Detection of Collisions and Self-collisions Using Image-space Techniques

Bruno Heidelberger Matthias Teschner Markus Gross

Computer Graphics Laboratory
ETH Zurich

Abstract. Image-space techniques have shown to be very efficient for collision detection in dynamic simulation and animation environments. This paper proposes a new image-space technique for efficient collision detection of arbitrarily shaped, water-tight objects. In contrast to existing approaches that do not consider self-collisions, our approach combines the image-space object representation with information on face orientation to overcome this limitation.
While image-space techniques are commonly implemented on graphics hardware, software solutions have been neglected so far. In this paper, the performance of two GPU-based implementations and one CPU-based implementation of the proposed collision detection algorithm are compared. Results suggest, that graphics hardware accelerates collision detection in geometrically complex environments, while the CPU-based implementation provides more flexibility and better performance in case of small environments.

Keywords. Collision detection, self-collision, image-space technique, graphics hardware, animation.

1 Introduction

Efficient collision detection is a fundamental problem in physically-based simulation and in computer animation [Gan00], [Tes03]. Especially, collision detection is addressed in cloth modeling [Vol95], [Pro97], [Bri02] and in medical simulations, e. g. to handle the interaction of surgical tools and tissue [Lom99]. Further applications can be found in robotics [Cam90], computational biology [Tur90], and games [Mel00].

In order to accelerate collision detection for rigid bodies, many approaches based on pre-computed bounding-volume hierarchies have been proposed [Hub93], [Qui94], [Got96], [Klo96], [Ber97], [Lar01], [Zac02], [Bra02]. However, in case of deformable objects these hierarchical data structures cannot be pre-computed, but have to be updated frequently. Although effort has been spent to optimize this update [Lar01], it is still expensive in case of dynamic environments.

Recently, image-space techniques have been proposed for collision detection [Mys95], [Bac99], [Hof01], [Kim02], [Hei03], [Kno03], [Gov03]. These approaches commonly process projections of objects. They do not require any pre-processing and employ graphics hardware. Therefore, they are especially appropriate for dynamic environments.

In this paper, a new image-space technique for collision detection of arbitrarily shaped, deforming objects with water-tight surfaces is presented. The approach is based on Layered Depth Images (LDI), as presented in [Hei03]. In contrast, the new method combines LDI-based object representation with information on face orientation to detect collisions and self-collisions.

Further, this paper compares CPU-based and GPU-based implementations of the presented collision detection algorithm. Results suggest, that graphics hardware can accelerate the collision detection in large environments with up to 500k faces, while the CPU-based implementation provides more flexibility and better performance in case of small environments consisting of up to several thousand surface triangles. Although graph-

Journal of WSCG, Vol. 12, No. 1-3, ISSN 1213-6972
WSCG 2004, Feb 2-6, 2004, Plzen, Czech Republic.

ics hardware is very efficient in rasterizing the LDI that is required for the collision detection, multiple rendering passes are required. In contrast, the CPU-based implementation processes in only one rendering pass. Additionally, data read-back from the GPU, which is comparatively expensive in small environments, is avoided.

2 Related Work

An early approach to image-space collision detection has been outlined in [Shi91]. The two depth layers of convex objects are rendered into two depth buffers. Now, the interval from the smaller depth value to the larger depth value at each pixel approximately represents the object and is efficiently used for interference checking. A similar approach has been presented in [Bac99]. Both methods are restricted to convex objects and do not consider self-collisions.

In [Mys95], an image-space technique is presented which detects collisions for arbitrarily-shaped objects. This approach can process concave objects. However, the maximum depth complexity is still limited. Additionally, object primitives have to be pre-sorted. This method cannot efficiently work with deforming objects and self-collisions are not detected.

A first application of image-space collision detection to cloth simulation has been presented in [Vas01], and a medical application is presented in [Lom99], where intersections of a surgical tool with deformable tissue are detected by rendering the interior of the tool. Again, these approaches are restricted to convex objects.

