

Interactive Collision Detection for Complex and Deformable Models Using Programmable Graphics Hardware

Wei Chen Huagen Wan Hongxin Zhang Hujun Bao Qunsheng Peng
State Key Lab of CAD&CG, Zhejiang University, Hangzhou, 310027, China
{chenwei,hgwan,zhx,bao,peng}@cad.zju.edu.cn

ABSTRACT

In this paper we present an interactive collision detection algorithm for complex and deformable objects. For two target models, our approach rapidly calculates their region of interests (ROI), which is the overlapping of their axis aligned bounding boxes (AABBs), in CPU. The surfaces of both models inside the ROI are then voxelized using a novel GPU-based real-time voxelization method. The resultant volumes are represented by two 2D textures in video memory. The collision query is efficiently accomplished by comparing these 2D textures in GPU. The algorithm is robust to handle arbitrary shapes, no matter geometric models are convex or concave, closed or open, rigid or deformable. Our preliminary implementation achieves interactive frame rate for complex models with up to one million triangles on commodity desktop PCs.

Categories and Subject Descriptors

I.3.3 [Computer Graphics]: Picture—*image generation*;

I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—*object representations*

General Terms

Algorithms

Keywords

Collision detection, graphics hardware, voxelization, deformation

1. INTRODUCTION

Collision detection plays an important role in computer graphics and its related areas, such as electronic entertainment and educational applications. Without it, the reality and immersion feelings of the user would be significantly weakened. In the past decades, collision detection has been

well studied, and various approaches have been developed [21, 27]. However, interactive collision detection is still a very challenging problem with growing scene complexity and diversity. On the one hand, though there are approaches for detecting collisions between deformable objects [14, 16, 26], most current collision detection methods are focused on handling rigid objects which limits their applicability. On the other hand, many typical algorithms have shown good performance for regular geometry models. But their efficiency decreases dramatically with highly complex geometry models.

In this paper, we present a novel approach for interactive collision detection between complex models as well as deformable objects. The key to achieve interactive performance in our context is a highly efficient voxelization algorithm exploiting the newest features of programmable graphics hardware. The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes our new method. Section 4 presents experimental results. Section 5 concludes the paper and proposes future work.

2. RELATED WORK

At a broad level, collision detection methods can be classified into two categories: object-space methods and image-space methods.

Most object-space methods employ a variety of software techniques such as bounding volume hierarchies, space partitions, and other optimization ways [3, 4, 6, 7, 8, 9, 19, 20, 13, 24, 25, 26, 30, 32, 35]. As these optimization techniques inevitably involve additional data structures besides model geometries, object-space methods inherently depend on the model representation scheme and the scene complexity is often limited.

There are also many methods exploiting the computational capabilities of graphics hardware for collision detection [1, 31, 33, 34]. These methods follow a simple mode to perform computations in image space and often involve no preprocessing. Early work mainly concentrates on utilizing depth and stencil buffers for collision query. Whereas, they only work for closed objects, or even convex objects, and require frequent accessing of frame buffers, which is costly on commodity graphics hardware. Recent researches in image-space approaches make elegant use of streaming rendering facilities and programmability of graphics hardware [16, 17, 18, 23, 28, 36]. Most of them can handle collision queries for deformable objects but limit to closed objects and suffer the same overhead as frame buffer accessing is concerned.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

VRST '04, November 10-12, 2004, Hong Kong

Copyright 2004 ACM 1-58113-907-1/04/0011 ...\$5.00.

More recently, Govindaraju *et al.* [14] introduce a fast collision detection approach among multiple deformable objects based on the concept of potentially colliding set (PCS) using graphics hardware. However, for complex objects, this algorithm has to group the individual triangles of objects into sub-objects for PCS computation as a preprocess, which reduces its generality.

Specifically, volumetric representation of objects can be used for fast collision detection [5, 11, 12, 15, 29, 38]. Among them, the algorithm proposed by Heidelberg *et al.* employs a view-dependent Layered Depth Image (LDI) decomposition of the intersection volume to compute the volumetric intersections of 3D shapes [16]. The algorithm needs depth sorting of the read results at each frame and its performance is not only dependent on scene complexity, but also on scene depth complexity. In addition, the construction of LDI can not avoid accessing the frame buffer as well.

