Practical Collision Detection in Rendering Hardware for Two Complex 3D Polygon Objects

Jonathan Dinerstein
Sorenson Media, Inc.
jon@sorenson.com

Parris Egbert
Brigham Young University
Egbert@cs.byu.edu

Abstract

We present a technique to perform real-time collision detection of two complex 3D polygon objects using commodity and workstation rendering hardware. This technique is O(n), and performs favorably in comparison to existing traditional methods in many circumstances. On PCs or workstations, this technique can be performed almost entirely in rendering hardware. This frees general CPU cycles for other processing tasks. Alternatively, more CPU cycles can be used for collision detection, freeing rendering hardware cycles for other tasks. This technique is also highly customizable to specific applications.

1: Introduction

Over the past several years, a few techniques have been presented for using rendering hardware to speed up collision detection queries [1,3,9]. These techniques greatly outperform traditional methods, but are only useful for certain limited applications. In general, the strengths of these existing hardware techniques include that they are simple to implement in OpenGL, and are faster than traditional collision detection techniques for specific applications. Further, they are stable at O(n), so performance does not degenerate when objects are in close proximity. These hardware techniques are especially efficient for dynamic surfaces, where traditional techniques require reconstruction of bounding volumes for all new geometry.

However, these rendering hardware-assisted techniques suffer from significant limitations. Most are limited to strictly convex objects, or one object must be simple enough to be approximated by a view volume. Some techniques only support 2D geometry and/or collisions. Some require features only available in high-end workstation rendering hardware. Further, some techniques require multiple rendering/analysis passes, and/or significant CPU use.

Because of their limitations, these techniques are only practical for a few applications. Indeed, some methods are even specialized to one specific use. This has kept hardware-assisted collision detection from being generally accepted as a practical technique.

We have developed a new technique that alleviates many of the weaknesses of previous methods. Our criteria for this new technique are that it performs well on all classes of rendering hardware, from commodity to workstation. It should also work well with any 3D polygon models. It can have the workload custom distributed between the CPU and rendering hardware, based on the application's needs and requirements. Further, can be performed almost entirely in either rendering hardware or CPU, freeing either resource for other tasks. This new technique should also have an adjustable speed/quality tradeoff, customizable for specific applications. It should also be fast, so that real-time frame rates are achieved. Preferably, it would be faster than traditional techniques in many circumstances.

We believe that by achieving these criteria, our technique is practical and useful for many applications. Also of benefit to our technique, recent advances in rendering hardware have greatly surpassed CPU improvements. As a result, many applications now fully utilize CPU while leaving rendering hardware mostly idle. The opportunity to utilize rendering hardware for collision detection is compelling, as it can free general CPU cycles for other processing tasks.

2: Previous techniques

Traditional collision detection methods have been described as *object-space* techniques [1], since the location of geometry in space is used directly in the computation. Alternatively, rendering hardware-assisted methods have been described as *image-space* techniques, since they reduce the problem to a 2D image that may have additional information associated with it (such as Z-buffer, stencil, accumulation buffer, etc). Modern optimized object-space techniques are generally O(n log n), but can degenerate to $O(n^2)$ worst case for objects in close proximity. Image-space techniques, however, are stable at O(n) where n is the number of polygons or the number of pixels in the frame buffer. This linear complexity is very valuable for collision detection between objects of many polygons. Further, image-space

Proceedings of the 20th Eurographics UK Conference (EGUK'02)
0-7695-1518-5/02 $17.00 © 2002 IEEE

techniques have no need for complex bounding volumes used by object-space techniques, and therefore are especially valuable for dynamic surfaces.

In this paper, hardware-assisted collision detection always refers to image-space techniques. Traditional collision detection always refers to object-space techniques.

A hardware-assisted collision detection technique has been developed for use in virtual surgery [9]. It is very fast and simple. However, it makes *very* strong assumptions about the problem domain. It is designed for virtual orthoscopic surgery, where there is a very small incision in the patient's abdomen. One end of the surgical tool is fixed at the incision, point P_0. The tip of the surgical tool can be located at any point P inside the patient. Collision detection is performed by approximating the tool with a view volume, then rendering any organs that the tool may collide with. If any polygons are drawn, a collision has taken place. This technique has been shown to be about 150 times faster than RAPID [5] for dynamic surfaces (such as a deformed organ). While useful for this specific application, it is unlikely this technique could be applied elsewhere.