In [Hof01], an image-space method is not only employed for collision detection, but also for proximity tests. This method is restricted to 2D objects. In [Kno03], edge intersections with surfaces can be detected in multi-body environments. This approach is very efficient. However, it is not robust in case of occluded edges. In [Gov03], several image-space methods are combined for object and sub-object pruning in collision detection. The approach can handle objects with changing topology. However, the setup is very complex and self-collisions are not considered.

In [Hei03], an image-space technique is used to compute an LDI representation for each object. Based on this data structure, collisions can be detected. However, self-collisions cannot be found.

In contrast to existing approaches, the proposed approach does not only detect collisions between different objects, but also handles self-collisions of arbitrarily-shaped, deformable objects with closed surface. The image-space algorithm computes an LDI, which allows to process three different collision and self-collision queries. First, self-collisions of an object can be detected. Second, the intersection volume of two objects can be computed. Third, vertices penetrating the volume of an object can be detected. Additionally, we investigate drawbacks and benefits of three different GPU-based and CPU-based implementations.

3 Method

In this section, the image-space approach for the detection of collisions and self-collisions is presented. An overview of the algorithm is given in Sec. 3.1, followed by a detailed description of three different implementations. Sec. 3.2 and Sec. 3.3 address two variants that are accelerated with graphics hardware. Sec. 3.4 describes an implementation that does not employ the GPU. Refer to Sec. 4 for a performance analysis and a discussion of the characteristics of the three implementations.

3.1 Overview

Our approach detects collisions and self-collisions of 3D objects of manifold geometry. Although the approach is not confined to triangular meshes, a watertight object surface is required in order to perform volumetric collision queries. The algorithm computes an approximative volumetric representation of the object. This representation is used for three different collision queries. The algorithm proceeds in three stages.

Stage 1 computes the Volume-of-Interest (VoI). The VoI is an axis-aligned bounding-box (AABB) representing the volume where collision queries are performed. In the case of a self-collision test, the VoI is chosen to be the AABB of the object. When a collision test between a pair of objects is performed, the VoI is given as the intersection of the AABBs of the objects. If the VoI is empty, no collision is detected and the algorithm aborts. If the VoI is not empty, the objects and the VoI are further processed in stage 2.

Stage 2 computes an LDI for objects inside the VoI. Note, that LDI generation is restricted to the VoI and object primitives outside the VoI are discarded. The LDI consists of images or layers of depth values representing object surfaces. The depth values of the LDI can be interpreted as inter-

sections of parallel rays or 3D scan lines entering or exit the object. Thus, an LDI classifies the VoI into inside and outside regions with respect to an object. The concept is similar in spirit to well-known scan-conversion algorithms to fill concave polygons [Fol90], where intersections of a scan line with a polygon represent transitions between interior and exterior. In contrast to [Hei03], additional information on face orientation is stored in the LDI. Thus, depth and front-face / back-face classification is known for each entry in the LDI data structure. Stage 2 results in an LDI with sorted depth values and explicitly labeled entry (front-face) and exit (back-face) points within the VoI.

Stage 3 performs one of three possible collision queries. *a)* Self-collisions are detected by analyzing the order of entry (front-face) and exit (back-face) points within the LDI. If they correctly alternate, there is no self-collision. If invalid sequences of front-faces and back-faces are detected, the operation provides the explicit representation of the self-intersection volume. *b)* Collisions between pairs of objects are detected by combining their LDIs using Boolean intersection. If the intersection of all inside regions is empty, no collision is detected. If it is not, the operation provides an explicit representation of the intersection volume. *c)* Individual vertices are tested against the volume of the object. The vertex is transformed into the local coordinate system of the LDI. If a transformed vertex intersects with an inside region, a collision is detected.

