An impulse-based dynamic simulation system for VR applications

Jan Bender, Dieter Finkenzeller, Alfred Schmitt

Universität Karlsruhe
Institut für Betriebs- und Dialogsysteme
Am Fasanengarten 5
76128 Karlsruhe
Germany
Phone: +49 (0)721 608 3965
Fax: +49 (0)721 608 8330
E-mail: {jbender, dfinken, aschmitt}@ira.uka.de

Abstract: This paper describes a system for dynamic simulation of linked rigid bodies in real-time. The system was developed to simulate mechanical behaviour in VR applications. An extension for a 3D modelling tool was developed which provides the possibility to model a VR scene including the geometries and mechanical parameters of all rigid bodies and the properties of the joints between them easily. For the dynamic simulation an impulse-based method is used. The distinguishing feature of this method is that all kind of constraints are satisfied with the iterative computation of impulses. The advantage of this iterative technique is that it is fast and accurate results can be achieved. The dynamic simulation system uses efficient collision detection methods. For every collision that is detected a contact region between the objects is determined to provide an accurate collision response.

Key words: VR-system, dynamic simulation, impulse-based method, multi body systems, mechanical behaviour

1- Introduction

Realistic mechanical behaviour in virtual reality applications becomes more and more important because it provides a higher degree of immersion to the user. A VR application with realistic mechanical behaviour has to simulate dynamic effects, to detect collisions and to resolve them with frictional forces in real-time.

For the dynamic simulation of a VR scene a description of the whole scene is needed. This description must include the geometries and properties of all rigid bodies as well as the constraints between them. A 3D modelling tool is needed to create a description of the geometries in a VR scene. In this work Maya [1] was used for modelling. In Maya it is already possible to describe the properties of rigid bodies. Just for the description of the constraints between them an extension of Maya had to be written in MEL (Maya Embedded Language). So now the complete description of a VR scene with all necessary parameters for the dynamic simulation can be created with this modelling tool.

In this work two types of constraints are used: joint and non-penetration constraints. There are many different kinds of joints, for example ball joints, hinge joints, servo motors etc. A joint links two rigid bodies together by constraints for certain points fixed to these rigid bodies and for their velocities. The dynamic simulation method has to guarantee that these constraints are satisfied in every simulation step. Interpenetration of two rigid bodies has to be detected and resolved to satisfy the non-penetration constraints.

The dynamic simulation system that is presented in this paper satisfies both kinds of constraints by computing impulses that simulate the forces acting on the rigid bodies. The use of impulses is possible because the simulation makes discrete time steps. Exactly the same change of velocity that a continuous force causes in one time step can be achieved by computing an impulse that changes the velocity at once. Many other simulation systems compute continuous forces to satisfy the constraints for joints and impulses for the collision response. The advantage of the presented method is that all constraints can be solved in a uniform way.

The outline of this paper is as follows. In the next section a short overview of dynamic simulation systems and collision detection and collision response methods is given. Afterwards the architecture of the presented simulation system is introduced. In the fourth section the dynamic simulation method, the collision detection and the collision response are described in detail. The succeeding section contains results achieved with the presented simulation system. This paper ends with a short conclusion and an outlook for future work.

2- Related Works

The "Open Dynamics Engine" [2] of Russel Smith is a free library for the dynamic simulation of rigid bodies with joint and non-penetration constraints. Brian Mirtich and John Canny [3][4] describe an impulse-based dynamic simulation of rigid bodies. In their simulator "Impulse" they use forces to solve joint constraints and impulses for the collision response with friction. David Baraff [5] researches in his work mainly the non-penetration constraints and compares his results with the penalty method. Smith, Mirtich and Baraff all use the friction model of Coulomb for their collision response.

For the collision detection and response in a dynamic simulation system it is necessary to determine the closest points of two rigid bodies. There are two very fast methods to solve this problem: the algorithm from Gilbert, Johnson and Keerthi [6] (GJK) and the one from Lin and Canny [7]. The GJK algorithm uses the Minkowski sum of two polytopes to compute the distance between them. It is even able to compute the penetration depth of two convex objects. A robust implementation of this algorithm was written by Van den Bergen [8]. The closest feature algorithm of Lin and Canny is not able to handle interpenetration of objects. It works with voronoi regions. Brian Mirtich has written an improved version of this algorithm. His implementation "V-Clip" [9] is more robust and can handle penetration.

One of the main problems of a collision response method is the handling of resting contacts. Brian Mirtich [10] uses a velocity threshold in his work to determine if the system must handle a collision or a resting contact. In the case of a resting contact an impulse is computed that reverses the relative velocity of the contact points. By applying this impulse interpenetration is prevented. Guendelman et al. [11] change the order of a simulation step to handle resting contacts. First the velocities of all rigid bodies are updated, then the contacts are processed and at last the positions of the bodies are updated. The new order of the simulation step prevents bodies with resting contacts from bouncing.

3- System architecture

The dynamic simulation system consists of two parts. One part is the extension for the 3D modelling tool Maya to create a VR scene including all parameters for the dynamic simulation. The other part is the dynamic simulator.

