Real–Time Volumetric Intersections of Deforming Objects

Bruno Heidelberger Matthias Teschner Markus Gross

Computer Graphics Laboratory
ETH Zurich

Abstract

We present a new algorithm for the computation of volumetric intersections of geometrically complex objects, which can be used for the efficient detection of collisions. Our approach requires neither expensive setup nor sophisticated spatial data structures and is specifically suitable for handling deformable objects with arbitrarily shaped, closed surfaces. The algorithm employs a Layered Depth Image (LDI) decomposition of the intersection volume.

Currently, we have implemented two types of collision queries. The first one comprises an explicit representation of the intersection volume. The second one computes vertex-in-volume tests. All queries are processed on the LDI-based representation of the intersection volume. Our algorithm is very easy to implement and can be accelerated in graphics hardware by using OpenGL.

1 Introduction

The detection of collisions of geometric objects constitutes a fundamental problem in computer graphics, computational geometry, modeling, robotics, and many other areas. For complex objects, the detection of interfering surfaces is computationally expensive, because the testing of many individual graphics primitives is required. Efficient collision detection algorithms are accelerated by spatial data structures including bounding–box hierarchies, distance fields, or other ways of spatial partitioning. Such object representations are built in a pre–processing stage, and once they are created, perform very well for rigid objects.

Modern physics–based simulations for games and computational surgery require the detection of collisions of objects deforming over time. In the case of deformable objects, the aforementioned acceleration structures have to be updated to adapt geometry changes. While some of these algorithms have been successfully employed in such cases, the acceleration structures constitute an overhead in memory consumption and computational efficiency.

We propose a simple and efficient algorithm based on Layered Depth Images (LDI) [22] as the fundamental representation to accelerate collision detection of rigid and deformable objects. While many existing approaches consider object surfaces, our method is inherently volumetric. Rather than explicitly detecting surface intersections, the LDI provides a discrete representation of the intersection volume and allows for volume–based collision queries. The generation of LDIs can be accelerated by graphics hardware and various optimizations minimize buffer read–backs.

Our algorithm proceeds in three stages. First, we calculate the intersection of axis–aligned bounding boxes of pairs of objects. If an intersection is non–empty, a second stage computes an LDI representation of each object within the bounding–box intersection. Finally, we obtain the overall intersection volume by simple Boolean operations on the LDI representation.

The simplicity of the algorithm, the absence of expensive setup and the ease of implementation in OpenGL makes it especially attractive for real–time simulations of deformable objects.

The contributions of this work can be summarized as follows:

- We present a new algorithm for the computation of volumetric penetrations of geometrically complex, deformable objects in real time. Our method handles arbitrarily shaped, closed surfaces of manifold geometry.
- We currently process two different types of collision queries: The first one includes the explicit computation of the intersection volume of two objects. The second one performs vertex-in-volume tests. Our approach is also robust in cases where one object lies completely inside another object.

- Unlike most existing approaches, our method does not require an involved setup or the implementation of a sophisticated spatial data structure. It is memory efficient, computationally economical and performs on deformable models of up to hundred thousands of primitives.

2 Background and Related Work

Collision algorithms. In graphics, efficient collision detection is an essential component in physically–based simulation or animation [2], [4], including cloth modeling [25], [24], [3]. Further applications can be found in robotics, computer animation, medical simulations, computational biology, and games.

Collision detection algorithms based on bounding–volume (BV) hierarchies have proven to be very efficient and many types of BVs have been investigated. Among the acceleration structures we find spheres [11], [20], axis–aligned bounding boxes [12], [23], oriented bounding boxes [8], and discrete–oriented polytopes [15].

Initially, BV approaches were designed for rigid objects. In this context, the hierarchy is computed in a pre–processing step. In the case of deforming objects, however, this hierarchy must be updated at run time. While effort has been spent to optimize BV hierarchies for deformable objects [17], they still pose a substantial computational burden and storage overhead for complex objects. As an additional limitation for some applications, BV approaches typically detect intersecting surfaces. The computation of the intersection volume requires an additional step.

