IMPULSE-BASED DYNAMIC SIMULATION ON THE GPU

Daniel Bayer, Jan Bender and Raphael Diziol
Universität Karlsruhe
Am Fasanengarten 5
76128 Karlsruhe
Germany

ABSTRACT

In this paper a new, efficient method for dynamic simulation on the GPU is presented. The method is based on an impulse-based approach which is an ideal candidate to simulate on limited hardware due to its simplicity. The proposed method shows how the impulse-based dynamic simulation can benefit from the highly parallel structure of the GPU without suffering too much losses by its limitations. This is achieved by the use of a new way to solve constraints. Most parts of the actual computation can be done in parallel, using only a few number of operations. This allows the implementation to run on a wide range of graphics boards.

KEYWORDS

physics-based animation, dynamic simulation, impulse-based simulation, GPU

1. INTRODUCTION

The physically-based simulation of constrained bodies is a challenging problem in the field of computer graphics. In recent years, it became more and more important to create animations that are closer to reality.
The requirements nowadays exceed the simulation of a comparable small number of constrained rigid bodies. Deformable bodies, for example, can be simulated through a huge number of interconnected parts. In this case the simulation quality is directly related to the number of parts in which the deformable body is subdivided. The computation of such complex simulations is quite expensive. To decrease the CPU load, parts of this computation can be outsourced to the GPU.
In this paper an impulse-based method is presented which computes dynamic simulations on the GPU. It was developed to simulate particle-based deformable bodies, such as cloth. Therefore, this paper is focused on particle simulations, but the given approach also works for the simulation of constrained rigid bodies.
Since constraints are handled implicitly, the simulation can use a large time step size without getting a stability problem. The method supports equality (holonomic) constraints which are used to simulate joints as well as inequality and velocity (nonholomic) constraints which are required to simulate collisions and contacts with friction. The approach handles systems with cycles and is not affected by numerical drift in contrast to the Lagrange multiplier method (cf. [1]). This enables the method to efficiently compute a wide range of simulations, such as articulated rigid bodies or constrained particle systems for deformable bodies.

2. RELATED WORK

Due to their increasing performance and programmability, graphics boards become more and more interesting for general purpose computation (GPCGPU). Today GPCGPU covers a wide range of topics, for example collision detection (cf. [2]). [3] gives a technical motivation for GPCGPU and a summary of its application domains.
In the area of physically-based simulation GPU-based methods are an important field of research. Harris et al., for example, used cellular automata to simulate various dynamic phenomena like boiling, convection and

chemical reaction-diffusion [4]. There are many other works to simulate a variety of physical processes, like fluid flow (cf. [5]) or n-body simulations.

In the area of dynamic simulation on the GPU most works discuss the simulation of particle systems. Kolb et al. introduced a state-preserving particle system fully implemented on the GPU [6]. They describe how to store the states of a particle system in textures on the GPU. Furthermore, methods for the detection and resolution of collisions are presented in this paper. Their collision detection is based on distance maps. The contact resolution and other constraints are computed as explicit forces. Therefore, the constraint error directly depends on the time step size and a fulfilment of the constraint cannot be guaranteed. The work of Kolb et al. motivated the data management and collision detection used in the approach presented in this paper.

Particle systems have also been used to simulate elastic bodies on the GPU. Zeller used spring-mass systems for a basic cloth simulation on the GPU [7]. Georgii et al. extended this approach to three-dimensional deformable bodies by adding volume preservation [8]. Since these approaches use spring forces, they cannot handle conflicting constraints. Furthermore, these methods can run into problems like stiff differential equations which cannot be solved efficiently using large time step sizes and lead to stability problems. Hence, for the simulation of inflexible materials, a very small time step size has to be used. An alternative for the simulation of cloth is the use of Finite Element Methods [9].

The dynamic simulation of rigid bodies is also an area of research which profits from GPU-based methods. Takahiro Harada presented several works in this area, as for example [10] and [11].

3. GPU-BASED DYNAMIC SIMULATION

The following sections describe how constrained particles are simulated dynamically and in which way this computation can be done using the graphics hardware.