The VoI computation in stage 1 and the collision queries in stage 3 do not significantly contribute to the computational cost of the algorithm (see [Hei03]). Since the LDI generation in stage 2 is comparatively expensive, we have analyzed three variants. Two variants employ the GPU, which is obviously useful, since stage 2 basically rasterizes triangles. However, lack of flexibility in GPU implementations and the data read-back delay motivated a third implementation, which is completely processed on the CPU. Sec. 3.2-3.4 give a detailed description of the three variants of stage 2.

3.2 Ordered LDI

The first method generates an ordered LDI using graphics hardware and is based on depth peeling. A similar approach has been used for correct rendering of transparent surfaces in [Eve01]. In contrast, we do not render a fixed number of layers, but use a flexible abort criterion to generate the entire LDI of an object with respect to the VoI. Additionally, color information is not considered in our approach. Back-faces and front-faces of an object are handled in two successive steps. This allows to label all LDI entries accordingly. This separation also eliminates singularities of contour edges, which is a robustness problem of the original approach. If back-faces and front-faces coincide, the original algorithm produces a single depth value instead of two, thus falsifying the inside / outside classification.

In order to solve this problem and to generate an LDI with additional information on face orientation, the following multi-pass algorithm is performed twice. The first pass processes front-faces, followed by a second pass for back-faces. Two depth buffers with corresponding depth tests are used. One buffer is active, one is passive. Switching between active and passive buffer is possible.

1. Active depth buffer is cleared. Object is rendered into active depth buffer with a regular depth test.
2. If no depth value has been written to the active depth buffer, LDI generation is aborted.
3. The content of the active depth buffer is stored in the LDI structure. Active and passive depth buffer are switched.
4. Active depth buffer is cleared. Object is rendered into the active depth buffer with regular depth test. In addition, a second test with the passive depth buffer is performed. Values with a camera distance smaller or equal than in the passive depth buffer are discarded.
5. Go back to step 2.

This algorithm is implemented using core OpenGL 1.4 functionality and a few widely used OpenGL ARB extensions. The second depth test is provided by the *GL_ARB_shadow* and *GL_ARB_depth_texture* extensions through hardware-accelerated shadow-mapping functionality as described in [Eve01]. In contrast, we use an occlusion query is used as a robust abort criterion for the multi-pass algorithm. By enabling the occlusion query mode using *GL_ARB_occlusion_query*, the number of written fragments is automatically accumulated by the GPU. This counter can be queried after the rendering is completed. If one or more fragments have been rendered, this counter is larger than zero, i. e. another valid layer of the LDI has been produced. Therefore, LDI generation is aborted if the occlusion query returns zero written fragments. Fig. 3.3 illustrates the Ordered LDI generation.

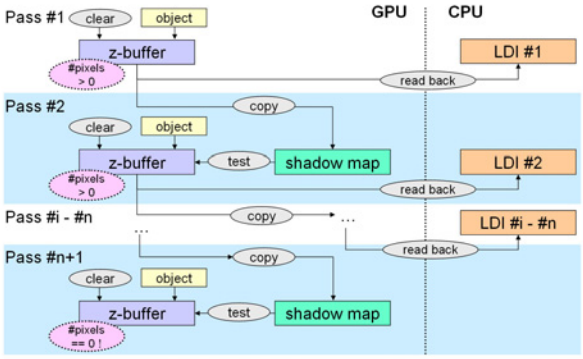

Figure 1: An ordered LDI is generated in multiple rendering passes. The result of a rendering pass is copied to the shadow map, which is used as a second depth buffer in the subsequent pass.

3.3 Unordered LDI

The second method generates an unordered LDI using graphics hardware. To get an ordered LDI, depth values are sorted on the CPU. In this approach, the second depth buffer, required in the Ordered LDI approach, is eliminated. Depth tests are disabled. In contrast to [Hei03], front-faces and back-faces are handled in successive steps to provide information for self-collision queries in stage 3.

The following multi-pass algorithm is performed once for front-faces and again for back-faces. Depth buffer and stencil buffer are employed. Stencil tests and stencil operations are used. Color buffer and depth test are disabled, which improves the performance.