As an alternative to object partitioning, other approaches employ discretized distance fields as volumetric object representation for collision detection. The presented results, however, suggest that this data structure is less suitable for real–time processing of geometrically complex objects [9].

Recently, various approaches have been introduced that employ graphics hardware for collision detection. In [1] and [19], multi–pass rendering methods are proposed for collision detection. However, these algorithms are restricted to convex objects. In [18], the interaction of a cylindrical tool with deformable tissue is accelerated by graphics hardware. The method is restricted to collisions of simplified surgical tools with object surfaces. [10] proposes a multi–pass rendering approach for collision detection of 2–D objects, while [14] and [13] perform closest–point queries using bounding–volume hierarchies along with a multipass–rendering approach. The aforementioned approaches decompose the objects into convex polytopes. In [16], the intersection of edges with another object can be detected using an image–space approach. However, this approach is not applied to deformable objects, does not compute the intersection volume, and fails in cases, where edges occlude each other in image space.

Layered Depth Images (LDI). LDIs have been introduced as an efficient image–based rendering technique. The LDI data structure essentially stores multiple depth values per pixel. Thus, an LDI can be used to approximate the volume of an object.

A method for generating LDIs is presented, for instance, in the depth–peeling approach to order–independent transparency [5]. This implementation, however, demands a complex setup including two active depth buffers and texture shaders. In contrast, our approach to LDI generation is much simpler by employing a more efficient rendering setup based on plain OpenGL 1.4.

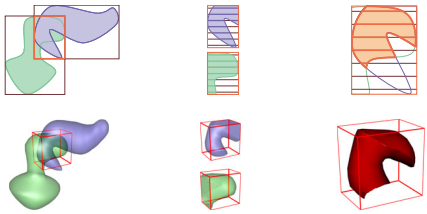
Hardware–based approaches to interactive CSG rendering, e. g. [7] and [21], could also be employed for LDI generation. However, since such CSG rendering implementations have to handle more complex modeling operations they feature more involved rendering setups. Furthermore, these CSG rendering approaches do not explicitly compute the resulting solid.

3 Method

This section presents an overview of our algorithm followed by a detailed description of its three stages.

Input. Our approach takes 3–D closed objects of manifold geometry as an input. Although the approach is not confined to triangular meshes, we require a watertight object surface to perform volumetric collision queries. In addition, a rendering method for the object is needed for LDI generation.

Output. The algorithm computes an explicit representation of the intersection volume as an output. Alternatively, the LDI structure facilitates the fast detection of vertices which penetrate the object.



(a) Stage 1. (b) Stage 2. (c) Stage 3a.

Figure 1: Algorithm overview in 2–D and 3–D. (a) AABB intersection. (b) LDI generation within the VoI. (c) Computation of the intersection volume.

Algorithm overview. Our approach proceeds in three stages.

Stage 1 computes the AABB intersection for a pair of objects. If the intersection is empty, the two objects do not overlap. If it is not, stages 2 and 3 are applied to the AABB intersection volume. We will refer to it as Volume-of-Interest (VoI).

Stage 2 computes two LDIs, one for each object. LDI generation is restricted to the VoI. The depth values of the LDI can be interpreted as the intersections of parallel rays or 3–D scan lines entering or leaving the object. Thus, an LDI classifies the VoI into *inside* and *outside regions* with respect to an object. Likewise, we classify the corresponding intersections into *entry points* and *leaving points*. The concept is similar in spirit to well–known scan–conversion algorithms to fill concave polygons [6], where intersections of a scan line with a polygon represent transitions between interior and exterior.

Stage 3 performs the actual collision detection. Currently, we distinguish two different types of queries: *a)* Both LDIs are combined using Boolean intersection. If the intersection of all inside regions is not empty, a collision is detected. This operation also provides an explicit representation of the intersection volume. *b)* Individual vertices within the VoI of one object are transformed into the LDI of the second object. If such a vertex lies in an inside region, a collision is detected. This query can also be used to check atomic or point–sampled objects for intersection.