3. COLLISION DETECTION METHOD

3.1 Overview

Our approach takes two models with arbitrary shapes as inputs. The interferences between them are checked only on their boundaries. Our solution to collision detection is based on the observation that two objects have interferences if and only if their surfaces intersect¹. Likewise, their volumetric representations have shared voxels given that the representations are accurate enough. As a result, we first compute the region of interest (ROI), i.e. the overlapped region of their axis aligned bounding boxes (AABBs). We then perform voxelization for surfaces of both objects inside the ROI, and query collisions by checking whether there is at least one common voxel occupied by both resultant volumes. Note that, the voxelization is a computation-intensive process and requires lots of computing resources [2, 10, 22, 37]. Our main contribution lies in that we propose a scan-conversion-based voxelization method which is feasible for voxelization of surfaces in real-time for complex models by exploiting the newest features of programmable graphics hardware. Figure 1 illustrates the pipeline of our collision detection algorithm. We will elaborate each stage in the following sections.

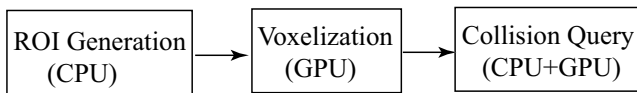


Figure 1: Our collision detection pipeline.

3.2 ROI Generation

Obviously interferences between two objects take place only in the intersected part of their bounding volumes, and hence the voxelization and collision query need not be performed in the whole scene. We simply traverse all vertices of both models and get their AABBs respectively. The intersection of the objects’ AABBs, which is also an AABB, forms the ROI (region of interests). We have implemented a CPU version and a GPU version for calculating the ROI, finding that their efficiencies are almost the same. We choose

¹The case where one object lies completely inside the other is neglected.

the CPU implementation since it is more flexible and controllable. Further, the objects are traversed again to retrieve the primitives (triangles) that are partially or entirely located in the ROI. These valid primitives are then classified according to the scheme described in the next section and sent to dynamic vertex buffers which are placed in AGP all the time. The ROI generation and the retrieval of valid primitives can be finished at a very low cost in CPU while reducing the redundancy and complexity of the further process in GPU.

3.3 Real-Time Voxelization

In standard rasterization hardware, triangles are scan-converted into a 2D frame buffer. Only the frontmost fragments are kept in the frame buffer storing the rasterization results. Whereas, voxelization is a 3D rasterization procedure and hence a discrete voxel space is required. The voxel space consists of an array of voxels that store all voxelized values. It can be represented as 2D or 3D textures in graphics hardware. Since writing directly to 3D texture is not supported in mainstream graphics card on PC platform, we choose to encode the volume in 2D texture. We call the texture *worksheet* as it records all voxelization information. Note that each texel in the graphics card typically consists of four components for red, green, blue and alpha channels respectively. Depending on the bit-depth of each voxel, one texel can represent one or multiple voxels. For instance, an 8-bit red component can store 8 voxels for binary voxelization. The conversion from volume space to worksheet invokes an encoding procedure called *texelization*. The storage of worksheet equals the size of the volume. For example, one 2048×2048 texture with four components is sufficient for binary voxelization at the resolution of 512^3 . Note that the width and length of a worksheet may be very large and it might be divided into multiple patches. Each patch has the same width and length as that of the volume which corresponds to a slab of the volume along some axis direction (Figure 2). In high volume resolution cases, multiple worksheets are required. To simplify the explanation, we suppose that one worksheet is used.

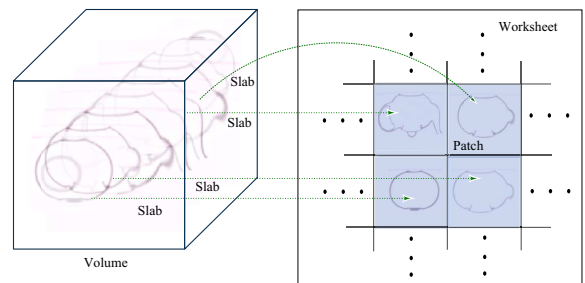


Figure 2: The worksheet versus the slabs of the volume.