3.1 - The modelling tool

The 3D modelling tool Maya has an own dynamic simulation system integrated. So it is already possible to define rigid bodies with all parameters in Maya. This can be used for the modelling of a VR scene with dynamics. The dynamic simulator in Maya only supports four different types of joint constraints: spherical joints, hinge joints, spring joints and a barrier constraint. The joint definitions of Maya cannot be used for the simulator in this work because more types of joints are needed. In Maya the geometry of a rigid body is used for the graphical output and the collision detection. In the dynamic simulation of this work a rigid body can have several geometries for the graphical output and several geometries for the collision detection. The geometries for the graphical output and the collision detection do not need to be the same. Maya provides the possibility to add own attributes to every geometry that is created.

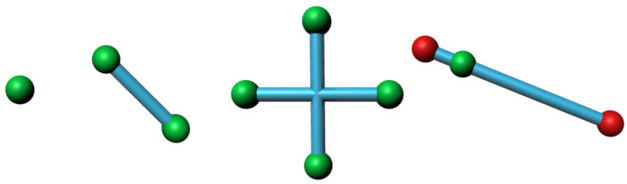

Figure 1: Spherical joint, hinge joint, cardan joint, slider joint

Every object that is created while modelling a VR scene for the dynamic simulator of this work gets an additional attribute with the type of this object, for example rigid body, collision geometry, spherical joint etc. If a rigid body with several geometries for the graphical output and for the collision detection should be created then first a rigid body must be built. This rigid body already has a geometry that will be used for the computation of the inertia tensor, for graphical output and for collision detection if there are no other collision geometries defined for this body. Afterwards all additional geometries can be created as children of the rigid body node in the scene graph of Maya. A joint needs at least two additional attributes for the two rigid bodies that it links together. Some types of joints need more attributes, for example for the torque of a servo motor. If the user adds a joint to the VR scene, a polygon primitive is added to the scene which enables him to define the positions of the joint points very easily. The polygon primitives for some kinds of joints are shown in figure 1.

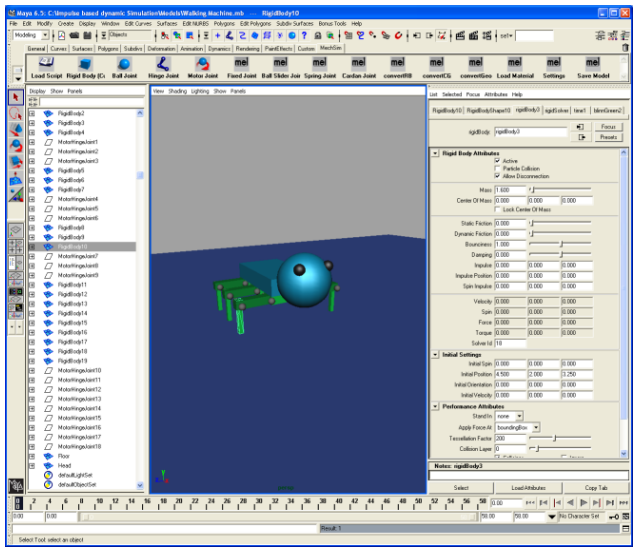

Figure 2: Modelling a VR scene with Maya

A complete VR scene modelled with Maya is shown in figure 2. After modelling the description of this scene can be exported in a XML-file. This file contains all geometries, joints and parameters for the dynamic simulation. Now the simulator can import the XML-file and start with the simulation.

3.2 – The dynamic simulator

The concept of multi-threading was used for the design of the simulation system to provide the possibility to run the dynamic simulation parallel to a VR application. The dynamic simulation runs in an own thread. In this way it is independent from the thread for the graphical user interface. The properties of the dynamic simulation system and the parameters of the model that is simulated must not be changed and the simulation data must not be read during a simulation step to guarantee thread safety. The dynamic simulator has a command queue. If a property of the simulator or the model should be changed by another thread, then this thread must put a command with the id of the property and the new value in the queue. Before every simulation step all commands in the queue are executed and deleted. The multi-threading of the simulation system is shown in figure 3.

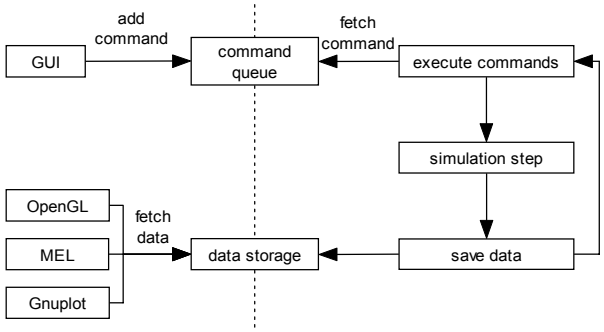

Figure 3: Separation of GUI and dynamic simulation in two different threads

After every simulation step the changed data, like for example the positions of the rigid bodies, is copied in a data storage. Other threads are allowed to use the data from this storage, for example for the graphical output.