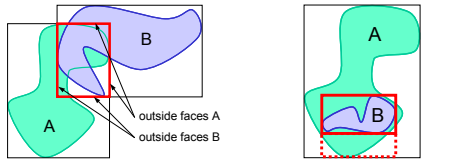
Fig. 1 illustrates the three stages of our algorithm. For a detailed description of all stages, refer to Sections 3.1–3.3.

3.1 AABB Intersection

AABB intersections are computed for pairs of objects. If two AABBs do not overlap, the corresponding objects cannot interfere. Otherwise, the intersection volume provides a VoI, which is considered for further processing. Although AABBs do not provide an optimal, i. e. smallest, bounding volume, they can be computed very efficiently. Furthermore, the intersection of two AABBs is again an AABB which keeps subsequent stages of the algorithm simple.

Alternative BVs, such as oriented bounding boxes [8] or discrete–oriented polytopes [15], provide tighter fitting object approximations. However, the computation of these representations is more involved and the resulting intersection volume can have a more complex shape making it much more difficult for further processing. In addition, such structures are impractical for deforming objects, which require a dynamic adaptation of the bounding box.

In order to simplify and accelerate further computations, the VoI has to meet the following requirements: Firstly, to guarantee proper boundary conditions, we have to render the LDI for each object from the outside. We select one of the six faces of the VoI, the so–called *outside face*, as the near rendering plane for each object (see Fig. 2). If a sorted LDI is generated with respect to this face, entry points and leaving points alternate. For closed objects, the first layer is assumed to contain entry points, the second layer leaving points and so on. Secondly, outside faces for both objects should correspond to opposite sides of the VoI (see Fig. 2(a)). This simplifies the combination of both LDIs for the computation of the intersection volume.



(a) VoI for A and B. (b) Extended VoI to obtain an outside face for A.

Figure 2: The intersection of the AABBs of two objects A and B provides the VoI.

The intersection volume of two AABBs, i. e. the VoI, is bounded by parts of their faces. Fig. 2(a) shows a VoI with corresponding outside faces. In some cases, for instance when one box is entirely within another box, appropriate outside faces for both objects cannot be found. This problem is solved by extending the VoI. If the outside face of one object is fixed, the opposite face of the VoI can be scaled to touch the bounding box of the other object (see Fig. 2(b)).

If several eligible pairs of outside faces exist, we choose the ones with the smallest distance between them. This criterion is a fast heuristic to optimize depth complexity for LDI generation assuming that depth complexity scales with the distance between the two outside faces.

3.2 LDI Generation

LDIs are computed within the VoI for both objects. This provides an explicit volumetric representation discretized at a predefined LDI resolution. The LDIs will be used for subsequent collision queries. The concept of a volumetric representation using LDIs is exemplified in Fig. 3.

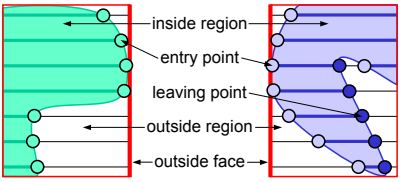


Figure 3: LDIs are computed for objects within the VoI. LDI values are *entry* or *leaving points* and classify the VoI into *inside* and *outside regions*.

In order to generate an LDI, the object is rendered multiple times. The viewing parameters for the rendering process are determined by the selected outside faces of the VoI defining the near planes, and their opposite faces defining the far planes. The remaining faces define top, bottom, left, and right of the rendering volume. Orthographic projection is used for rendering. The LDI resolution, which defines the accuracy of the object approximation, is specified by the user. Objects are rendered n_{max} times for LDI generation, where n_{max} denotes the depth complexity of the relevant part of the object within the VoI. Thus, the computational complexity of the LDI generation is $O(n_{max})$. For details regarding the OpenGL implementation, refer to Sec. 4.

The first rendering pass generates a single LDI layer and computes n_{max}. Depth testing and face culling are disabled. Only the stencil test is employed to discard fragments. The stencil test configuration allows the first fragment per pixel to pass, while the corresponding stencil value is incremented. Subsequent fragments are discarded by the stencil test, but still increment the stencil buffer. Hence, after the first rendering pass the depth buffer contains the first object layer per pixel and the stencil buffer contains a map representing the depth complexity per pixel. Thus, n_{max} is found by searching the maximum stencil value. Note, that the $max()$ computation constitutes a very small computational burden, given the typical LDI resolution.

If $n_{max} > 1$, additional rendering passes 2 to n_{max} generate the remaining layers. The rendering setup is similar to the first pass. However, during the n-th rendering pass, the first n fragments per pixel pass the stencil test and, as a consequence, the resulting depth buffer contains the n-th LDI layer. During these passes, the stencil buffer is not incremented, if the stencil test fails.

Since fragments are rendered in arbitrary order, the algorithm generates an unsorted LDI. For further processing, the LDI is sorted per pixel. Only the first n_p layers per pixel are considered, where n_p is the depth complexity of this pixel. n_p is taken from the stencil buffer as computed in the first rendering pass. If n_p is smaller than n_{max}, layers $n_p + 1$ to n_{max} do not contain valid LDI values for this pixel and are discarded (see Fig. 4).

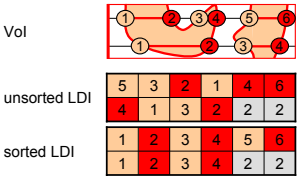


Figure 4: LDI for two pixels, unsorted and sorted with respect to depth values.

Although the order in which the fragments of an object are rendered is arbitrary, our algorithm relies on consistency across the individual passes. Our LDIs store normalized values. For advanced collision queries, additional information, such as view-

ing direction and viewing volume are stored as well.

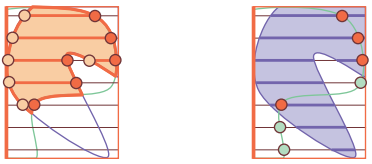
The accuracy of the method can be adjusted. In (x, y)-direction, accuracy corresponds with the user–defined LDI resolution. The accuracy in z-direction is given by the depth–buffer resolution.

Compared to alternative data structures, such as bounding–volume hierarchies or distance fields, the LDI representation is very memory efficient. Both the memory consumption and the performance of the LDI generation depend on the geometric complexity and the depth complexity of the objects. However, our results presented in Sec. 5 suggest, that the number of LDI layers does not exceed reasonable values even for complex objects. Further, collision queries performed on the generated LDIs are very efficient even in the case of a large number of LDI layers.

3.3 Collision Detection

The computed LDIs can be utilized to process a variety of collision queries in an efficient way. Our current implementation comprises two different types of collision queries:

Intersection volume. Two LDIs can be combined to compute an intersection volume. The LDIs for two colliding objects are discretized with the same resolution on corresponding sampling grids, but with opposite viewing directions. Hence, pixels in both LDIs represent corresponding volume spans. Therefore, intersection volumes can be computed by a pixelwise intersection of the inside regions of both LDIs. If the resulting intersection is non–empty, a collision is detected. The sum of all pixelwise intersection regions constitutes a discrete representation of the intersection volume (see Fig. 5(a)).



(a) Intersection volume. (b) Vertex-in-volume.

Figure 5: Two types of collision queries. Pixelwise intersection and testing of the LDIs.

Vertex-in-volume. Individual vertices can also be tested against an LDI. To this end, the vertex is transformed into the local coordinate system of the LDI. If the transformed vertex intersects with an inside region, a collision is detected (see Fig. 5(b)). As demonstrated in Fig. 10 and Fig. 11, this query can be used to compute collisions with atomic objects, such as particles.

4 Implementation

The implementation of VoI computation (stage 1) and collision queries (stage 3) can be easily accomplished by following the descriptions in Sec. 3.1 and Sec. 3.3, respectively. This section describes some details regarding the efficient implementation of the LDI generation (see Sec. 3.2) using core OpenGL 1.4 functionality.

The LDI generation is performed in a multi–pass rendering process following the rendering setup as described in Sec. 3.2. To improve performance, the color buffer and z–tests are disabled. The algorithm just performs a plain rasterization of depth values. Each LDI layer is generated in one rendering pass. Except for the first pass, which computes the depth complexities per pixel, all passes are identical. Our actual OpenGL implementation is very similar to the following pseudo code.

```
rendering setup;
// pass 1 computes n_max and the first LDI layer
glClear(GL_DEPTH_BUFFER_BIT |
        GL_STENCIL_BUFFER_BIT);
glStencilFunc(GL_GREATER, 1, 0xff);
glStencilOp(GL_INCR, GL_INCR, GL_INCR);
render object;
depth complexities ← stencil buffer;
n_max ← max(depth complexities);
layer[1] ← depth buffer;
// passes 2 to n_max for remaining LDI layers
n ← 2;
while n ≤ n_max begin
  glClear(GL_DEPTH_BUFFER_BIT |
          GL_STENCIL_BUFFER_BIT);
  glStencilFunc(GL_GREATER, n, 0xff);
  glStencilOp(GL_KEEP, GL_INCR, GL_INCR);
  render object;
  layer[n] ← depth buffer;
  n ← n + 1;
  end;
```

Each rendering pass actually requires a buffer read–back. Reading back buffers from the GPU, however, can be expensive depending on the actual architecture. We propose two methods for optimizing the data transfer using OpenGL extensions.

Firstly, depth and stencil values of a pixel are usually stored in the same 32–bit word in the framebuffer. This allows to read–back both buffers in a single pass using an OpenGL extension, such as GL_NV_packed_depth_stencil. Secondly, 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 framebuffer. Once finished, the whole framebuffer is read back in a single pass. This optimization reduces the number of read–backs to a maximum of two, assuming that the framebuffer memory is sufficiently large. It reduces stalls in the rendering pipeline and substantially improves the performance of the algorithm.

5 Results

All of the experiments described in this section have been performed on a PC with 2.8 GHz Intel Pentium 4, 2 GB memory, and NVidia GeForce 4 Ti 4600 graphics card with 128 MB memory. Results for the two types of collision queries are presented using five different test objects (see Fig. 6).

Intersection volume. In Tab. 1, computing times for six cases are given. Since the performance of our algorithm depends on the geometrical complexity of the object and on the depth complexities of both objects within the VoI, we have carried out experiments as shown in Fig. 7 and Fig. 8. In the worst case, when both AABBs coincide, the VoI and the depth complexity are maximal. Hence, both LDIs are computed for the entire object. The performance of these cases can be derived from Tab. 2, where timings for the LDI generation for entire objects are given.

An additional scenario with a very complex intersection volume is illustrated in Fig. 9. The Knot and the Dragon have 12k faces and 520k faces, respectively. Depth complexities within the VoI are 7 and 8. With an LDI resolution of 128x128 the intersection volume can be computed in 256ms.

Vertex-in-volume. Computing times for the vertex-in-volume test are given in Tab. 2. For each object, 100,000 arbitrarily positioned vertices within the AABB of an object are tested. Fig. 10 illustrates the experiment for one of the test objects.

The vertex-in-volume test is shown in Fig. 11.

Objects	Faces	Depth Complexity	Stage 1+2 [ms]	Stage 3a [ms]	Fig.
Knot / Rabbit	12k / 50k	3 / 1	19	1	7
Knot / Rabbit	12k / 50k	4 / 6	56	1	8
Rabbit / Rept.	50k / 110k	7 / 5	36	1	7
Rabbit / Rept.	50k / 110k	4 / 2	48	1	8
Rept. / Santa	110k/150k	1 / 3	25	1	7
Rept. / Santa	110k/150k	8 / 6	114	1	8

Table 1: Computation of the intersection volume using various object pairs as shown in Fig. 7 and Fig. 8. The resolution of the LDI is 128x128. The depth complexities are given for both objects within the VoI. Stage 1 and 2 compute the AABBs, the VoI, and both LDIs. Stage 3a generates the intersection volume. In the case of rigid objects, stages 1 and 2 can be pre–computed.