The conversion from the triangles to the discrete voxel representation is accomplished in programmable graphics hardware. The volume is generated slab by slab. In other words, the worksheet is filled patch by patch. For each slab, only the triangles that intersect the slab are processed. Each chosen triangle is rasterized against an axis direction along which it has the maximum projection area. The position of each voxel is transformed to its 3D volume coordinates

immediately. These coordinates are used to find the correct position in the worksheet. Note that, the discrete voxel space is only a virtual concept and is not explicitly represented. In order to add a voxel to the worksheet, a blending operation is carried out at corresponding location. When all triangles are processed, the worksheet encodes the discrete voxel space.

The 2D rasterization in standard graphics hardware involves a 2D linear interpolation process. If a triangle is parallel to the rasterization direction, the interpolation process results in a line segment in the discrete voxel space. Therefore, a triangle should be rasterized along the direction that is most parallel to its orientation. And three *directional sheet buffers* are used as intermediate space during the rasterization and texelization procedures. Each sheet buffer represents a part of the discrete voxel space. After these sheet buffers are accomplished, an additional process is performed to transcode them to the final worksheet (Figure 3). In this stage, each element is first transformed to the discrete voxel space and then encoded to the appropriate texel in the worksheet. Actually, the worksheet reformulates the slabs of the volume along a desired axis direction.

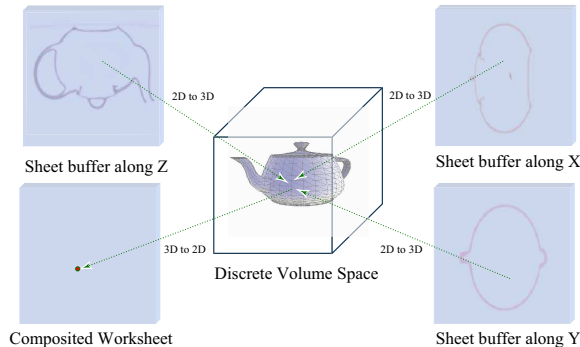


Figure 3: Synthesis of three directional sheet buffers.

To sum up, the voxelization algorithm consists of three stages as follows.

- **Rasterization** The triangles are rasterized to the discrete voxel space.
- **Texelization** Each voxel is encoded and accumulated in some directional sheet buffer.
- **Synthesis** Three sheet buffers are transcoded to the worksheet representing the final volume.

3.4 Collision Query

Collision query is carried out as a subsequent stage following the voxelization process. The voxelization results of both models are represented by two textures in video memory. To accomplish collision query, a screen-filling rectangle is rendered with these textures and each pixel of them is compared pairwise. If there are common voxels shared by the two textures, the objects are regarded as colliding each other. The collision query is accomplished by programmable fragment shader combining the *occlusion-query* functionality. Occlusion query can report the total number of colliding voxels rapidly. By this way, the collision status between the

two models is reported. To obtain detailed interference information, we can encode the identification of each primitive instead of just the occupancy during the voxelization process. In this way, exact information about which triangles of both models are colliding can be retrieved from the 2D textures by frame buffer accessing. However, due to the limited bandwidth between host memory and video memory, it is hard to achieve in this mode interactive frame rates yet.

3.5 Extension to Handling Deformable Models

As illustrated in Figure 1, our collision detection pipeline makes no assumption about the motion of objects or any spatial/time coherence between successive frames. Therefore, it can be easily extended to detecting collisions for deformable models. For deformable models, deformation takes place at each frame. The computation of deformation can be accomplished in either CPU or GPU. We propose to put the computation of deformation in CPU because it is still costly and inconvenient to implement complex deformation with programmable graphics hardware. Considering deformable models, we can extend our collision detection pipeline to that of as illustrated in Figure 4.

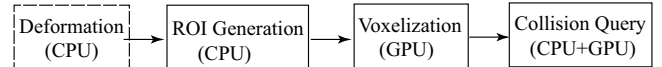


Figure 4: Collision detection pipeline with extension to deformable models.

4. RESULTS AND DISCUSSION

Our algorithm has been implemented on a PC with a single 2.4 GHz Pentium IV CPU and 2GB RAM. An ATI Radeon 9800 Pro graphics card with 256MB RAM is configured in our experiments.