1. Layer index counter is initialized to 0. Stencil test is set to pass if current stencil value is less than or equal to layer index counter. Stencil operation is defined to always increment stencil value for each incoming fragment.

2. Depth and stencil buffer are cleared. Object is rendered into depth buffer.

3. If maximum value in the stencil buffer is zero, LDI generation is aborted. Otherwise, the maximum stencil value and the content of the depth buffer are stored in the LDI structure. Layer index is incremented by one.

4. If the current layer index is greater or equal to the maximum stencil value, the LDI generation is aborted.

5. Depth and stencil buffer are cleared. Object is rendered into depth buffer.

6. Content of the depth buffer is stored in the LDI structure. Layer index counter is incremented by one.

7. Go back to step 4.

Since fragments are rendered in arbitrary order, the algorithm generates an unsorted LDI. For further processing, the LDI is sorted per pixel on the CPU. Although the rendering order is arbitrary, our algorithm relies on consistency across individual passes, which is provided by the GPU.

This algorithm is implemented using core OpenGL 1.4 functionality. OpenGL extensions are not employed. Each rendering pass requires a buffer read-back. Reading back data from the GPU, however, can be expensive depending on the actual hardware. We propose two methods for optimizing the data transfer using OpenGL extensions. First, depth and stencil values of a pixel are usually stored in the same 32-bit word in the frame buffer. This allows to read-back both buffers in a single pass using an OpenGL extension, such as *GL_NV_packed_depth_stencil*. Second, all rendering passes can be performed independently from each other except for the first pass. We exploit this fact to render individual layers into different regions of the frame buffer. Once finished, the whole frame buffer is read back in a single pass. This optimization reduces the number of read-backs to a maximum of two, assuming that the frame buffer memory is sufficiently large. Thus, stalls in the rendering pipeline are reduced and the performance of the algorithm is significantly improved. Fig. 2 illustrates the Unordered LDI generation.

3.4 Software LDI

The third method for LDI generation is completely processed on the CPU. The investigation of this variant has been motivated by two drawbacks of GPU implementations. First, detailed timing measurements of the two graphics-hardware accelerated techniques have indicated that buffer read-backs are a main performance bottleneck. Second, graphics hardware requires multiple passes for LDI generation, since the output is restricted to one value per fragment in the frame buffer. In contrast, a software-renderer does not suffer from this

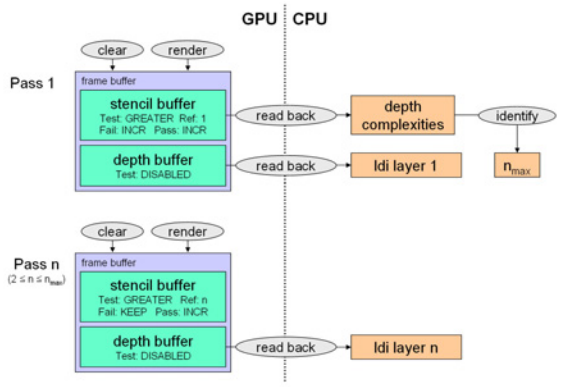

Figure 2: Each rendering pass generates an unsorted LDI layer. Additionally, the first rendering pass computes the depth complexity, i. e. the number of rendering passes.

restriction, as it can rasterize into multiple buffers at the same time. Therefore, a simplified software-renderer has been implemented, which is especially designed for LDI generation.

The simplified software-renderer only features basic frustum culling, face clipping, orthogonal projection, and rasterization of triangle meshes. The produced fragments are directly stored in the LDI structure. This is in contrast to hardware-accelerated methods which depend on an intermediate frame buffer.

First, the software-renderer culls object triangles against the VoI provided by stage 1. If necessary, the remaining triangles are clipped against the VoI. This clipping might produce additional triangles, as shown in [Sut74]. Then, the software-renderer rasterizes all triangles that have passed the culling and clipping stages. The depth of each generated fragment is stored in the LDI structure along with the information on front-face / back-face orientation. Fragments rasterized at the same position in the LDI do not overwrite each other, but result in the generation of an additional LDI layer for this pixel.