Another hardware-assisted technique, RECODE [1], has been developed for computing collisions between convex objects. RECODE is faster than traditional techniques for objects of many polygons, but slower for few polygons. This is due to advanced uses of the stencil buffer not supported in hardware. Another weakness of RECODE is that only convex objects can be used. Non-convex objects must be subdivided into convex sub-objects and tested for collisions separately. However, RECODE has been shown to be very accurate. In fact, in certain tests, RECODE finds collisions missed by RAPID [5] and I-Collide [4].

Other hardware-assisted techniques include volumetric collision detection [3], proximity determination [6,7], and motion planning for robots [8]. However, a discussion of such is beyond the scope of this paper.

3: The new practical technique

Our technique is designed to maintain all general strengths of previous hardware-assisted collision detection methods, while alleviating many key weaknesses that have kept these methods from being generally accepted. The strengths and weaknesses of previous methods, and the criteria for our new technique, are listed in the introduction.

We use the color buffer to determine if a collision has taken place. Rasterization naturally converts 2D/3D geometry into discrete 2D space (the color buffer). This information alone is enough to know whether, in two dimensions, a collision has taken place. However, the third dimension (Z) can also be discretized into the color buffer by intelligently using all color available. We do this by rendering each discrete depth location a different 1-bit color, and OR'ing rendered pixels together. Each color buffer pixel now contains all information about which objects occupy it and at what depths.

Step 1 of our technique is to determine if the objects may collide using either aligned or oriented bounding boxes. Step 2 is to set the view volume equal to the intersection of the aligned bounding boxes of the two objects. Step 3 is to render the objects to a 32-bit RGBA color buffer. To retain depth (Z) information, a 1D texture is used for each object. Two color channels (R,G and B,A respectively) are used by the texture of each object. Each texture has 16 entries, 1 bit used in each entry (e.g. 0x0001, 0x0002, 0x0004…0x8000). This provides depth-based slices in a single pixel, coded by color. By performing a bitwise OR on all incoming rendered pixels (fragments) and pixels already in the frame buffer, all previously rendered information is retained. Finally, in step 4, after both objects are rendered, the color buffer is examined to determine if any pixel contains both objects in the same depth slice. In summary, the steps of this technique are:

1. Test two objects' aligned or oriented bounding boxes for collision.
2. Set the view volume equal to the intersection of the two objects' axis-aligned bounding boxes.
3. Render both objects, using a 1D texture to discretize depth according to color.
4. Examine the color buffer to determine if any pixel contains both objects at the same depth.

The X and Y dimensions are discretized into pixels, while Z is discretized into color-coded depth slices. Therefore, the two objects collide if they both occupy the same pixel and depth slice. No information besides the color buffer is needed to perform collision detection. Our discretization of space is shown in figure 1.

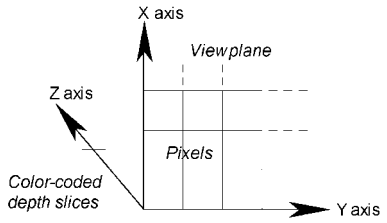

Figure 1. Discretization of space by our technique. The X and Y axis are located in the view plane (discretized into pixels). The Z axis is discretized through color-coded depth slices. A collision has occurred if both objects occupy the same discrete location (IE pixel and depth slice).

This technique can be easily implemented in OpenGL. Further, most available rendering features should not be used. For example, Z-buffer, back-face culling, dithering, etc. All polygons not clipped by the view volume should be drawn, and 1-dimensional texturing should be the only effect applied.