We implemented all examples using binary voxelization under the volume resolution 256^3 . Figure 5 demonstrates the stages of the collision detection for two deformable models. The dolphin model has 564 triangles and 284 vertices. The cylindrically shaped deformable object has 6,804 triangles and 3,416 vertices. Their intersected parts are shown in the middle row of Figure 5. The bottom row of Figure 5 illustrates two different frames where 538 and 358 colliding voxels are detected respectively.

In our algorithm, the computational overhead of collision detection mainly consists of the time costs for the voxelizations of two models as well as collision query. While the computation cost of voxelizing models is dependent on model complexities and the volume resolution, the cost of collision query is constant under the same volume resolutions (e.g., around 15ms for a 256^3 volume in our testing cases) since it is performed in image space. Table 1 lists the voxelization performance in milliseconds for four complex models ranging from about 650k triangles to about 1.8M triangles, as illustrated in Figure 6. According to Table 1, for two complex models with about 1 million triangles each, collision detection can be accomplished in about 175ms (80+80+15). We have tested 50 rigid models and the timing for collision detection between models with different complexities is plotted in Figure 7. For example, if both models have 800k triangles, collision detection can be accomplished in 155ms.

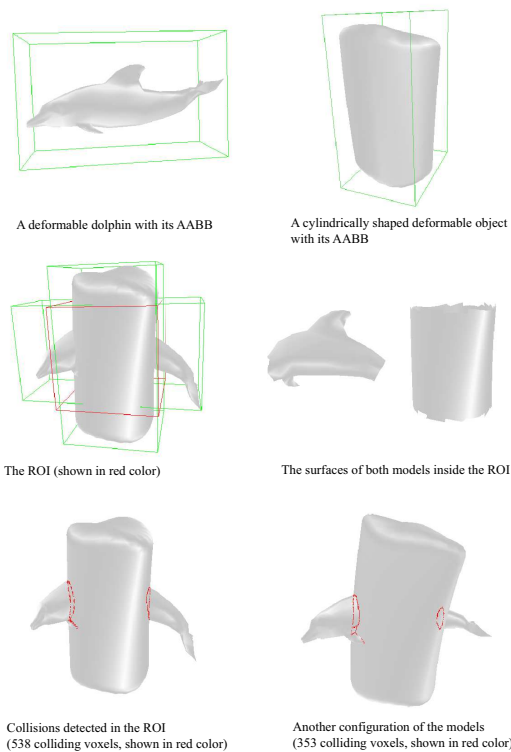


Figure 5: Illustrations of collision detection between two deformable models. The whole procedure is accomplished in 115ms, among which 40ms are spent on deformation computations, 60ms are spent on voxelizing the models and 15ms are spent on querying the results.

Model	#Triangles	#Vertices	Voxelization Time
Hand	654,666	327,694	68ms
Dragon	871,326	439,370	70ms
Buddha	1,087,514	550,868	78ms
Blade	1,765,388	898,796	95ms

Table 1: Voxelization timing of different models. All models are voxelized in $256 \times 256 \times 256$ resolution.

The accuracy of our algorithm depends on the volume resolution. Suppose that the length of the ROI is l , and the ROI is uniformly voxelized into 256 voxels along each axis, then the maximum error between the accurate model and its voxelized counterpart is less than $\sqrt{3}l/512$. We think this kind of accuracy is enough for typical graphics-related applications.

Our method is similar to the LDI method proposed in [16] for collision detection. However, our method differs with the LDI method in at least following three aspects.

- Our algorithm traverses the object only once while the LDI method has to render the object the same times as the scene depth complexity.
- Our regular sampled volumes ensure the order of sampled points and avoid re-sorting and region filling to the depth lists.
- Our method is view independent and the memory con-

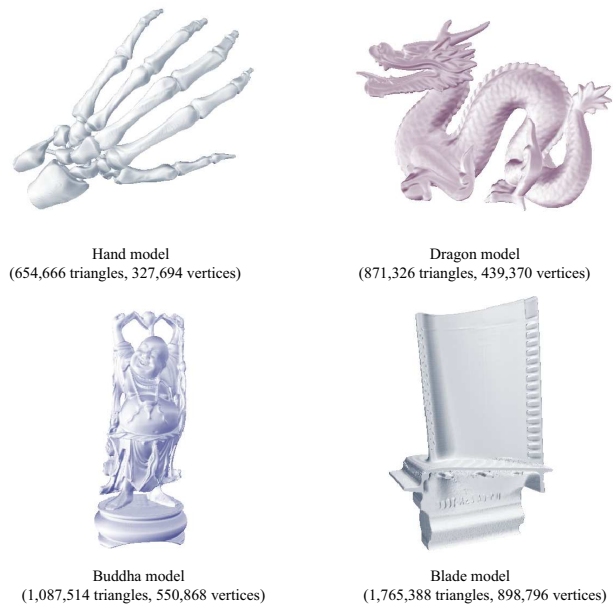


Figure 6: Illustrations of four complex models.