The software-renderer generates fragments in arbitrary order, similar to the Unordered LDI method. Therefore, per pixel sorting of the LDI is performed for further processing.

4 Results

We have implemented the three methods presented in Sec. 3 on a PC Pentium 4 with 3 GHz and a GeForce FX Ultra 5800 GPU. Various experiments have been carried out to measure the performance of the methods. This section presents three test scenarios employing different collision queries.

The first scenario uses a dynamically deforming hand and a phone, consisting of 4800 faces and 900 faces, respectively. Volumetric collision detection is performed between the hand and the phone. Additionally, the animated hand is tested for self-collisions. The LDI resolution is 64x64. Tab. 1 shows timings for both collision queries.

Method	Collision min / max [ms]	Self-collision min / max [ms]	Overall min / max [ms]
Ordered	28.6	40.0	68.6
	37.3	54.2	91.5
Unordered	9.5	12.3	21.8
	12.2	17.7	29.9
Software	2.8	4.9	7.7
	4.1	7.1	11.2

Table 1: Dynamic test sequence with an animated hand (4800 faces) and a phone (900 faces). Minimum and maximum values are given for collisions between hand and phone, and for self-collisions of the hand. Overall time for collision and self-collision detection is shown in the last column. The resolution of the LDI is 64x64. Fig. 3 and Fig. 4 illustrate the test.

In the first test scenario, the software LDI provides the best performance, since data transfer from and to the GPU significantly reduces the performance of the Ordered and Unordered LDI approach. Data read-back is independent from the model geometry. Therefore, rasterization performance of the graphics hardware is outweighed by data read-back for smaller geometries. Further, the Unordered LDI approach is more efficient than the Ordered LDI approach. Although both methods require about the same number of rendering passes, the rendering setup for the Ordered LDI approach is more involved. All experiments indicate, that the Unordered LDI approach is about three times faster compared to the Ordered LDI approach.

In the second scenario, arbitrarily placed particles are tested against the volume of a dragon. The particles are randomly positioned within the AABB of the model. The LDI resolution is 64x64.

Three different scene complexities have been tested with up to 500k faces and 100k particles. Measurements are given in Tab. 2.

Method	High Complexity [ms]	Medium Complexity [ms]	Low Complexity [ms]
Ordered	453.1	160.6	50.3
Unordered	225.2	74.5	24.4
Software	398.7	105.2	37.1

Table 2: Particles at arbitrary positions within the AABB of a dragon model are tested against the volume. Measurements are given for three model complexities: High (520k faces, 100k particles), medium (150k faces, 30k particles), and low (50k faces, 10k particles). The LDI resolution is 64x64. Fig. 5 illustrates the test.

In the second experiment, the VoI encloses the entire model, i. e. all triangles of the dragon have to be rendered. In highly complex scenes, most time is spent for triangle rasterization and the read-back delay is less important. In this case, the GPU-based Unordered LDI approach outperforms the software-renderer and a rate of 5Hz can be achieved for a large scene with 500k triangles and 100k particles.

In the third experiment, collisions of a hat and a mouse model with fixed geometric complexity are detected using varying LDI resolutions.

Method	LDI 32 x 32 [ms]	LDI 64 x 64 [ms]	LDI 128 x 128 [ms]
Ordered	24.0	26.0	51.1
Unordered	8.0	8.7	17.2
Software	2.1	3.0	5.9

Table 3: Test with an animated hat (1500 faces) and a mouse (15000 faces). Collisions are detected with intersected LDIs. LDI resolution varies from 32x32 to 128x128. Fig. 6 illustrates the test.

Due to the moderate geometric complexity of the scene, the Software LDI shows the best performance in the third experiment. It can be seen, that the LDI resolution significantly influences the performance of our approach. In all methods, higher LDI resolution requires more fragments to be rendered. Additionally, data read-back slows down in case of higher LDI resolution for GPU-based approaches.