Texture coordinate generation is used to access the 1D textures in a depth-wise manner similar to [2]. The Z coordinate of transformed/clipped vertices is simply normalized to a range of [0,1]. In OpenGL, this is done very easily. Shown below is an example, where the view volume near and far clipping planes are respective distance D_1 and D_2 from the camera:

```
float TexGenParams[] =
{
    0.0f,
    0.0f,
    -1.0,(D2-D1),
    -D1/(D2-D1)
};

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,
            GL_EYE_LINEAR);
glTexGenfv(GL_S, GL_EYE_PLANE, TexGenParams);
glEnable(GL_TEXTURE_GEN_S);
```

The bitwise OR operation is supported in newer commodity rendering hardware, such as the ATI Radeon line of video cards. It is supported in virtually all workstation-class rendering hardware. In OpenGL, this feature is used by:

```
glEnable(GL_COLOR_LOGIC_OP);
glLogicOp(GL_OR);
```

Analysis of the color buffer is very simple and fast to perform, but must be done by the CPU (the imaging extensions of OpenGL 1.2.1 and later do not support this type of analysis). Further, analysis can be effectively done in SIMD, such as Intel MMX/SSE/SSE2 and Motorola AltiVec. To transfer the pixels to client memory, glReadPixels( ) is usually appropriate.

3.1: Error compensation

Our technique is prone to error in two ways. Each of these will be discussed in turn:

1. The frame buffer and color-coded depth slices are discrete.
2. Polygons which degenerate to line segments are not rendered.

3.1.1: Discrete error. We tend to find some false collisions and not find some true collisions. However, the error rate is very low (about 1.55% of cases for two complex objects in very close proximity using a 32x32 frame buffer, 16 slices—consider table 1). Therefore, for applications where such error can be tolerated, this fast, low-resolution test may provide a complete solution.

However, it is undesirable to always render at a high resolution and number of slices to get more accurate collision detection. Ideally, a small frame buffer (32x32) and 16 slices would be used for first-pass collision detection to accept/reject possible object pairs. This could then be refined with a more accurate test. We accomplish this fast first-pass test with offset surface generation.

Offset surface generation is performed on a polygon model by translating all vertices a fixed distance along the associated outward-facing vertex normals. This makes the surface "swell". Alternatively, negative offset surface generation makes the surface "shrink". If an offset surface is generated by translating a large enough distance, it is known that no collisions will be missed (although some additional false collisions may be found). Alternatively, for a negative offset surface generated by translating a small enough distance, it is known that no false collisions will be found. Offset surface generation is demonstrated in figure 2.

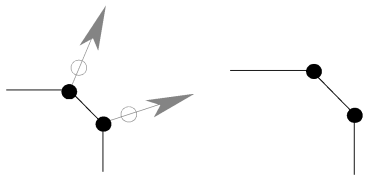

Figure 2. Offset surface generation performed on a polyline. All vertices are translated a fixed distance along the outward-facing vertex normal.

For most applications, collisions happen less often than "non-collisions". Based on this assumption, a fast first-pass collision test can be performed at low resolution using positive offset surfaces. If it seems that a collision took place, a more robust test can then be performed. This additional test should be done by examining only the volume where the collision took place – we will call this the *collision volume*.

The *collision volume* is found based on what pixels at what depth reported a collision. This usually localizes the volume to examine so greatly that very accurate collision detection can be performed at the original low frame buffer resolution. Further, each collision volume might, after analysis, produce a smaller collision volume. However, if the collision volume is fully refined and further accuracy is required, the frame buffer resolution can be increased. Also, multiple renderings can be performed with the view volume iteratively occupying different depths, such that the total number of depth slices can be increased (16 slices for each rendering). It is important to note that if the collision volume is large (e.g.

collision was reported in many pixels), it is likely that a collision must have taken place and assumptions may be made.

Ultimately, completely accurate collision detection is not practical with our technique because it is discrete. For example, we found that to match RAPID in accuracy, we have to use a 256x256 framebuffer, 128 depth slices, and refine the collision volume 3 times. This is 12 renderings, which is cost prohibitive. If absolutely accurate collision detection is required, a traditional technique should be used. However, our technique can still be used as a rapid test for deciding if collision probably did happen, and in what collision volume the alleged collision took place. The collision volume can then be used as an optimization in the traditional method, limiting what portions of the objects need to be tested.

3.1.2: Degenerate polygon error. Polygons which degenerate to line segments (with respect to pixels in the frame buffer) are not rendered. This includes any polygon whose plane is orthogonal to the view plane. We have found this to not be an issue for most models, especially those with no large flat surfaces. However, for models with large flat surfaces, this problem can be easily dealt with by specially positioning the camera. As an example, consider a desk. The top surface will not be drawn if orthogonal to the view volume. Therefore, the camera must be positioned so that this degenerate case is avoided.

3.2: Collision response

To compute collision response, it is usually necessary to know where on a surface a collision took place. Traditionally, this has been determined by finding which polygons collided. Using our technique, this same general location can be immediately determined from the collision volume (or actual collision points inside the collision volume). This is done by performing an inverse camera/object transformation to return to object coordinates. OpenGL provides this functionality through gluUnProject().

However, our technique has no support for finding exactly which polygons collided. If this information is required, a traditional collision detection technique should be used to compute this. The collision volume can be used to optimize this search (clipping polygons that are known to not collide).

3.3: Dynamic collision detection

Collision detection between moving objects is a more difficult problem, because a fourth dimension is introduced. If very small motion has occurred between frames, it can usually be ignored. However, if large motion has taken place, it must be dealt with. For example, a fast moving object may completely pass through a static object from one frame to the next.

In 2-body techniques (such as RAPID, RECODE, and our technique), motion is usually dealt with by testing for collisions iteratively between the time of the previous frame and current frame. Our technique can perform this iterative test in a standard exhaustive method. However, if collisions are not expected to take place often, this test can be optimized by rendering both objects at all times to test, without examining or clearing the frame buffer. Then, after the objects have been rendered at all times, examining the frame buffer once. This test cannot determine if a collision has absolutely taken place, only that the two objects' paths cross at some point. If a crossing has occurred, then a full iterative search must be performed.

3.4: Performance features of the technique

The processing requirements of our technique are split between the CPU and rendering hardware based on two criteria: (1) what resolution/number of slices are rendered, and (2) is a traditional collision detection technique used at some point. Due to the high speed of modern rendering hardware, the number of pixels in the frame buffer does not significantly impact rendering time (unless the frame buffer is exceedingly large). However, it does impact the bus and CPU, as it must be copied to local memory and examined there. The number of slices has a more direct impact on the rendering hardware, as every set of 16 slices requires a separate rendering.

If rendering hardware is always used (traditional techniques never used), most of the workload will be on the rendering hardware regardless of whether the frame buffer/number of slices is large or small. Alternatively, if only a quick first-pass test is done with rendering hardware and then robust testing with traditional techniques, more of the load will be on the CPU.

If offset surfaces are to be used, we recommend computing them once offline. They can then be reused frame-by-frame without any computational cost. We have found that translating the vertices 1% of the dimensions of the model works well in general. Certainly, this is not as robust as generating a new offset surface based on the dimensions of the current collision volume being tested, but gives good results and is more computationally feasible.

This technique is highly tunable for specific applications. For example, if collisions are expected to be rare, a quick first-pass test can be done at a low resolution with positive offset surface models. Alternatively, if collisions are expected to happen often, a robust test can be performed initially by rendering at a high resolution.

Also, as discussed previously, the workload can be greatly varied between the CPU and rendering hardware.

We recommend the use of OpenGL display lists with this technique. A display list is a recording of OpenGL instructions and data received from the client program. Display lists are usually stored in video memory, and only a single function call is required to render them. This helps to reduce CPU and bus impact. Further, if offset surfaces are not used, the same display lists could be used for both rendering and collision detection.

We also recommend using a pbuffer with this technique, a feature added to OpenGL in version 1.3. A pbuffer is an off-screen frame buffer located in video memory (so it is still fully hardware optimized). With a pbuffer, collision detection can be performed in rendering hardware while being invisible to the user. A similar feature exists in other rendering systems such as Microsoft's Direct3D.

4: Results

As discussed in [1], it is very difficult to quantitatively compare collision detection techniques. There are no standard benchmarks to do so, and each technique has unique strengths and weaknesses. Therefore, we have performed experiments similar to those in [1], as they have a historic place in the literature. These tests involve 2 or more moving rigid polygon objects. However, our tests are performed on complex 3D models, whereas RECODE is limited to convex models.

All experiments have been performed on a PC with 256 MB RAM, 1.7 GHz Pentium 4 processor, ATI Radeon 8500 video card, and Windows 2000 OS. We selected this system because it is generally mid-range with respect to hardware on which this technique may be used.

We have chosen to compare our technique against RAPID [5]. This is because of the problem-domain similarities between it and our technique: 2-body, any topology of polygons supported, and do not use motion-time coherence.

We have performed an experiment similar to [9], where dynamically deformed surfaces are tested for collision. Our image-space technique vastly outperforms RAPID, since RAPID must reconstruct the oriented bounding boxes for the model before being able to test for collision. However, our results were very similar to those in [9], so we refer the reader there for further detail.

4.1: Accuracy

Accuracy is a concern for our technique, since it is discrete. To test this, we performed a series of experiments where we randomly translated two complex polygon models, placing them within very close proximity to each other (their bounding boxes intersected). As an example, consider figure 3. In our experiments, collision occurred In approximately 20% of test cases. Different models were used in each experiment. Models included objects such as passenger jet airplanes, anatomy, and plants. Offset surfaces were not used.

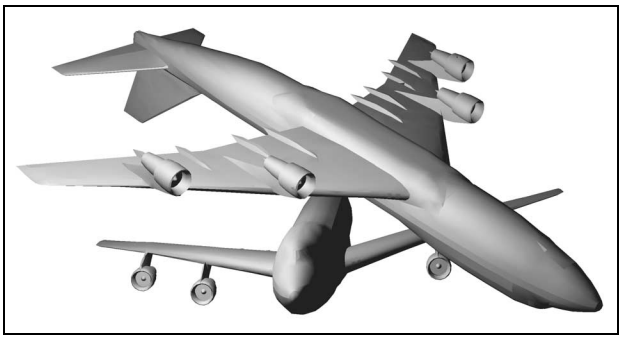

Figure 3. Our accuracy experiments involved randomly positioning two complex polygon models. For example, these two passenger jet airplane models. Each model contains approximately 6000 triangles.

We have found this technique to be surprisingly accurate, even at low resolutions. For fast, somewhat accurate collision detection, a frame buffer resolution of 32x32 with 16 depth slices works well. For medium speed and accuracy, a 64x64 frame buffer, 16 depth slices, and 1 collision volume refinement. For lower speed and high accuracy, a 128x128 frame buffer, 16 or 32 depth slices, and 1 or 2 collision volume refinements. Frame buffers larger than 128x128 do not provide much additional accuracy. We have assembled our findings in table 1.

It is important to note that error only occurs when two surfaces are in very close proximity. In other words, if two objects are clearly separate or clearly collide, error never occurs. Error can only occur when the two surfaces are in the same pixel and depth slice. This reduces the impact of the error for many applications. For example, it may be sufficient to know that two surfaces are very, very close to determine that collision has taken place.

Discretization can result in not detecting some true collisions and detecting some false collisions. For framebuffer resolutions of 128x128 and above, almost all error is due to the discretization of depth (Z) information. However, this is not easily resolved, since every set of 16 color-coded depth sliced requires a rendering. Further, increasing the number of depth slices has been shown to provide small benefit. We recommend using collision volume refinement to reduce error. Only if error must be further reduced should more depth slices be rendered.

4.2: Performance

We have performed an experiment where the performance of our technique and RAPID were compared for complex 3D models. For this test we used two models of passenger jet airplanes (shown in figure 3), each containing approximately 6,000 polygons. The models were randomly positioned such that they were within close proximity (their axis-aligned bounding boxes intersected). The rate of collision was varied to gather information on how stable each technique is. The results of this experiment are compiled in figure 4.

It is important to note that we have purposefully created a test situation that favors RAPID over our technique. We used a 1.7 GHz processor (higher-end processor), but only an ATI Radeon 8500 video card (mid-range for PC's, designed for games). