \sigma^j$$

The constants a_{ij}, c_i, b_i, b'_i and $\beta_{i,j}$ for the formulas above can be found in [Heinzel 92] or in the original texts [Dormand et al. 80], [Dormand et al. 81] and [Dormand et al. 86].

4.3 General State Transition Operation

The computational structure in figure 27 shows the interaction of the individual components of the state transition calculations used in the *AERO* system. From a state containing the location and movement of all bodies as well as the states of the connections and user specifics forces in effect, a new state at the point $t + \Delta t$ is calcu-

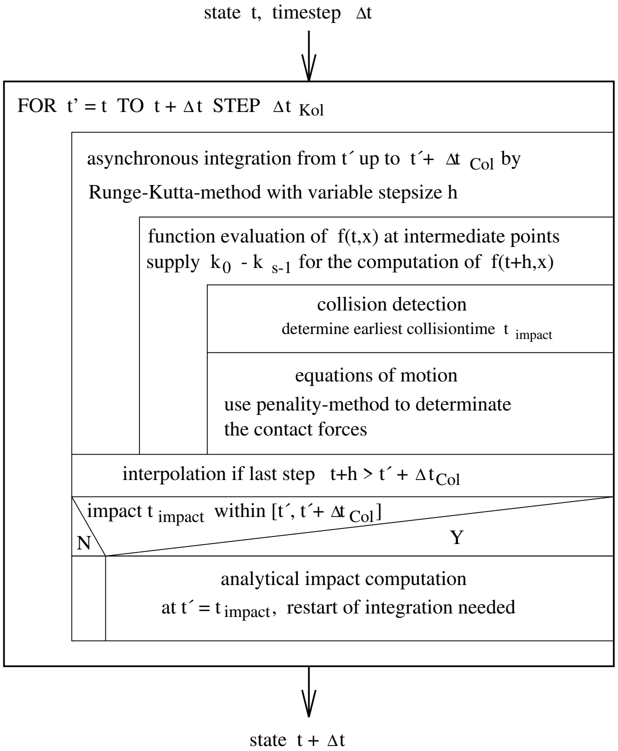

Figure 27: Schematic of the state transition calculations

lated. The integration method has the task of summing or accumulating the results of the motion equations from t to $t + \Delta t$. However, it became apparent, that the interval should first be divided into smaller pieces known as Δt_{Col} in order to allow a finer grained collision detection within the interval. In addition, when using the analytical collision handling method, the integration method must be restarted, since the newly calculated velocities are seen as start values for the systems of differential equations.

The motion equations are only nested within the integration method in order to achieve the correct processing order. The integration method is implemented fully independent of the problem at hand. The selected modelling of the forces (gravity, contact, friction and air resistance) allows direct calculation from the state variable $\vec{x}$. Since they only use the position and velocity, they are directly inserted into the linear and angular momentum equations. However, in order to take the contact forces into account, a collision detection must have occurred previously. This is handled by recording the collisions as soon as they are recognized and storing them to be handled at a later point. One should note that since the step size h is variable, the integration method occurs asynchronously to the requested states at $t' + t'_{Col}$. When the integration method takes larger steps, the intermediate states are approximated via interpolation. If it takes smaller steps, intermediate states must be inserted.

4.4 Simulation Requirements

In order to carry out a simulation true to the model, four conditions must be met. Due to their mutual interaction, these conditions cannot be viewed as being independent of each other. Finally, the resulting animation sequence should be analyzed for unexplained motion changes or large position changes, in order to fine tune the simulation parameters.

Numerical Stability

The integration method works stably if the accumulation of a derived function sufficiently approximates the function itself:

$$f(x) \approx I(x) = \sum_{x' = x_0}^{x} \dot{f}(x')$$

In order to achieve this, the step size h must be selected appropriately for the problem function $f(x)$. The error control selects a suitable value for h independently, to which the accuracy ε of the intermediate steps is applied. Since h only varies within a specified interval, if the lower bounds of the interval, corresponding to the minimal step size, is reached, this indicates that the accuracy specified can no longer be met. In this case, the sum $I(x)$ may diverge considerably from the nominal value $f(x)$. In extreme cases, this will lead to full separation, which will be noticeable in the simulation as sudden position changes of the affected object. This could deteriorate to the point that an object totally disappears when its position coordinates diverge rapidly toward infinity.