In this experiment, collision detection of the test scenario with 16k triangles can be performed with a rate of 500Hz using the Software LDI approach. Therefore, this method is well-suited for games or for interactive animation environments.

In general, all experiments show that our approach to collision and self-collision detection can be accelerated with graphics hardware for large environments. In typical game environments however, collisions are checked between less complex characters comparable to the third experiment. In such applications, the CPU-based implementation provides best performance with up to 500Hz.

5 Ongoing Work

We are currently focussing on applications of the presented collision detection technique. Based on research in efficient collision response and fast deformable models, we intend to integrate all components into a system for real-time cloth simulation and for interactive surgery simulation. In cloth simulation, fast collision detection for deformable objects is required to handle the interaction of cloth and walking avatars. In surgery simulation, the presented approach can be employed to interactively simulate interactions between different deformable organs.

Further, we are investigating collision detections methods, that overcome the current limitation of closed objects. We intend to generalize our method to arbitrary triangle meshes.

6 Conclusions

We have presented a new image-space technique for the detection of collisions and self-collisions for arbitrarily shaped, water-tight objects. Due to the fact, that no pre-processing is required, our method is especially useful in dynamic environments with deformable objects.

We have presented three implementations of our approach. The performance analysis has shown, that our approach can be accelerated with graphics hardware in case of geometrically complex scenes, while the CPU-based implementation provides better performance in case of small environments.

Our method efficiently detects collisions of complex objects with up to 500k triangles. Smaller environments can be handled at rates of up to 500Hz on current PCs. Therefore, the approach can be used in games, in interactive simulation or animation environments.

7 Acknowledgement

This research is supported by the Swiss National Science Foundation. The project is part of the Swiss National Center of Competence in Research on Computer Aided and Image Guided Medical Interventions (NCCR Co-Me).

References

[Bac99] G. Baciu, W. Sai-Keung Wong, and H. Sun. RECODE: an image–based collision detection algorithm. *The Journal of Visualization and Computer Animation*, vol. 10, pp. 181–192, 1999.

[Ber97] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools,* vol. 2:4, pp. 1–13, 1997.

[Bra02] G. Bradshaw, C. O'Sullivan. Sphere-tree construction using medial–axis approximation. *Proc. of ACM Symposium on Computer Animation SCA '02*, pp. 33–40, 2002.

[Bri02] R. Bridson, R. Fedkiw, J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. *Proc. of SIGGRAPH '02,* pp. 594–603, 2002.

[Cam90] S. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transaction on Robotics and Automation,* vol. 6:3, pp. 291–302, 1990.

[Eve01] C. Everitt. Interactive order-independent transparency. *Technical Report*, NVIDIA Corp., 2001.

[Fol90] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes. *Computer Graphics: Principles and Practics.* Addison-Wesley Publishing Company, 1990.

[Gan00] F. Ganovelli, J. Dingliana, C. O'Sullivan, BucketTree: Improving collision detection between deformable objects. *Proc. Spring Conference on Computer Graphics SCCG '00*, Budmerice Castle, Bratislava, 2000.

[Got96] S. Gottschalk, M. Lin, D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Proc. SIGGRAPH '96*, pp. 171–180, 1996.

[Gov03] N. Govindaraju, S. Redon, M. Lin, D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM Graphics Hardware*, 2003.

[Hei03] B. Heidelberger, M. Teschner, M. Gross. Real-time volumetric intersections of deforming objects. *Proc. of Vision, Modeling, Visualization VMV'03,* pp. 461–468, 2003.

[Hof01] K. Hoff, A. Zaferakis, M. Lin, D. Manocha. Fast and simple 2D geometric proximity queries using graphics hardware. *Proc. of Symposium on Interactive 3D Graphics '01,* pp. 145–148, 2001.

[Hub93] P. Hubbard. Interactive collision detection. *Proc. of IEEE Symposium on Research Frontiers in Virtual Reality '93,* pp. 24–31, 1993.

[Kim02] Y. Kim, M. Otaduy, M. Lin, D. Manocha. Fast Penetration Depth Computation for Physically-based Animation. *Proc. of Computer Animation '02,* pp. 23–31, 2002.

[Klo96] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, K. Zikan. Efficient collision detection using bounding volume hierarchies of k–DOPs. *Proc. of SIGGRAPH '96,* pp. 171–180, 1996.

[Kno03] D. Knott, D. Pai. CinDeR: Collision and interference detection in real–time using graphics hardware. *Proc. of Graphics Interface '03*, 2003.

[Lar01] T. Larsson, T. Akenine-Moeller. Collision detection for continuously deforming bodies. *Proc. of Eurographics '01,* pp. 325–333, 2001.

[Lom99] J. Lombardo, M.-P. Cani, F. Neyret. Real-time collision detection for virtual surgery. *Proc. of Computer Animation '99,* pp. 33–39, 1999.

[Mel00] S. Melax. Dynamic plane shifting BSP traversal. *Proc. of Graphics Interface '00*, pp. 213–220, 2000.

[Mys95] K. Myszkowski, O. Okunev, T. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, vol. 11:9, pp. 497–512, 1995.

[Pro97] X. Provot. Collision and self-collision handling in cloth model dedicated to design garment. *Proc. of Graphics Interface '97*, pp. 177–189, 1997.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 3324–3329, 1994.

[Shi91] M. Shinya, M. Forgue. Interference detection through rasterization. *Journal of Visualization and Computer Animation*, vol. 2, pp. 132–134, 1991.

[Sut74] I. E. Sutherland, G. W. Hodgman. Reentrant polygon clipping. *Communications of the ACM*, vol. 17:1 , pp. 32–42, 1974.

[Tes03] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, M. Gross. Optimized spatial hashing for collision detection of deformable objects. *Proc. of Vision, Modeling, Visualization VMV'03,* pp. 47–54, 2003.

[Tur90] G. Turk. *Interactive collision detection for molecular graphics.* Technical Report TR90-014, University of North Carolina at Chapel Hill, 1990.

[Vas01] T. Vassilev, B. Spanlang, Y. Chrysanthou. Fast cloth animation on walking avatars. *Proc. of Eurographics '01*, pp. 260–267, 2001.

[Vol95] P. Volino, M. Courchesne, N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. *Proc. of SIGGRAPH '95*, pp. 137–144, 1995.

[Zac02] G. Zachmann. Minimal hierarchical collision detection. *Proc. of ACM Virtual Reality Software and Technology VRST '02*, pp. 121–128, 2002.

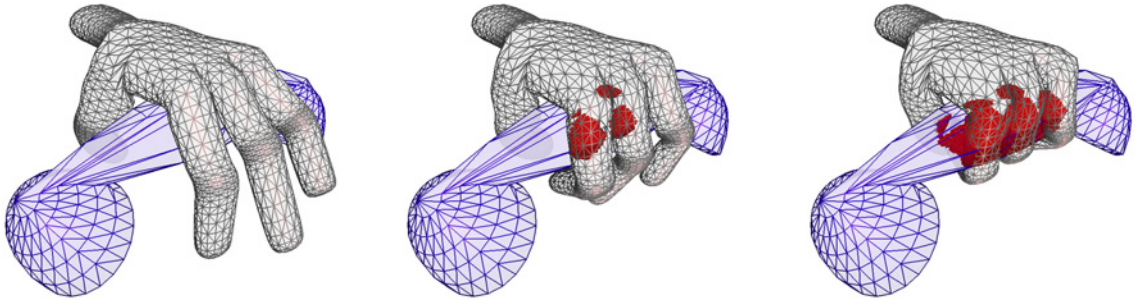

Figure 3: Dynamic animation of a hand with 4800 faces and a phone with 900 faces. Collisions (red) are detected. Refer to Tab. 1 for performance measurements.