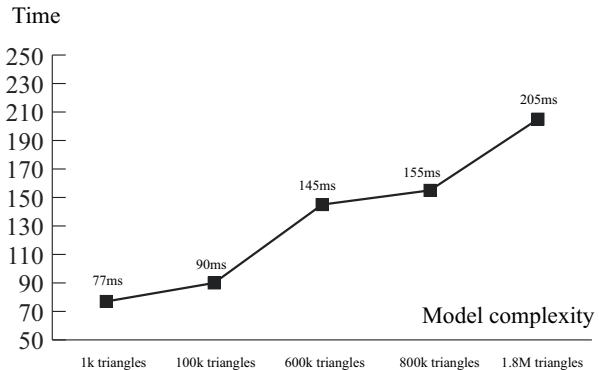


Figure 7: Timing statistics for collision detection between rigid models with different complexities. Volume resolution: 256^3 .

sumption for voxelizing objects is fixed for a certain volume resolution, while the LDI method is view dependent and needs to allocate the storage space dynamically.

5. CONCLUSION AND FUTURE WORK

The approach presented in this paper takes a pair of general polygonal objects as input, and detects collisions between their surfaces. It makes no assumption about the motion of objects or any spatial/time coherence between successive frames. As a result, it can be used to detect collisions for rigid objects as well as deformable models. The algorithm is robust due to its simplicity and the credibility of the voxelization results. More specifically, the algorithm is efficient even for highly complex models.

One drawback of our approach is that the accuracy of collision detection is limited to the resultant volume resolutions. While this sounds like a severe weakness, the volume resolutions under our current implementation are sufficient

for most graphics related applications. Moreover, the limitation can be removed by recording the identifications of primitives (triangles) when voxelizing objects, and using our method as an on-line pre-processing step before accurate intersection tests between primitives (triangles).

We will focus on the following two issues in the future work. First, we will develop a robust scan filling method to realize solid voxelization, which facilitates the identification of the case where an object is embedded entirely in another object. Second, we will extend the algorithm to collision query between different representations of objects such as point based models, parametric surfaces, implicit surfaces and CSG models. In fact, our framework for collision detection does not put any restriction on model representation schemes as long as voxelizations of the models can be performed in real-time.

6. ACKNOWLEDGMENTS

We would like to acknowledge Mr.Zhongding Jiang for his careful proofreading and Mr.Zhao Dong for his help. This work is partially supported by 973 program of China (No.2002CB312100), NSF of China for Innovative Research Groups (No.60021201), NSF of China (No.60103003,60103017, 60303028), Zhejiang Provincial Natural Science Special Fund for Youth Scientists' Cultivation (No.R603046).