Sufficient Collision Detection

A timely recognition of collisions is crucial for processing collisions and physical contact. The collision step size Δt_{Col} represents the maximum time interval between two collision detection invocations. It should be selected based on the spatial dimensions of the simulated bodies and the maximum velocity.

Selecting too large a value for Δt_{Col} has many negative consequences:

- Bodies pass through each other because the time interval of their overlap is smaller than Δt_{Col}.
- The penetration of bodies is recognized too late, which leads to very large penetration depths Υ. This leads to enormous contact forces, which cause the objects to jump away from each other. Furthermore, this breaks up the smooth application of the contact forces, which in turn drives the step size of the integration method down to its minimum value.

Moderate Collision Classification

Using the analytical collision handling method, the system must first classify the intersection of two bodies as a collision or contact. This is accomplished via a comparison with the maximum contact velocity $\upsilon_{N\max}$. This velocity cannot be considered to be independent of the collision step size.

A body which rests upon an unmoving base should not be supported by collision forces, it should be supported by contact forces. The difference in velocity between two bodies after a collision is approximately $g \cdot \Delta t_{Col}$. Practical values for $\upsilon_{N\max}$ should be 10 to 100 times larger, in order to cover higher velocities as well. These could occur when forces in addition to gravity are applied to a body. Having too small a $\upsilon_{N\max}$ value leads to collisions during the settling process. In addition to extending the settling period, this also consumes more computing time due to the unused collision inertia calculations and the automatic restart of the integration method.

Appropriate Parameters for Body and Material

The modelling of contact forces together with the spring collision method and the connection types *spring*, *damper*, and *rod* result in a feedback control loop which must be kept stable. The principal parameters of this feedback control loop are the elasticity and damping factors of the materials and connections as well as the gravitational force of the bodies. Since the mass of a body comes from its physical size, a prudent selection of the material type is important. Radically different body masses should be avoided. Elasticity and damping factors should protect the settling process from the elastic behavior of the connections. This property is comparable to the effect of shock-absorbers in an automobile. The goal here is to have as short a settling period as possible and to avoid over reaction upon contact. Previously, figure 23 showed the settling period of a sphere on a plane using the penalty method and the spring method with the same elasticity factor.

Since the spring method only smooths collisions based on the collision value ε, slowly fading oscillations with high amplitudes can occur. For this reason, simulation of collision problems using high velocities should not use the spring method. Only by selecting larger material factors can the maximum penetration depth be contained. However, such “rigid” problems have large computational costs. Problems with radically different time factors in the differential equations are known as “rigid” differential equations. The are only solvable by using very fine time intervals. In contrast, the penalty method takes the difference in velocity at the collision point into account and thereby more quickly achieves a standstill without large oscillations.

The default values for the damping and spring constants were generated by analyzing the settling behavior of spheres from 1 to 600kg. A universal selection of the factors, which provides optimal results for every thinkable case is not possible. A simulation should therefore be built from bodies and factors that are suitable for each other. Bodies in the simulation which should not move can be tagged as massless. This saves processing time since these bodies are not added to the motion equations and collision detection between massless bodies is not required.

5. Sample Animations

5.1 Pendulum

Five rubber balls are connected to fixed points via rods as shown in figure 28. One of the outside balls is brought to the horizontal position so that it falls due to its weight in a circular path toward the other still balls. In this way, a system of five mathematical pendulums is constructed. The energy of the outer pendulums is propagated through collision impulses to the other pendulums. In order to beautify the simulation, a supporting stand can be built. The cylinders comprising the stand must be defined as immobile. In order to avoid changes in the length of the rods, one can increase the elasticity and damping constants by a factor of ten.

Figure 28: Step-by-step construction of a five sphere pendulum

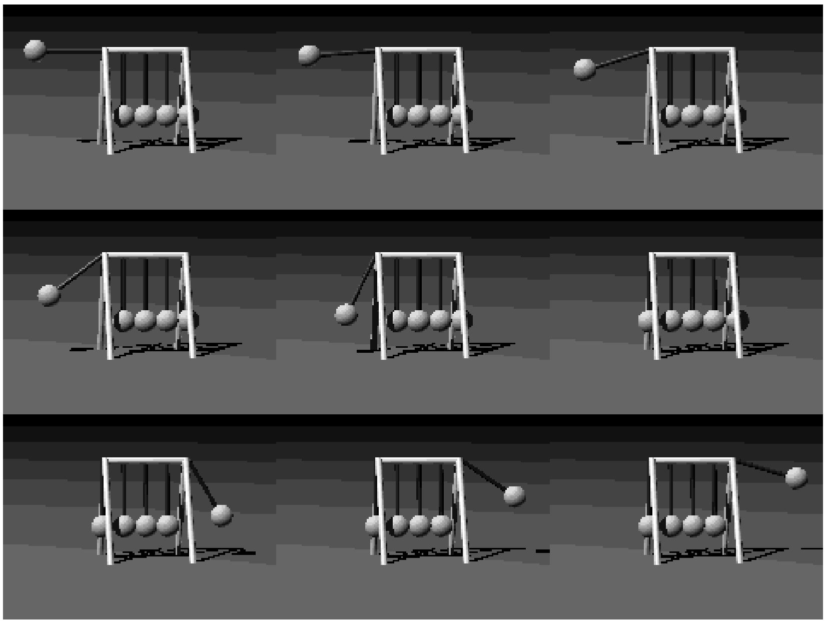

Figure 29: Animation sequence of the five sphere pendulum

After all bodies and connections have been entered, the animation can be started. Normally the user first selects a suitable frame frequency. For example, figure 29 shows an animation sequence that was produced with 15 frames per second. Every image frame is stored, so after recording, the whole animation sequence can be played forward and backwards without a large computational overhead. In this way, one can find the optimal camera position for viewing the sequence. Once the animation is set-up perfectly, the ray tracer code for each image frame is generated and the ray-traced images can be computed in batch mode.

5.2 Can Toss

The next example shows a can toss simulation well known from county fairs: six cans are set up in a pyramid shape and all of them must be knocked down with a single ball (see figure 30). It is interesting to see how realistically the cylindrical cans fly in all directions after being struck by the ball. Or expressed more mathematically: how the impulse of the ball is propagated to the previously still cylinders.

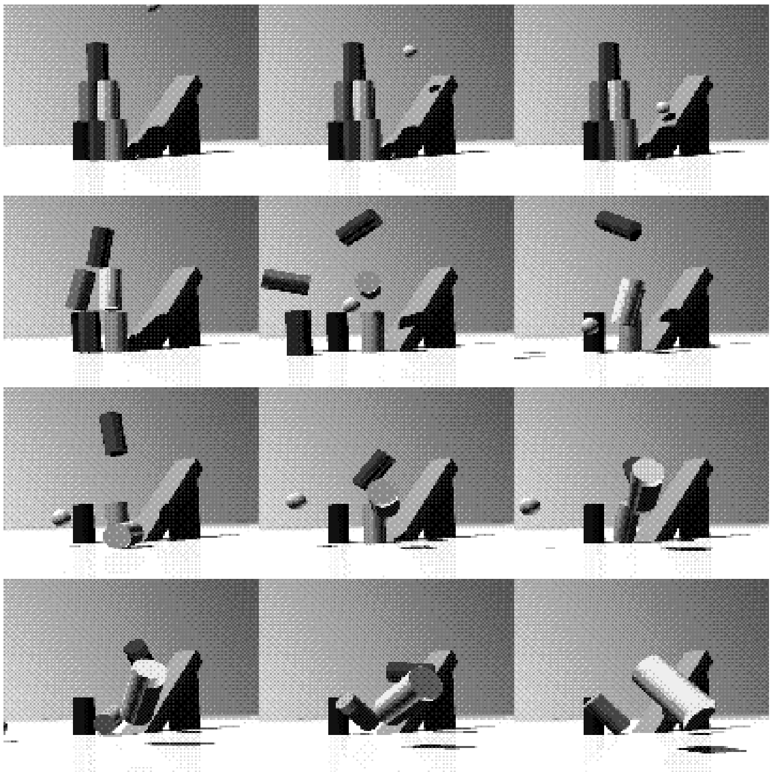

Figure 30: Can toss

5.3 Vehicle Simulation

This is an example of a longer sequence. Due to space restrictions, only a few images are shown here (approximately every 30th image). The original version of this sequence is approximately 500 image frames long and takes up 440MB of disk space in uncompressed format. Each image is 640×480 pixels large, and when shown with 25 frames a second, the sequence has a duration of about 20 seconds. This high playback speed can not be directly achieved on a workstation class computer. However, using a video recorder with single frame write capability or a laser disc system, one can write each image frame individually. Thereafter, the image can be played back at full speed. In the future, image compression algorithms such as MPEG and faster workstations could allow software playback at full speed *and* at full size.

The sequence (see figure 31) shows a three wheeled vehicle with shock absorbers and spherical wheels. The vehicle starts out standing on small elevated surface, which is implemented as a cuboid with a simple pendulum in the background. It is accelerated by inducing torque on the front wheel, so it rolls off the stairs (note the shock absorbers). The vehicle passes under a stylized tree, encounters a small hill, causing it to turn to the right onto an ice surface, where it receives an angular momentum and spins to a stop.

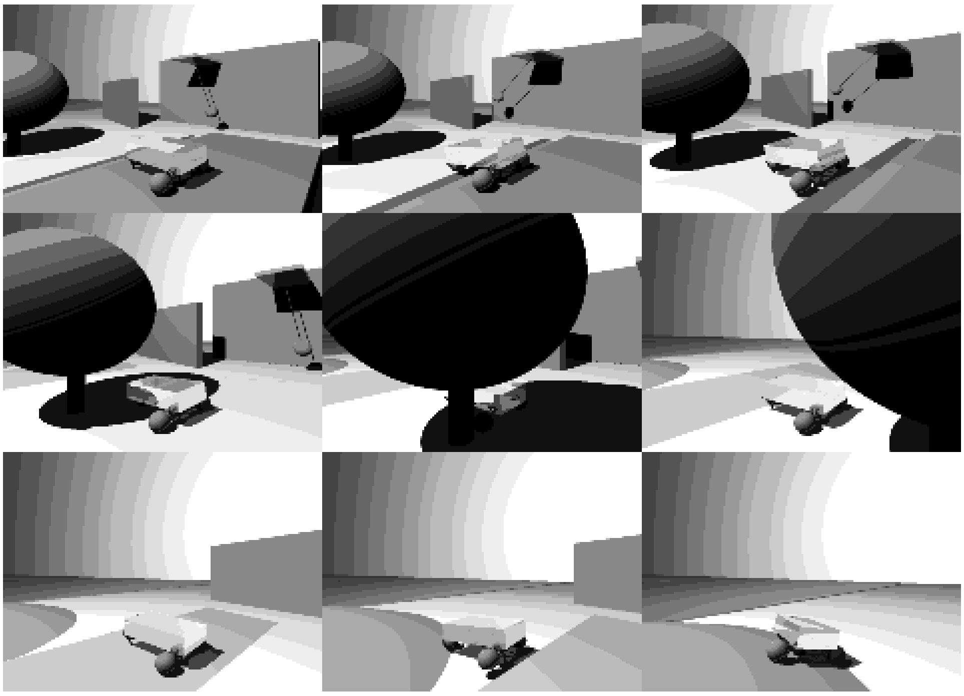

Figure 31: Vehicle on ice

6. Acknowledgements

In the beginning, the idea behind the *AERO* system was to create a simulation of the real world according to elementary physics. Particularly the articles from David Baraff provide convincing evidence of the feasibility of such a project. A collection of various articles came together quickly, in which each article very accurately takes individual physical effects into consideration. However, the sum of the articles did not provide a functioning model. It became apparent, that the well known formulas of mechanics are mainly directed toward special cases and introductory lectures on the subject are normally simplified to demonstrate the special cases. Universal analytical handling, in contrast, is only possible through the principles of mechanics or through approximations such as the penalty method.

We would like to thank Brian Blevins for translating the original German manuscript and Prof. Levi for his support. We also acknowledge the work of Drew Wells and colleagues for their public domain POV ("persistance of vision") ray tracing package, available via "anonymous ftp" from `alfred.ccs.carleton.ca` (134.117.1.1), as well as the work from Lawrence Rowe, Kevin Gong, Ketan Patel, Brian Smith, and Dan Wallach from the University of California at Berkeley for their public domain MPEG encoding and decoding package, available via "anonymous ftp" from `toe.cs.berkeley.edu` (128.32.149.117), directory `pub/multimedia/mpeg`. We used their systems for rendering frames and composing animation sequences.

The *AERO* system has been implemented in C/Unix with X-Windows and tested on Sun workstations and IBM-PC/linux systems by Andreas Ziegler (3D-scene editor), Hartmut Keller (3D-graphics, spatial representation, scene generation for ray tracing) and Horst Stolz (simulation computation) under the direction of Thomas Bräunl. The

AERO system is also available as public domain software via "anomymous ftp" and may be copied from our server: `ftp.informatik.uni-stuttgart.de` (currently 129.69.211.2) in subdirectory: `pub/AERO` .

7. References

[Barzel et al. 88] Ronen Barzel, Alan H. Barr: *A Modeling System Based on Dynamic Constraints*; pp. 179-187 in Computer Graphics (Proceedings of SIGGRAPH'88), ACM Press, Atlanta, vol. 22, no. 4, August 1988

[Baraff 91] David Baraff: *Coping with Friction for Non-penetrating Rigid Body Simulation*; pp. 31-40, in Computer Graphics (Proceedings of SIGGRAPH '91), ACM Press, Las Vegas, vol. 25, no. 4, July 1991

[Dormand et al. 80] J. R. Dormand, P. J. Prince: *A Family of Imbedded Runge-Kutta Formulae*; pp. 19-27 in Journal of Computational and Applied Mathematics, Elsevier Science Publishers B.V., North-Holland, vol. 6, no. 1, 1980

[Dormand et al. 81] J. R. Dormand, P. J. Prince: *High Order Embedded Runge-Kutta Formulae*; pp. 203-211 in Journal of Computational and Applied Mathematics, Elsevier Science Publishers B.V., North-Holland, vol. 7, no. 1, 1981

[Dormand et al. 86] J. R. Dormand, P. J. Prince: *A Reconsideration of some Embedded Runge-Kutta Formulae*; pp. 203-211 in Journal of Computational and Applied Mathematics, Elsevier Science Publishers B.V., North-Holland, vol. 15, 1986

[Haug 84] E. J. Haug (Ed.): *Computer Aided Analysis and Optimization of Mechanical System Dynamics*; NATO ASI Series, Series F - Computer and System Science, vol. 9, Springer Verlag, Berlin, Heidelberg, 1984

[Heinzel 92] Gerhard Heinzel: *Beliebig genau -- Moderne Runga-Kutta-Verfahren zur Lösung von Differentialgleichungen*; pp. 172-185 in c't, Verlag Heinz Heise GmbH & Co KG, Hannover, no. 8, 1992

[Isaacs 87] Paul M. Isaacs, Michaell F. Cohen: *Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions and Inverse Dynamics*; pp.215-224 in Computer Graphics (Proceedings of SIGGRAPH'87), ACM Press, Anaheim vol. 21, no. 4, July 1987

[MacMillan 60] William D. MacMillan: *Dynamics of Rigid Bodies*; Dover Publications, New York, 1960

[Moore et al. 88] Matthew Moore, Jane Wilhelms: *Collision Detection and Response for Computer Animation*; pp. 289-298 in Computer Graphics (Proceedings of SIGGRAPH '88), ACM Press, Atlanta, vol. 22, no. 4, August 1988

[Murty 88] K. G. Murty: *Linear Complementary, Linear and Nonlinear Programming*; Heldermann-Verlag, Berlin, 1988

[Shoemake 85] Ken Shoemake: Animating Rotation with Quaternion Curves; pp. 245-254 in Proceedings of SIGGRAPH'85, ACM Press, San Francisco, July 1985

[Wang et al. 92] Yu Wang, Matther T. Mason: *Two-Dimensional Rigid-Body Collision with Friction*; pp. 635-642 in Journal of Applied Mechanics, vol. 59, September 1992

[Witkin 90] Andrew Witkin, Micheal Gleicher, William Welch: *Interactive Dynamics*; pp. 11-21 in Computer Graphics, ACM Press, 1990

[Wittenburg 77] Jens Wittenburg: *Dynamics of Systems of Rigid Bodies*; Teubner, Stuttgart, 1977