3.1 Particle simulation

A particle, or point mass, is a body without extend and has therefore, in contrast to rigid bodies, no rotation. Its physical state can be expressed with its constant scalar mass m , its position $\boldsymbol{p}$ and its linear velocity $\boldsymbol{v}$. To dynamically simulate a particle, its position and velocity have to be integrated over a time step of size h . Using external forces $\boldsymbol{F}$ that are constant during $[t_0;t_0+h]$, the new position and velocity can be directly computed by the following equations:

$$\boldsymbol{v}(t_0+h)=\boldsymbol{v}(t_0)+\int_0^h \frac{\boldsymbol{F}}{m}dt=\boldsymbol{v}(t_0)+\frac{\boldsymbol{F}}{m}h \tag{1}$$

$$\boldsymbol{p}(t_0+h)=\boldsymbol{p}(t_0)+\int_0^h \boldsymbol{v}(t_0)+\frac{\boldsymbol{F}}{m}t\,dt=\boldsymbol{p}(t_0)+\boldsymbol{v}(t_0)h+\frac{1}{2}\frac{\boldsymbol{F}}{m}h^{2,} \tag{2}$$

where t_0 is the current time before the time step.

3.2 Constraint enforcement

The particles' motion is constrained by the use of implicit functions $\boldsymbol{C}(\boldsymbol{p},\boldsymbol{v},t)=\boldsymbol{0}$ or $\boldsymbol{C}(\boldsymbol{p},\boldsymbol{v},t)\geq\boldsymbol{0}$ where the vectors $\boldsymbol{p}$ and $\boldsymbol{v}$ contain all positions and velocities of the particles. Any function of this kind can be used to restrict the motion. The simulation of these constraints is explained in detail in the following. At first, the joint state is evolved forward in time in order to predict the constraint error at the end of the simulation step. This is done by integrating the differential equation 2 to get the positions of the particles at time t_0+h . With this predicted state the constraint error $\boldsymbol{e}$ can be computed by solving the constraint function at time t_0+h :

$$\boldsymbol{e}=\boldsymbol{C}\left(\boldsymbol{p}(t_0+h),\boldsymbol{v}(t_0+h),t_0+h\right).$$

For unilateral constraints $\boldsymbol{e}$ is zero, if $\boldsymbol{C}(t_0+h)\geq\boldsymbol{0}$. If the error is greater than a certain tolerance value ε , a correction impulse $\boldsymbol{I}$ is applied. For two bodies i and j , which are linked by a constraint $\boldsymbol{C}$, this impulse is given by:

$$\boldsymbol{I}=\left(\frac{1}{m_i}+\frac{1}{m_j}\right)^{-1}\begin{cases}\dfrac{\partial\boldsymbol{C}}{\partial\boldsymbol{p}}\boldsymbol{e}\dfrac{1}{h}, \textit{for a position constraint}\\ \dfrac{\partial\boldsymbol{C}}{\partial\boldsymbol{v}}\boldsymbol{e}, \textit{for a velocity constraint.}\end{cases}$$

The impulse points in the direction $\nabla\boldsymbol{C}$, as the principle of virtual work states, and is applied to both linked bodies in opposite directions. Hence, $\boldsymbol{I}$ does not violate the conservation of momentum of the multi-body system. The resulting impulse instantaneously changes the velocity of the linked bodies, so that the error is eliminated.
If the system contains more than one constraint, the corresponding impulses are computed and applied one after another. The effect of one impulse may violate a constraint that was already satisfied. These dependencies between the constraints are resolved by computing the impulses in an iterative process. The process ends, when all errors are below a certain threshold value ε . This process converges to the physical correct solution (cf. [12]). The iterative approach has the advantage that it can even handle multi-body systems with cycles without additional effort (cf. [1]). This is an important property, especially when simulating cloth models.