7. REFERENCES

- [1] G. Baciú, S. K. Wong, and H. Q. Sun. An image-based collision detection algorithm. In *Proceedings of Pacific Graphics 1998*, pages 497–512, 1998.
- [2] S. Beckhaus, J. Wind, and T. Strothotte. Hardware-based voxelization for 3d spatial analysis. In *Proceedings of CGIM 2002*, pages 15–20, 2002.
- [3] G. Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.
- [4] G. Bergen. A fast and robust gjk implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.
- [5] M. Boyles and S. Fang. Slicing-based volumetric collision detection. *Journal of Graphics Tools*, 4(4):23–32, 1999.
- [6] S. Cameron. Enhancing gjk: computing minimum and penetration distances between convex polyhedra. In *Proceedings of IEEE International Conference of Robotics and Automation*, pages 3112–3117, 1997.
- [7] K. Chunk and W. Wang. Quick collision detection of polytopes in virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 125–131, 1996.
- [8] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proceeding of ACM Interactive 3D Graphics Conference*, pages 189–196, 1995.
- [9] S. Ehmann and M. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Proceedings of Eurographics 2001*, pages 500–510, 2001.
- [10] S. Fang and H. Chen. Hardware accelerated voxeliation. *Computers and Graphics*, 24(3):433–442, 2000.
- [11] N. Gagvani and D. Silver. Shape-based volumetric collision detection. In *Proceedings of the 2000 IEEE symposium on volume visualization*, pages 57–61, 2000.
- [12] S. Gibson. Beyond volume rendering: visualization, haptic exploration, and physical modeling of voxel-based objects. In *Proceedings of 6th Eurographics Workshop on Visualization in Scientific Computing*, pages 10–24, 1995.
- [13] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: a hierarchical structure for rapid interference detection. In *Proceedings of ACM SIGGRAPH*, pages 171–180, 1996.
- [14] N. K. Govindaraju, S. Redon, and D. Manocha. Cullide: Interactive collision detection between complex models in large environments using graphics hardware. In *ACM SIGGRAPH/Eurographics Workshop On Graphics Hardware*, pages 25–32, 2003.
- [15] T. He and A. Kaufman. Collision detection for volumetric objects. In *Proceedings of the 8th conference on Visualization*, pages 27–35, 1997.
- [16] B. Heidelberger, M. Teschner, and M. Gross. Real-time volumetric intersections of deforming objects. In *Proceedings of Vision, Modeling, Visualization*, pages 461–468, 2003.
- [17] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 145–148, 2001.
- [18] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast 3d geometric proximity queries between rigid and deformable models using graphics hardware acceleration. Technical Report TR02-004, 2002.
- [19] P. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.
- [20] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-collide: accelerated collision detection for vrml. In *Proceedings of VRML Conference*, pages 119–125, 1997.
- [21] P. Jimenez, F. Thomas, and C. Torras. 3d collision detection: A survey. *Computers and Graphics*, 25(2):269–285, 2001.
- [22] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *IEEE Computer*, 26(7):51–64, 1993.
- [23] Y. Kim, M. Otaduy, M. Lin, and D. Manocha. Fast penetration depth computation using rasterization hardware and hierarchical refinement. UNC-CH Technical Report TR02-014, 2002.
- [24] Y. Kitamura, A. Smith, H. Takemura, and F. Kishino. A real-time algorithm for accurate collision detection for deformable polyhedral objects. *Presence*, 7(1):36–52, 1998.
- [25] J. T. Klosowski, J. T. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transaction on Visualization and Computer Graphics*, 4(1):21–36, 1998.
- [26] T. Larsson and T. Akenine-Moller. Efficient collision detection for models deformed by morphing. *The Visual Computer*, 9(2):164–174, 2003.
- [27] M. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.
- [28] J. C. Lombardo, M.-P. Cani, and F. Neyret. Real-time collision detection for virtual surgery. In *Proceedings of Computer Animation*, pages 33–39, 1999.
- [29] W. McNeely, K. Puterbaugh, and J. Troy. Six degree-of freedom haptic rendering using voxel sampling. In *Proceedings of ACM SIGGRAPH*, pages 401–408, 1999.
- [30] B. Mirtich. V-clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, 1998.
- [31] K. Myszkowski, O. G. Okunev, and T. L. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9):497–512, 1995.
- [32] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. In *Proceedings of*

Eurographics, pages 278–288, 2002.

- [33] J. Rossignac, A. Megahed, and B. D. Schneider. Interactive inspection of solids: cross-sections and interferences. In *Proceedings of SIGGRAPH 1992*, pages 353–360, 1992.
- [34] M. Shinya and M. C. Fogue. Interference detection through rasterization. *The Journal of Visualization and Computer Animation*, 2(4):131–134, 1991.
- [35] A. Smith, Y. Kitamura, H. Takemura, and F. Kishino. A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In *Proceedings of Virtual reality annual international symposium*, pages 136–145, 1995.
- [36] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. *Computer Graphics Forum*, 20(3):260–267, 2001.
- [37] S. Wang and A. Kaufman. Volume-sampled 3d modeling. *IEEE Computer Graphics & Applications*, 14(5):26–32, 1994.
- [38] D. Zhang and M. Yuen. Collision detection for clothed human animation. In *Proceedings of Pacific Graphics 2000*, pages 328–337, 2000.