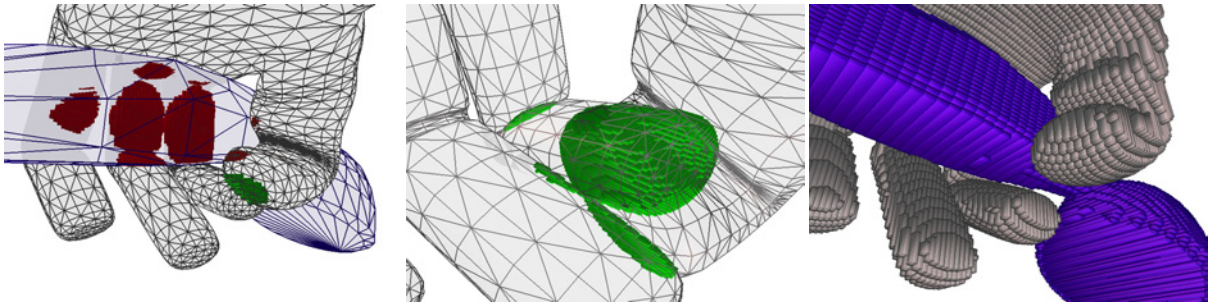

Figure 4: Left: Collisions (red) and self-collisions (green) of the hand are detected. Middle: Self-collisions (green) are detected. Right: LDI representation with a resolution of 64x64. Refer to Tab. 1 for performance measurements.

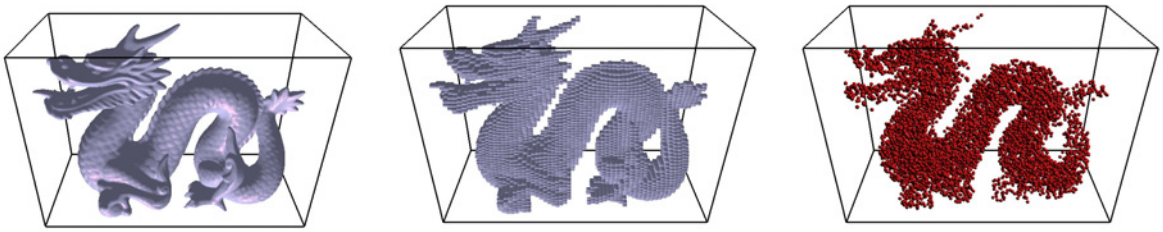

Figure 5: Left: Dragon with 500k faces. Middle: LDI representation with a resolution of 64x64. Right: Particles penetrating the volume of the dragon are detected. Performance measurements are given in Tab. 2.

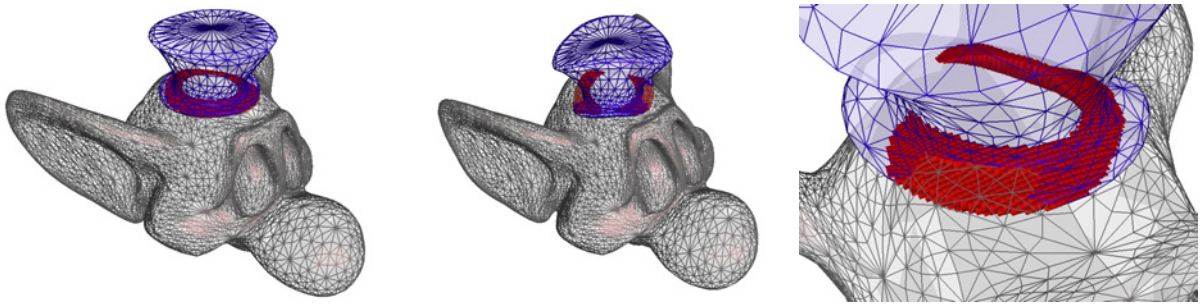

Figure 6: Left, middle: Dynamic animation of a mouse with 15000 faces and a hat with 1500 faces. The intersection volume (red) is shown. Right: Intersection volume with an LDI resolution of 64x64. Performance measurements are given in Tab. 3.