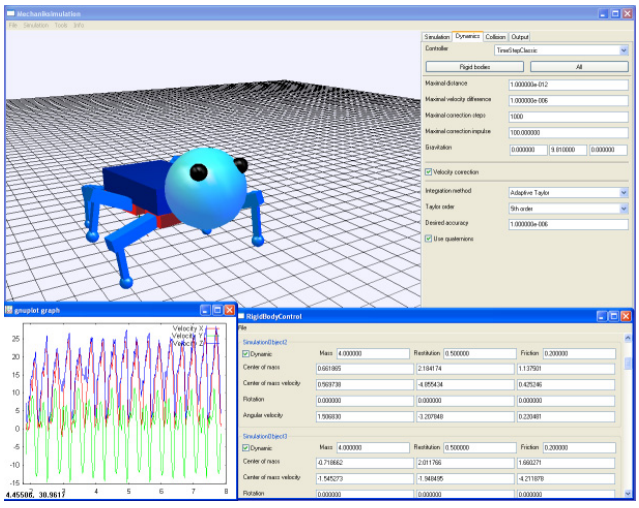

Figure 4: Test environment of the dynamic simulator

For the development of the dynamic simulator a test environment (see figure 4) has been implemented. This test environment uses the same access to the dynamic simulator as a VR application. In this environment the parameters of the simulator and of every object in the simulated VR scene can be changed at run-time. This is necessary to analyze the behaviour of the simulation method with different settings. The user can also interact with the objects by applying impulses with a simple mouse click. The test environment provides three output possibilities for the simulation data. First there is an OpenGL output for real-time visualization. Then the data can be used for plotting with the program Gnuplot [12] and the last possibility is an output as a MEL script which contains all geometries and movements of the bodies. This script file can be executed in Maya and generates an animation sequence of the dynamic simulation which can be rendered to get a photorealistic video.

4- Dynamic simulation

Every rigid body needs six parameters for the dynamic simulation. For the computation of the translational movement of a rigid body its mass m, its centre of mass $C(t)$ and its velocity $v(t)$ are needed. The rotational movement is computed with the inertia tensor J in body-space coordinates, a unit quaternion $q(t)$ that describes the rotation of the body [13] and the angular velocity $\omega(t)$. The inertia tensor can be computed with the mass and the geometry of the rigid body [14]. The advantage of using a unit quaternion instead of a rotation matrix is that the numerical error can be reduced [15]. A quaternion has just four parameters to describe the three degrees of freedom of the rotation whereas a rotation matrix needs nine parameters. Because of this a solver for ordinary differential equations (ODE) which is needed for the dynamic simulation produces a greater numerical error if rotation matrices are used. The numerical error in a rotation matrix has the effect that the vectors of the coordinate system that is represented by the matrix are not orthonormal anymore. A quaternion just looses its unit length which can be easily corrected by renormalizing it.

In the next section the dynamic simulation of an unconstrained rigid body will be described. The following section will explain how joint constraints can be satisfied with impulses. In the last two sections of this chapter the collision detection and collision response which are needed to satisfy non-penetration constraints will be described.

4.1 – Simulation step of an unconstrained rigid body

For a dynamic simulation step of a rigid body without any constraints the parameters of this body at time t_0 must be known and a time step size h must be given. Then the parameters for the time $t_0 + h$ can be computed. The mass and the inertia tensor in body-space coordinates of a rigid body are constant over time so just the other four parameters have to be updated. If gravitation is the only external force that acts on the rigid bodies then the acceleration due to gravity g changes the velocity v and the centre of mass C as follows:

$$v(t_0 + h) = v(t_0) + \int_0^h a(t)\,dt = v(t_0) + g \cdot h \qquad (1)$$

$$C(t_0+h) = C(t_0) + \int_0^h v(t_0) + g \cdot t \, dt = C(t_0) + v(t_0) \cdot h + \frac{1}{2} g \cdot h^2 \quad (2)$$

The angular velocity ω for time $t_0 + h$ can be computed by solving the following differential equation (in body-space coordinates):

$$\omega'(t) = J^{-1} \cdot (\omega(t) \times (J \cdot \omega(t))) \quad (3)$$

The last step is to compute the unit quaternion q for the time $t_0 + h$. To get $q(t_0 + h)$ the following differential equation has to be solved:

$$q'(t) = \frac{1}{2} \tilde{\omega}(t) \cdot q(t) \quad (4)$$

where $\tilde{\omega}(t)$ is the quaternion $[0, \omega_x(t), \omega_y(t), \omega_z(t)]$.

Equation (3) is solved with the Taylor series method. Therefore the first n derivatives are computed with equation (5) to get an error of the order $O(h^{n+1})$.

$$\omega^{(i+1)}(t) = \sum_{k=0}^{i} J^{-1} \cdot \binom{i}{k} (\omega^{(k)}(t) \times (J\omega^{(i-k)}(t))) \quad (5)$$

Using the derivatives the new angular velocity can be computed with:

$$\omega(t_0 + h) = \sum_{k=0}^{n} \omega^{(k)}(t_0) \cdot \frac{h^{(k)}}{k!} + O(h^{n+1}) \quad (6)$$

After the first n derivatives of $\omega(t)$ are computed the first n derivatives of the unit quaternion *q(t)* can be determined:

$$q^{(i+1)}(t) = \frac{1}{2} \sum_{k=0}^{i} \binom{i}{k} (\omega^{(k)}(t) \cdot q^{(i-k)}(t)) \quad (7)$$

Now the quaternion $q(t_0 + h)$ can be computed with the Taylor series of order n:

$$q(t_0 + h) = \sum_{k=0}^{n} q^{(k)}(t_0) \cdot \frac{h^{(k)}}{k!} + O(h^{n+1}) \quad (8)$$