3.3 Parallel constraint enforcement

The dependence of single operations has to be reduced to make reasonable use of the parallel GPU architecture. As seen in the previous section any computed impulse changes the velocity of both linked bodies. The single iterations depend on each other, since the velocity change of one body, to satisfy one constraint, may violate another constraint. In dense systems, like cloth models, each particle is linked with all adjacent particles. The resolution of one constraint will effect the velocities of all neighbouring particles and therefore the whole model.
The main contribution of this paper is an efficient strategy to process the constraints, so that large parts can be solved independently. Schmitt et al. have proofed that the iterative process that is used in the presented method always converges to the physical correct solution [12]. The order in which the corrections are performed can influence the convergence rate but not the convergence itself. As the results show the influence on the convergence rate is even insignificant.
In order to do the computation on the GPU, the constraints are separated in several groups, so that every group is internally independent. This means every group consists of constraints whose bodies are pairwise linked by at most one constraint of this group. Algorithm 1 shows how these groups of independent constraints are determined. Thereby, the maximum number of constraints per body is the number of groups needed. With this arrangement, all impulses for one group can be determined in parallel.

```
Input:  List of all constraints C, G=[]
Output: List of constraint groups G
for all c1 in C
        addToNewGroup = true
        for all g in G
                addToThisGroup = true
                for all c2 in G
                        if haveCommonBody(c1, c2)
                                addToThisGroup = false
                                break
                if addToThisGroup
                        addConstraintToGroup(c1, g)
                        addToNewGroup = false
                        break
        if addToNewGroup
                addConstraintToNewGroup(c1)
return G
```

Algorithm 1: Build groups of independent constraints

3.4 Circumvention of memory interchanges

Interchanges between system memory and graphics memory are a performance bottleneck and should be reduced to a minimum. In this approach this is done by managing the data only in the graphics memory whenever possible. Therefore, external forces, positions, velocities and constraint states are maintained completely on the GPU. Memory interchanges are only necessary to decide, if the error of all constraints is below ε or when the constraint structure or particle states are changed on the CPU side.

3.5 Real-time simulation

A major advantage of the proposed method is, that the computation can be aborted with an approximate solution at any time. This property makes it well-suited for real-time applications. In such an application the iteration process is interrupted, when the time of the next frame is reached. This allows a guaranteed and fixed frame rate, although the constraint errors may exceed the desired tolerance ε temporarily. Another advantage of this procedure is that the status of the constraints has not to be accessed by the CPU.

4. IMPLEMENTATION

Based on the previous sections this chapter describes details of the implementation of the presented method. The next section presents the structure of the simulated model. Afterwards the data structures and shader programs used for the impulse-based dynamic simulation on the GPU are explained in detail.

4.1 Simulated model

A rectangular piece of cloth interacting with a static environment was chosen to test the given approach. This simulation contains equality constraints for the cloth model and inequality constraints for the contacts.

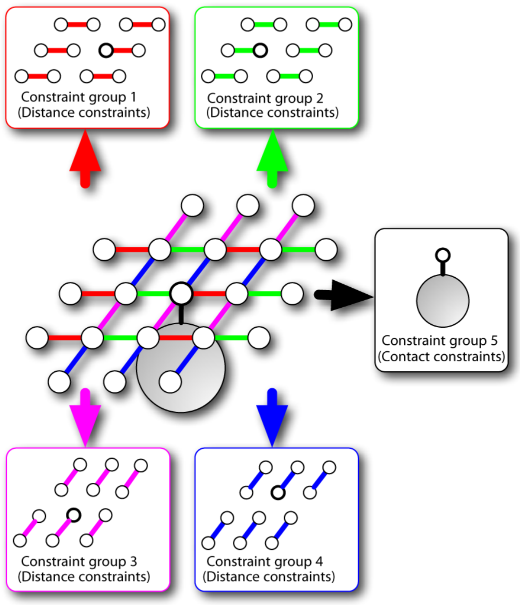


Figure 1. Subdivision of a small cloth model in independent constraint groups which are required for the parallel simulation

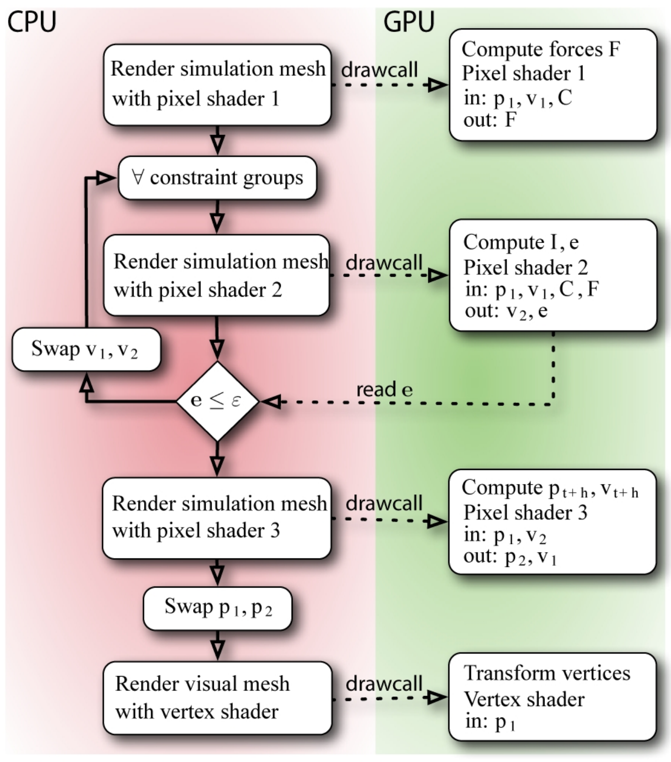
A simple collision detection with static obstacles like planes and spheres was implemented in order to detect the contacts. The obstacles are described as implicit functions. By the use of distance maps this method is

easily extended to work for any polyhedral model. In case of the GPU-based simulation the contacts are directly computed on the graphics hardware.
The cloth model is motivated by [13] and consists of distance constraints connecting the particles with their direct neighbours. These constraints restrict the inner plane motions, like stretching and shearing, and allow to make the cloth inextensible (cf. [14]). The model is completed by interlaced constraints connecting always two particles by skipping the direct neighbour. These constraints are necessary for bending and compression resistance and are modelled as explicit spring-dampers.
Figure 1 shows how a small piece of cloth is subdivided in independent constraint groups. Using these groups a whole iteration can be performed by just three shader calls.

4.2 Data structures

128-bit floating point textures are used to store the physical properties of the system in the graphics memory. This means there are four floating point values per pixel. It is important to note that it is not possible to read from and write to the same texture at the same time. Therefore, pairs of textures are used for each variable property of the system. After each computation these textures are swapped so that the output can be used as input for the next render pass.

4.3 Shader programs



CPU
GPU
Render simulation mesh with pixel shader 1
drawcall
Compute forces F Pixel shader 1 in: p_1, v_1, C out: F
$\forall$ constraint groups
Render simulation mesh with pixel shader 2
drawcall
Compute I, e Pixel shader 2 in: p_1, v_1, C, F out: v_2, e
Swap v_1, v_2
$e \leq \varepsilon$
read e
Render simulation mesh with pixel shader 3
drawcall
Compute p_{t+h}, v_{t+h} Pixel shader 3 in: p_1, v_2 out: p_2, v_1
Swap p_1, p_2
Render visual mesh with vertex shader
drawcall
Transform vertices Vertex shader in: p_1

Figure 2. Procedure of the GPU-based computation and its memory interchanges

Three pixel shaders and one vertex shader were implemented to compute the impulse-based dynamic simulation on the GPU. Figure 2 summarizes the whole computation process and displays all memory interchanges between the CPU on the left side and the GPU on the right side. Every task except the scene management is performed on the GPU. The CPU accesses the GPU memory only for initialisation and to check, if all constraints are satisfied. In the case of real-time computation (see section ”Real-time simulation”) this test is not even necessary, since a fixed amount of iterations are computed for each frame.

At first, a pixel shader is used to compute external forces like the gravity $\boldsymbol{g}$ and a set of spring-dampers S. These forces are considered constant for one time step. Depending on the number of explicit constraints this shader may need multiple passes. For all particles i:

$$\boldsymbol{F}_i = \boldsymbol{g}\, m_i + \sum_{\forall s \in S} \begin{cases} \boldsymbol{F}_{spring}(i, s), & s \textit{ connected to } i \\ \boldsymbol{0}, & \textit{otherwise} \end{cases}$$

is computed, where $\boldsymbol{F}_{spring}(i, s)$ returns the spring damper force of spring s and body i.
The second pixel shader computes the impulses and, therefore, the new velocities. This shader also writes the according constraint errors in a status textures. If there is an error greater than ε, the input and output velocity textures are swapped. This process is repeated for each joint group until the velocity texture is not swapped any more.
In this implementation two different implicit constraints are used (see section ”Simulated scene”). At first there is a distance joint connecting two particles i and j at positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ with an initial distance l_0. This constraint is given by the following equation:

$$\boldsymbol{C}_{distance}(\boldsymbol{p}_i, \boldsymbol{p}_j) = |\boldsymbol{p}_i - \boldsymbol{p}_j| - l_0 = 0.$$

The second constraint is a simple contact constraint to avoid penetration with the static environment. It is given by the equation

$$\boldsymbol{C}_{contact}(\boldsymbol{p}_i) = (\boldsymbol{p}_i - \boldsymbol{c})\, \boldsymbol{n} \geq 0$$

where $\boldsymbol{c}$ is the contact point on the static geometry and $\boldsymbol{n}$ the outwards pointing normal of the contact surface.
After all constraints have been fulfilled the third pixel shader computes the new positions and velocities of the bodies using equations 1 and 2. After that, the system is in a legal state and can be drawn. This is done by the vertex shader which computes the positions of the vertices of a mesh according to the new physical positions of the particles.

5. RESULTS

To test the proposed method, a simulator based on DirectX/HLSL was implemented. The simulator enables the change of global parameters during runtime, for example, the external force or the scene structure. It also allows the direct comparison between the CPU- and the GPU-based computation by supporting both techniques.
Figure 3(a) shows a comparison of the GPU and CPU simulation in the simulator. The cloth model used for the comparison consists of 1024 particles. A maximum of 100 iterations was used for the simulation. The maximal strain of the cloth model was less than one percent during the simulation.
The results of the runtime measurements made with the simulator are summarized in figure 3(b). For these measurements also a maximum of 100 iterations was used. The figure displays the average computation time in milliseconds against the number of connected particles for the CPU and the GPU computation with and without shader model 3.0 support.
The cloth model with 1024 particles required less than 100 iterations and had a maximal strain of less than one percent. The iteration process of the biggest model with 16384 particles was stopped after 100 iterations and therefore, had a maximal strain of eight percent. The tests where run on a Intel Core2 Quad Q9450@2.66GHz with 8GB Ram and a NVIDIA GeForce9800 GTX@675MHz with 512MB Ram.
The diagram shows, that the GPU-based method for large systems is faster than the CPU-based. It is almost able to compute 16384 constrained particles in real-time. Whereas the CPU is able to compute up to 512 in real-time. Up to this number of particles the CPU-based approach is faster than the GPU-based method. The computation times of the GPU-based methods with and without shader model 3.0 support are nearly equal, which shows that even older hardware can challenge the CPU for larger systems. It is important to mention that this measurements are made without rendering the scene, because the GPU-based method with shader model 3.0 support gets a major boost through the direct rendering using the vertex shader described before.
The more complicated the piece of cloth gets, the more iterations are needed to satisfy all constraints. Therefore, an inextensible piece of cloth containing a very large number of particles cannot be simulated in real-time. The GPU-based approach requires nearly the same computation time for one piece of cloth and for

the simulation of many independent pieces of cloth due to the parallel computation. For the next comparison multiple cloths with 64, 128, 256, 512 and 1024 particles where computed by the GPU. As figure 3(c) shows, up to 16 pieces with 1024 particles could be computed in real-time by the GPU, whereas the CPU could not even compute one in real-time.

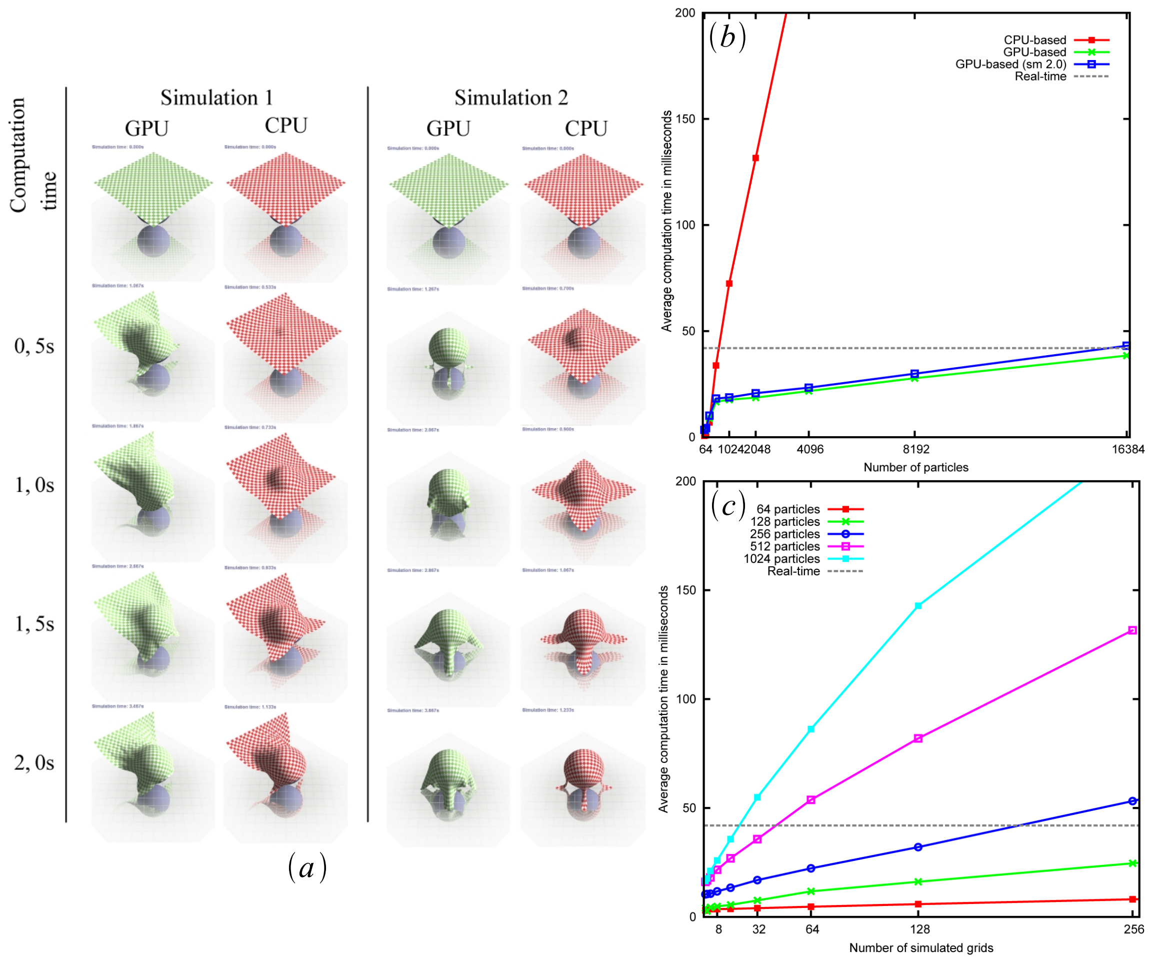


Figure 3. (a) Simulation of a piece of cloth consisting of 1024 linked particles. (b) The results show the average computation time of the CPU-based and the GPU-based approach, with and without shader model 3.0 support. (c) Average computation time of the GPU-based approach simulating multiple grids of the same dimension.

Using a fixed number of iteration steps, even more complicated pieces of cloth can be simulated in real-time, since the status of the constraints has not to be checked on the CPU. Figure 4 shows the simulation of a cloth model consisting of 65536 particles. The drawback is, that the cloth may get more expandable as it should, but this error is comparable small and eliminated over the time, thus leading to at least visual plausible results.

6. CONCLUSION

In this paper a method for the dynamic simulation on the GPU is presented. This method is well suited for large systems of articulated bodies. As figure 3 shows, a huge number of constrained particles can be computed in real-time on the GPU. This is possible by the use of the presented constraint solving strategy to minimise interdependent operations and memory interchanges.

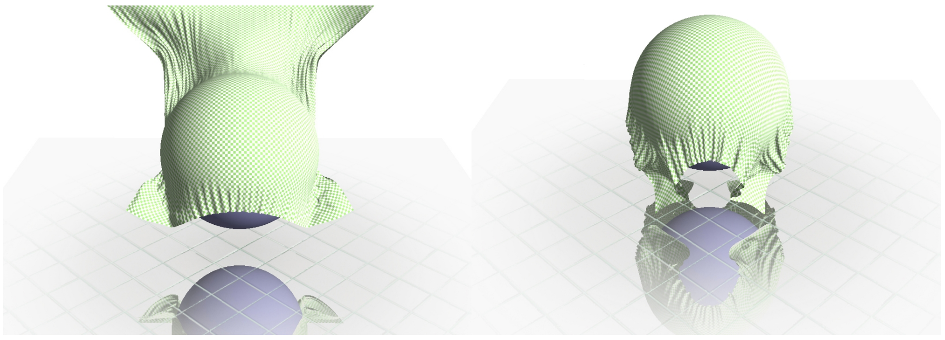


Figure 4. Real-time simulation of a cloth model consisting of 65536 particles with more than 130 000 distance constraints and a few thousand contact constraints

A wide range of different constraints can be computed, due to the general constraint specification. Inequality constraints and cyclic systems are handled without special treatment. The implicit constraint enforcement allows the system to take large time steps and to make connections stiff.

The implementation on the graphics hardware is simple, so that it is possible to do the complete computation within the limits of a shader model 2.0 GPU. With little changes, the implementation can easily be extended to support rigid-bodies as well, which has already been proven.

The presented approach has real-time capabilities. This makes it interesting for computer games and virtual reality applications, where a guaranteed frame rate is more important than an exact solution. Especially, if the GPU-based objects have no physical influence on the user action, managed on the CPU, the method is well suited. This holds for example for cloths and curtains, simulated in a computer game.

REFERENCES

[1] Bender, J. and Schmitt, A., 2006. Fast dynamic simulation of multi-body systems using impulses. *In Virtual Reality Interactions and Physical Simulations (VRIPhys).* Madrid, Spain, pp. 81–90.

[2] Heidelberger, B. et al, 2004. Detection of collisions and self-collisions using image-space techniques. *In Journal of WSCG.* pp. 145–152.

[3] Owens, J. D. et al, 2007. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum,* Vol. 26, No. 1, pp. 80–113.

[4] Harris, M. J. et al, 2002. Physically-based visual simulation on graphics hardware. *In HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware.* Aire-la-Ville, Switzerland, pp. 109–118.

[5] Krüger, J. et al, 2005. A particle system for interactive visualization of 3d flows. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, No. 6, pp. 744–756.

[6] Kolb, A. et al, 2004. Hardware-based simulation and collision detection for large particle systems. *In HWWS '04: Proc. of the ACM SIGGRAPH/ EUROGRAPHICS conference on Graphics hardware.* New York, USA, pp. 123–131.

[7] Zeller, C., 2005. Cloth simulation on the gpu. *In ACM SIGGRAPH 2005 Sketches.* New York, USA, pp. 39.

[8] Georgii, J. et al, 2005. Interactive simulation of deformable bodies on gpus. *In Proceedings of Simulation and Visualisation 2005.* pp. 247–258.

[9] Rodriguez-Navarro, J. and Susin, A., 2006. Non structured meshes for cloth gpu simulation using fem. *In Virtual Reality, Interactions and Physical Simulations (VRIPhys).* Madrid, Spain, pp. 1–7.

[10] Harada, T., 2007. *Gpu gems 3, Real-time Rigid Body Simulation on GPUs, chapter 29, pp. 611– 632*. Addison-Wesley Professional.

[11] Harada, T. et al, 2007. Acceleration of rigid body simulation using graphics hardware. *In Symposium on Interactive 3D Graphics and Games.* Seattle, USA.

[12] Schmitt, A. et al, 2005. On the convergence and correctness of impulse-based dynamic simulation. *Internal Report 17, Institut für Betriebsund Dialogsysteme*.

[13] Choi, K.-J. and Ko, H.-S., 2002. Stable but responsive cloth. *In SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques.* New York, USA, pp. 604–611.

[14] Bender, J. and Bayer, D.,2008. Parallel simulation of inextensible cloth. *In Virtual Reality Interactions and Physical Simulations (VRIPhys).* Grenoble, France, pp. 47–55.