The simulation environment includes 20,000 particles and the deforming Santa with 10k faces. The object deformation, the particle dynamics, the collision detection and response can be computed and rendered in real time with 29 fps.

Object	Faces	Depth Complexity	Stage 1+2 [ms]	Stage 3b [ms]
Knot	12k	8	11	26
Rabbit	50k	8	29	27
Reptile	110k	12	83	28
Santa	150k	8	79	26
Dragon	520k	10	307	27

Table 2: Vertex-in-volume test for various objects. The VoI computed in stage 1 coincides with the AABB of the object. Stage 2 computes the LDI representation for the entire object with a resolution of 128x128. In stage 3, 100,000 vertices with arbitrary positions within the AABB of the object are tested. In the case of rigid objects, stages 1 and 2 can be pre–computed. Fig. 10 illustrates the test for one object.

6 Conclusions

We have presented a fast, hardware–accelerated method for volumetric collision detection. As a cen-

tral acceleration structure our algorithm computes an LDI that approximates the intersection volume. Unlike many existing collision detection schemes, our method does not require expensive setup or pre–processing and is, hence, specifically suited for deformable objects and dynamic simulation. Currently, we have implemented two different types of collision queries and demonstrated that the method performs at interactive rates up to very complex geometry.

As a limitation, our collision detection approach is restricted to objects with closed, watertight surfaces, since it relies on the notion of inside and outside.

7 Ongoing Work

While we compute an explicit representation of the intersection volume, additional queries, such as penetration depth or closest point, are needed for advanced collision response.

At this point, self–collisions are not detected. Actually, our algorithm could be extended towards self–collisions by treating the front and back faces separately during the rendering process. This would allow us to efficently detect an invalid ordering of front and back faces.

8 Acknowledgements

This project is funded by the National Center of Competence in Research “Computational Medicine” (NCCR Co–Me).

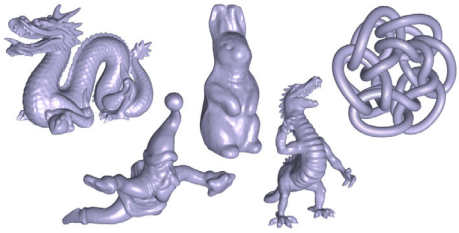


Figure 6: Test objects: Dragon, Santa, Rabbit, Reptile, Knot.

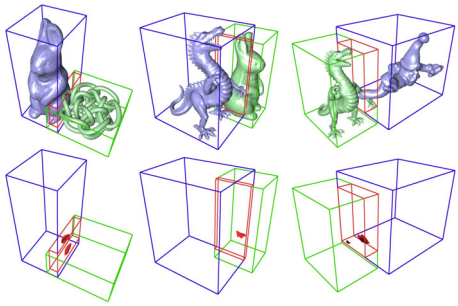


Figure 7: Computation of the intersection volume. AABBs, VoI, and intersection volume are shown for three examples with simple intersections.

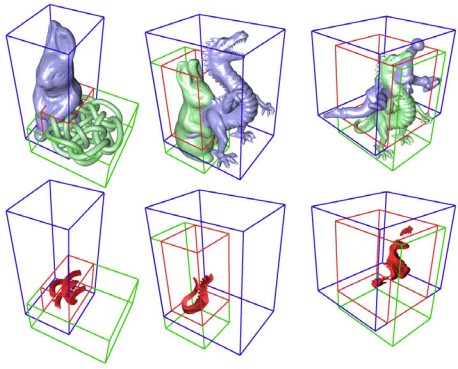


Figure 8: Computation of the intersection volume. AABBs, VoI, and intersection volume are shown for three examples with complex intersections.