If a desired accuracy $\phi_{desired}$ should be achieved from the ODE solver an adaptive time step size is needed. For the computation of a new time step size an estimation of the error made during a time step is needed. The error made when computing the quaternion $q(t_0 + h)$ with the Taylor series method of order n can be estimated by the magnitude of the *(n+1)* th element of the Taylor series:

$$\phi_{error} := \left| q^{(n+1)}(t_0) \cdot \frac{h^{(n+1)}}{(n+1)!} \right| \quad (9)$$

Exactly as Press et al. did for an embedded Runge-Kutta method [16] a new time step size can be computed as follows:

$$h_{new} = h \left(\frac{\phi_{desired}}{\phi_{error}} \right)^{\frac{1}{n}} \quad (10)$$

If ϕ_{error} is larger than $\phi_{desired}$ then the simulation step must be repeated and this time the new step size is used. Otherwise the simulation can continue and the new step size is used for the next simulation step.

4.2 – Joint constraints

In this section will be described how to satisfy joint constraints using impulses. The method will be explained for a spherical joint. Other joint types can be handled in a similar way.

If two rigid bodies are linked with a spherical joint, then a point *A* in the first rigid body is connected to a point *B* in the second one. In each simulation step the two points should have the same position and the same velocity, so the two constraints of this joint are:

$$\begin{aligned} |A(t) - B(t)| &\le \varepsilon_{pos} \\ |u_A(t) - u_B(t)| &\le \varepsilon_{vel} \end{aligned} \quad (11)$$

where ε_{pos} and ε_{vel} are tolerance values. If both constraints are satisfied for time t_0 and a simulation step as in section 4.1 is made, then both constraints are generally unsatisfied for time $t_0 + h$. Now impulses have to be computed to correct this.

4.2.1 – Position constraint

First an impulse p must be found to satisfy the point position constraint. The impulse p will be applied at point *A* to the first rigid body and the impulse $-p$ at point *B* to the second rigid body. These impulses must eliminate the distance between the two points at time $t_0 + h$. They do not have any influence on the energy of the system because they have opposite directions. The positions of the two points at time $t_0 + h$ can be determined by first solving the following differential equation for the vector $r(t_0) = P(t_0) - C(t_0)$ which points from the centre of mass to the respective point:

$$r'(t) = \omega(t) \times r(t) \quad (12)$$

This equation can be solved with the Taylor series method from section 4.1. Then the new position of the point is given by the sum of $r(t_0 + h)$ and $C(t_0 + h)$ which can be determined with equation (2). Now the distance vector d

between the two points at time $t_0 + h$ can be determined:

$$d(t_0 + h) := B(t_0 + h) - A(t_0 + h) \quad (13)$$

The following matrix has the property that the result of multiplying it to an impulse is the velocity change this impulse causes:

$$K := \frac{1}{m} \cdot I_3 - \tilde{r} \cdot J^{-1} \cdot \tilde{r} \quad (14)$$

where I_3 is the three-dimensional identity matrix and $\tilde{r}$ is the cross-product matrix corresponding to *r*. An impulse *p* that eliminates the distance *d* in a time step of length *h* must change the relative velocity of the two points by d/h. This impulse must satisfy the following equation:

$$\frac{d(t_0 + h)}{h} = K_1 \cdot p - K_2 \cdot (-p) = (K_1 + K_2) \cdot p \quad (15)$$

The matrix $(K_1 + K_2)$ is constant, non-singular, symmetric and positive definite [10] so the impulse *p* can be determined by inverting this matrix. The point position constraint can be satisfied by applying this impulse at time t_0.

This method works well for one joint but not for a chain with several joints as shown in figure 5. The constraint of the left joint of the pendulum is satisfied but not the one of the right joint. If an impulse is computed and applied for the right joint its constraint will be satisfied but the left joint will break. In this case an impulse for every joint is computed in an iterative loop. In every iteration step the distances between the joint points get smaller. The loop terminates if every constraint is satisfied for a certain tolerance ε_{pos}.

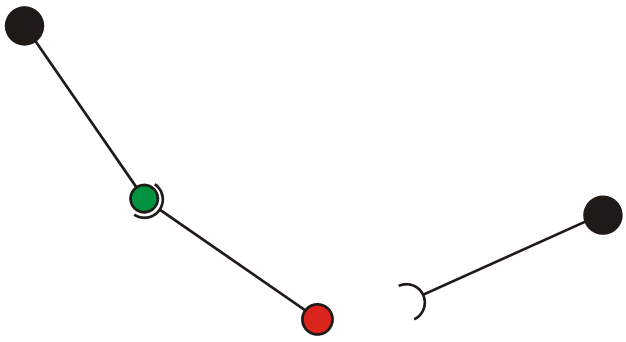

Figure 5: Pendulum with two spherical joints

With the iterative method it is possible to satisfy all position constraints in a multi body system. The computed impulses simulate the inner forces in the joints.

4.2.2 – Velocity constraint

After computing an impulse for every position constraint and applying it to the rigid bodies, the velocities of all bodies change. Now a simulation step like in section 4.1 is made. The new velocities have the effect that all position constraints are satisfied after the simulation step but now in general the velocity constraints of the joints are not satisfied anymore. This can be corrected with a very similar method like the one of section 4.2.1.

First the velocity difference of the two joint points *A* and *B* must be determined. The velocity u_p of a point in a rigid body can be computed with the following equation:

$$u_P(t) = v(t) + \omega(t) \times (P(t) - C(t)) \quad (16)$$

With the velocity difference $\Delta u(t) := u_B(t) - u_A(t)$ and the matrix $K := (K_1 + K_2)$ from the last section an impulse *p* can be computed which eliminates the velocity difference if it is applied to the points in opposite directions:

$$p := K^{-1} \cdot \Delta u(t) \quad (17)$$

For every joint an impulse is computed in an iterative loop with this equation until every velocity constraint is satisfied for a certain tolerance ε_{vel}.

4.3 – Collision detection

If the multi body system which is simulated has non-penetration constraints, a collision detection method is needed to find out if bodies are interpenetrating or colliding. For the collision detection every rigid body has at least one collision geometry. This geometry can be different from the geometry used for the graphical output and for the computation of the inertia tensor. If a rigid body has a very complex geometry with thousands of triangles it can be advantageous to use a collision geometry of lower detail to increase the speed of the simulation. The collision detection method that is used in this work can only handle convex polyhedra. If a non-convex object is simulated, the collision geometry must be split into several convex parts.

It makes no sense to test collision geometries of the same rigid body for contacts or interpenetration. A collision test between static bodies is also unnecessary and just consumes time for computation. In these two cases the system can decide automatically not to make a collision test. Sometimes it is very useful if the modeller of a VR scene can decide which pairs of bodies are tested for collisions, for example if a pair of bodies will never have contact because of certain constraints. Because of this the collision detection has a table that defines which pairs of collision objects will be tested for contact or interpenetration.

Figure 6 shows a simulation step with collision detection. First the translation and rotation of all collision objects must be updated. Afterwards the collision detection must find all contact points and normals between the bodies. Then a simulation step with collision response can be executed. After the simulation step the system must check if there is any penetration between two bodies. In case of a penetration the last simulation step must be undone and the point of time t_c when the respective bodies just had contact must be

determined. This time of contact t_c can be determined with a binary search method. The difference between the time of contact and the time before the simulation step gives the time step size h_c. With the new time step size the simulation step will be executed again. Now the bodies which were interpenetrating before will just have contact.

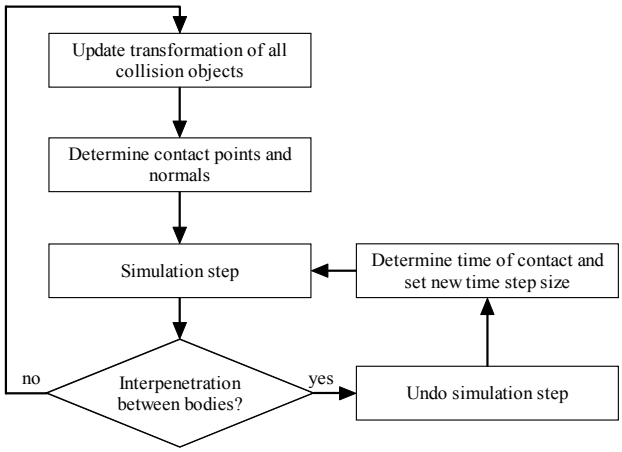

Figure 6: Simulation step with collision detection

The collision detection of the simulation system works in three phases. The first phase reduces the collision tests to speed up the collision detection. Every body has an axis-aligned bounding box with dynamic size. A bounding box test is a very fast method to find all rigid bodies that are close enough for a collision. In every simulation step a fast sweep and prune method determines all overlapping bounding boxes by checking whether the intervals on the x-, y- and z-axis of the bounding boxes overlap. This method uses insertion sort and caches the sorted lists of the last simulation step. Because of the spatial and temporal coherence of the dynamic simulation the sweep and prune method runs in linear time. For each pair of overlapping boxes the collision test continues with the second phase.

In the second phase the closest features of every collision pair that was found in the first phase are computed. A feature is a vertex, an edge or a face. For the determination of the closest features V-Clip [9] is used which makes use of voronoi regions. V-Clip also computes the distance, the closest points and the contact normal of these features. If this distance is smaller than a certain tolerance value $\varepsilon_{collision}$ but greater than zero a collision is found. Otherwise if the distance is smaller than zero the bodies interpenetrate and a binary search must start to find the time of contact t_c. If the distance is greater than $\varepsilon_{collision}$ the bodies are too far apart for a collision.

In the second phase the closest features f_1 and f_2, the closest contact points P_1 and P_2 and the contact normal n have been determined. In the third phase a test for multiple contact points for each collision is performed. In this phase the system searches for features that have a distance which is greater or equal than the distance of the closest features and smaller than the tolerance value $\varepsilon_{collision}$. If such features are found the closest points of these features are additional contact points. All contacts between two bodies have the same contact normal n. This normal points from the second body to the first one. The contact points of each body build a contact polygon. Figure 7 shows an example of such a contact polygon.

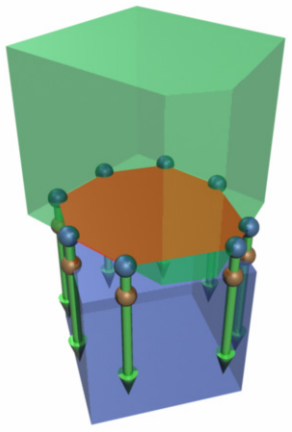

Figure 7: Two rigid bodies with multiple contact points, the contact polygon of the second body is marked red

In the following it will be explained how the features with additional contact points for a collision pair can be determined. The contact polygon of each body lies in the same plane as a face of this body and as its closest feature which was computed in the second phase. So firstly these two faces must be determined.

As the closest feature lies in the same plane as the searched face, the face must be the closest feature itself or one of the neighbouring faces. After the candidates for the two searched faces have been determined, the dot product of the face normal of each candidate of the first body with the face normal of each candidate of the second body is build. The minimum of the dot products belongs to the searched pair of faces because the smallest angle is between these faces. The two faces and all neighbouring vertices and edges are candidates for features with additional contact points.

If one of the candidate features of the first body lies in the voronoi region of a candidate feature of the second body and the distance between these two features is smaller than the collision tolerance value $\varepsilon_{collision}$ then a new contact is found. The contact points of this new contact are the closest points of the two features. The contact normal is the normal of the contact polygon of the second body. With this method all additional contact points can be determined.

4.4 – Collision response

The method presented in the paper of Guendelman et al. [11] was implemented at first for the collision response in the dynamic simulator. This method works well for single rigid bodies but it is hard to combine with the method for resolving joint constraints. Because of this a new method is developed at the moment which is based on the hypothesis of Poisson and on the Coulomb friction model. The hypothesis of Poisson gives an approximation for the collision impulse of two bodies in the direction of the contact normal and the

Coulomb friction model is an approximation for the friction forces that act in the opposite direction of the tangential relative point velocities of the contact point pairs.

The new method for collision response differentiates between three cases. If the dot product of the contact normal with the relative point velocity $u_1(t)-u_2(t)$ of the contact points is greater than zero then the points are separating and no impulse needs to be computed. If this product is smaller than zero but greater than a tolerance value $-\varepsilon_{contact}$ then the contact points build a resting contact. Otherwise a real collision has occurred.

In the last two cases a collision impulse has to be computed. If p_n^- is the impulse that makes the relative velocity of the contact points in normal direction to zero, then the impulse for collision response in normal direction p_n^+ is computed with the following equation (hypothesis of Poisson):

$$p_n^+ = (1+e)\cdot p_n^- \qquad (18)$$

where e is the coefficient of restitution. This coefficient is a constant between zero and one depending on the materials of the two colliding bodies. If the coefficient of restitution is one, then the collision is totally elastic and if it is zero, the collision is totally plastic.

To determine the impulse p_n^- first the relative velocity of the two contact points in normal direction n must be computed with the following equation:

$$\Delta u_n(t) = ((u_2(t)-u_1(t))\cdot n)\cdot n \qquad (19)$$

Afterwards the impulse can be determined with the following formula:

$$p_n := \frac{1}{n^T\cdot K\cdot n}\cdot \Delta u_n(t) \qquad (20)$$

This works for collisions with one contact point but if there are multiple contact points the impulse for each pair of contact points influences the relative point velocities of all other pairs of points. This problem can be solved by iteratively computing the impulses p_n^-. In every iteration step an impulse is computed for each pair of contact points until all relative point velocities are smaller than a certain tolerance value. This method works for a pair of rigid bodies that have multiple contact points and that are not linked to other rigid bodies with joints. If a non-penetration constraint and a joint constraint exist for a rigid body then the iteration loop can be combined with the one from section 4.2.2 to find a correct impulse p_n^- for each contact point. So in every iteration step first an impulse for each contact point pair is computed with equation (20) and then the velocity constraint for each joint must be satisfied by computing an impulse with equation (17). So all bodies of a chain have influence on the collision impulse p_n^-. With this method a collision response for linked rigid bodies is possible.

The computation of friction impulses using the Coulomb friction model has not been implemented yet but this will be done soon.

5- Results

In this section the results of the dynamic simulation of three different models are presented. The first simulation was made to measure the performance of the impulse-based method. Then with the next model the speed of the collision detection will be determined. The last simulation was made to take a look at the accuracy of the collision response method with friction that was already implemented. All simulations were performed on a PC with the following configuration: Intel Pentium 4 with 3.4 GHz, 1024 MB DDR2 memory and nVidia GeForce 6600 GT with 128 MB GDDR3 memory.

The first model that was simulated is a chain of eight rigid bodies which are linked with spherical joints (see figure 8). The spherical joints at both ends of the chain are fixed so the chain will not fall down.

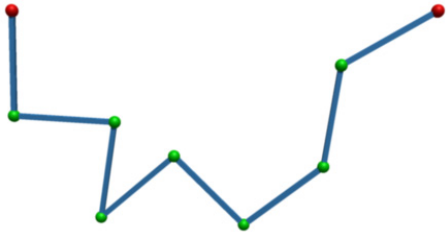

Figure 8: Chain of rigid bodies

The simulation was performed without collision handling to measure only the time that is needed by the impulse-based method. A fixed time step size of h=0.01 s was used. The tolerance values were set to $\varepsilon_{pos} = 10^{-12}$ m and $\varepsilon_{vel} = 10^{-12}$ m to achieve a very high accuracy. The Taylor method of order five was used to solve the differential equations. The simulation had a length of ten seconds, so one thousand simulation steps were performed.

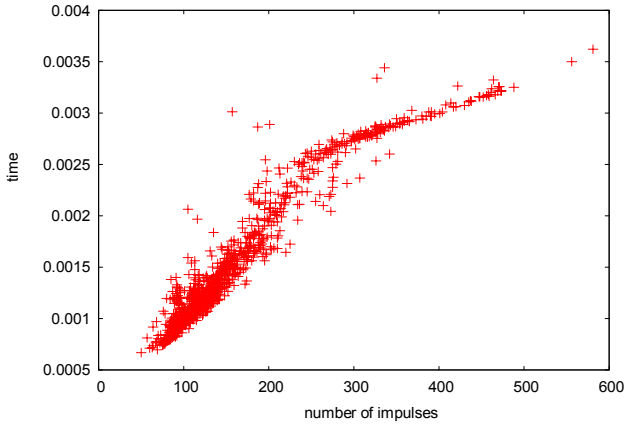

Figure 9: Simulation performance

For each step the number of impulses that had to be

computed for satisfying the position and velocity constraints of all joints was counted and the time needed for the whole step was measured. The results are shown in figure 9. The simulation had to compute between 50 and 581 impulses in every step. The mean value was 162. The impulse-based method needed between 0,000668769 s and 0,003621727 s for a whole simulation step. The average time was 0,001537897 s. Even in the worst case the simulation was about three times faster than real-time. In the average the simulation was more than six times faster than real-time. With higher tolerance values the simulation can run even faster.

The second model was build to measure the performance of the collision detection. The model consists of a grid of cubes which fall down on a fixed plane. In this model up to one thousand cubes can have contact with each other. Each contact can consist of several contact points. During the simulation the time needed for all three phases of the collision detection was measured. The results are shown in figure 10.

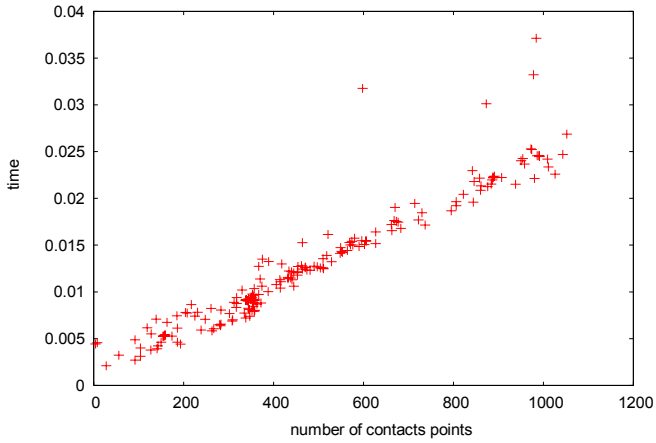

Figure 10: Time needed for collision detection

The figure shows that the time needed for the collision detection increases nearly linearly as the number of contact points increases which can be explained easily. In the first phase of the collision detection a sweep and prune method which runs in linear time is used. The closest features algorithm of the second phase exploits the spatial and temporal coherence of the dynamic simulation and therefore needs only constant time for each collision pair. So the second phase needs linear time to process all collision pairs. The last phase needs constant time for each collision pair to determine the collision polygon. Altogether the collision detection needs linear time. The mean value of contact points that occurred was 484 and the average time needed for the collision detection was 0,012924245 s. A normal VR scene consists of far less than one thousand rigid bodies but even for this amount the collision detection method is fast enough for a simulation in real-time.

The last simulation was made to take a look at the accuracy of the collision response method. The model consists of an inclined plane with an angle of 22.5 degree and ten cubes which are sliding down the plane with different friction coefficients. The stopping distance of every cube can be computed. After the simulation the computed value is compared to the simulated stopping distance. The model is shown in figure 11. The transparent cubes in the picture mark the computed points where the simulated red cubes are expected to stop.

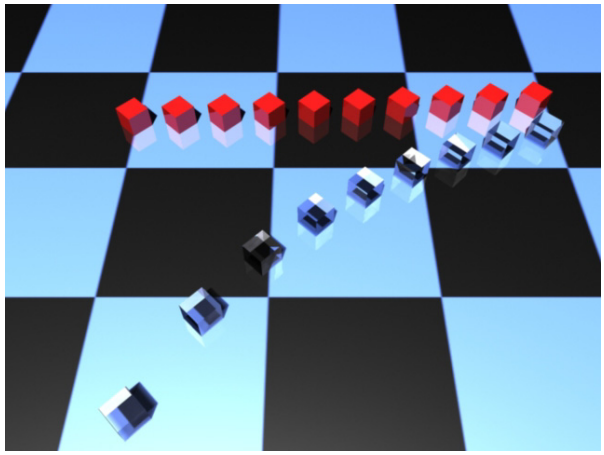

Figure 11: Cubes on an inclined plane

The time step size used for the simulation was h=0.01 s and for the collision detection a tolerance value of $\varepsilon_{collision} = 0.00001$ m was used. The following table shows the results of this simulation.

friction	simulated stopping distance	computed stopping distance	difference
0.535	25.70322	25.70234	0.00088
0.57	19.37585	19.37792	-0.00207
0.605	15.72423	15.71905	0.00518
0.64	13.12114	13.11542	0.00572
0.675	11.11596	11.11587	0.00009
0.71	10.37414	10.37604	-0.00189
0.745	9.54625	9.54379	0.00247
0.78	8.7845	8.78449	0.00001
0.815	7.55768	7.55506	0.00262
0.85	6.48953	6.48689	0.00264

Table 1: Stopping distances

The simulation of this model needed about nine seconds until all cubes stopped. In the worst case the difference between the simulated and the computed stopping distance was 0.000572 m. The velocities of the decelerating cubes are shown in figure 12. This figure shows how the velocities of the rigid bodies get linearly smaller until they reach zero. The accuracy of the collision response depends on the tolerance value of the collision detection and on the used time step size. By using a smaller time step size higher accuracy can be reached.

The results in this section show that the introduced impulse-based method is fast and accurate enough for the use in a VR application. The VR system VISUM [17] already uses the dynamic simulation system of this work for the simulation of mechatronic systems. The simulation system is integrated as a plug-in in VISUM.

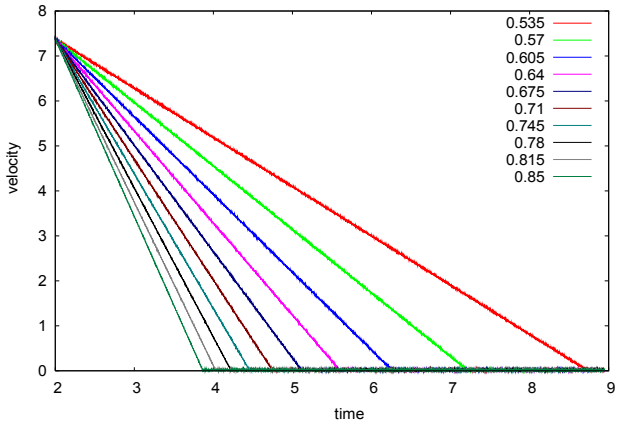

Figure 12: Velocities of decelerating rigid bodies

6- Conclusion and future work

In this paper an impulse-based dynamic simulation system was presented. This system is able to run parallel to a VR application and to simulate the dynamics of a VR scene. In a multi-processor system the simulation can run on an own processor. This raises the performance of the VR application. The VR scene for the simulation can be created with an extension of the 3D modelling tool Maya and the description of this scene can be exported in a XML-file. The impulse-based method that was introduced can handle joint and non-penetration constraints. All constraints are satisfied by applying impulses to the rigid bodies. The inner forces of the joints are simulated by these impulses. The dynamic simulation system can achieve very accurate results in real-time.

It is planned to improve the collision detection by exchanging the binary search method with a method that is based on a better estimation for the time of contact. As result the simulation will run faster. Another two improvements are planned for the collision response. The friction model of Coulomb will be implemented for the new method and a special joint for resting contacts will be developed. A higher accuracy should be achieved with this joint.

7- References

[1] Alias Wavefront. Maya. http://www.aliaswavefront.com, March 2005

[2] Smith R. Open dynamics engine.

http://opende.sourceforge.net, March 2005

[3] Mirtich B., Canny J. Impulse-based Dynamic Simulation. In Proceedings of Workshop on Algorithmic Foundations of Robotics, February 1994.

[4] Mirtich B., Canny J. Impulse-based Simulation of Rigid Bodies. Symposion on Interactive 3D Graphics, Monterey, Cal., April 1995.

[5] Baraff D. Dynamic simulation of non-penetrating rigid body simulation. PhD thesis, Cornell University, 1992.

[6] Gilbert E. G., Johnson D. W., Keerthi S.S. A fast procedure for computing the distance between complex objects in three-dimensional space. In IEEE Journal of Robotics and Automation, vol. 4, pp. 193-203, Apr.1988.

[7] Lin M., Canny J. A fast algorithm for incremental distance calculation. In IEEE Conf. on Robotics and Automation, pages 1008-1014, 1991.

[8] Van den Bergen G. A Fast and Robust GJK Implementation for Collision Detection of Convex Objects. In Journal of Graphics Tools, 4(2):7-25, 1999.

[9] Mirtich B. V-Clip: Fast and robust polyhedral collision detection. ACM Transactions on Graphics, 17(3):177-208, July 1998.

[10] Mirtich B. Impulse-based dynamic simulation of rigid body systems. PhD thesis, University of California, Berkeley, 1996.

[11] Guendelman E., Bridson R., Fedkiw R. Nonconvex rigid bodies with stacking. In Proceedings of SIGGRAPH 2003, pp. 871–878.

[12] GNUplot. http://www.gnuplot.info/, March 2005

[13] Shoemake K. Animating rotation with quaternion curves. In SIGGRAPH ’85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques, pages 245–254. ACM Press, 1985.

[14] Mirtich B. Fast and accurate computation of polyhedral mass properties. In Journal of Graphics Tools: JGT, 1(2):31–50, 1996.

[15] Baraff D. Physically Based Modeling: Rigid Body Simulation. SIGGRAPH Course Notes, ACM SIGGRAPH, 2001.

[16] Press W. H., Flannery B. P., Teukolsky S. A., Vetterling W. T. Numerical Recipes in C, Cambridge University Press, New York, 1992

[17] Finkenzeller D., Baas M., Thüring S., Yigit S., Schmitt A. VISUM: A VR system for the interactive and dynamics simulation of mechatronic systems. Virtual Concept 2003, Biarritz, 5.-7. November 2003.