References

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

[2] D. Baraff. Collision and contact. *SIGGRAPH 2001 Course Notes*, 2001.

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

[4] G. Debunne, M. Desbrun, M.-P. Cani, and A. H. Barr. Dynamic real-time deformations using space & time adaptive sampling. *Proceedings of SIGGRAPH ’01*, pp 31–36, 2001.

[5] C. Everitt. Interactive order–independent transparency. Technical report, NVIDIA Corporation, 2001.

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

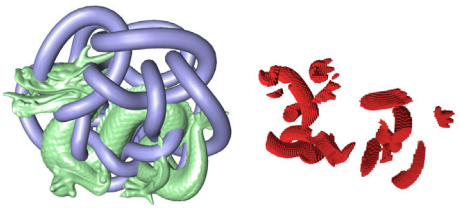


Figure 9: Intersection volume for Knot and Dragon.

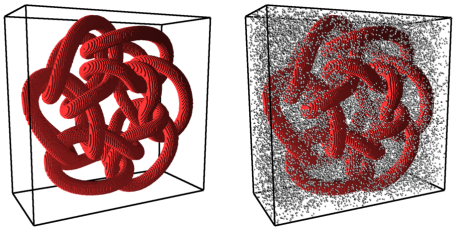


Figure 10: Vertex-in-volume test. The LDI representation is computed for the entire object. VoI and AABB coincide. The vertex-in-volume test is processed for 100,000 arbitrarily positioned vertices.

[7] J. Goldfeather, J. P. M. Hultquist, and H. Fuchs. Fast constructive–solid geometry display in the pixel–powers graphics system. *Proceedings of SIGGRAPH '86*, pp. 107–116, 1986.

[8] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. *Proceedings of SIGGRAPH '96*, pp. 171–180, 1996.

[9] G. Hirota, S. Fisher, and M. C. Lin. *Simulation of non–penetrating elastic bodies using distance fields.* Technical Report TR00-018, UNC, 2000.

[10] K. E. Hoff III, A. Zaferakis, M. C. Lin, and D. Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. *Proceedings of 2001 Symp. on I3D*, pp. 145–148, 2001.

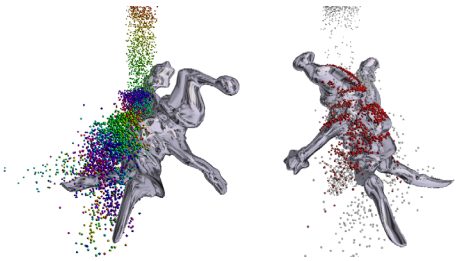


Figure 11: Deforming Santa and particles. Red color on the right–hand side indicates actual or recent contact.

[11] P. M. Hubbard. Interactive collision detection. *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, pp. 24–31, 1993.

[12] M. Hughes, C. DiMattia, M. C. Lin, and D. Manocha. Efficient and accurate interference detection for polynomial deformation and soft object animation. *Proceedings of Computer Animation*, pp. 155–166, 1996.

[13] Y. J. Kim, K. E. Hoff III, M. C. Lin, and D. Manocha. Closest point query among the union of convex polytopes using rasterization hardware. *Journal of Graphics Tools*, to appear, 2003.

[14] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Fast penetration depth computation for physically-based animation. *Proc. of SIGGRAPH Symp. on Computer Anim.*, pp. 23–31, 2002.

[15] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k–DOPs. *Proceedings of the SIGGRAPH '96*, pp. 171–180, 1996.

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

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

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

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

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

[21] A. Rappoport and S. Spitz. Interactive boolean operations for conceptual design of 3–D solids. *Proceedings of SIGGRAPH '97*, pp. 269–278, 1997.

[22] J. Shade, S. Gortler, Li wei He, and R. Szeliski. Layered depth images. *Proceedings of SIGGRAPH '98*, pp. 231–242, 1998.

[23] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.

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

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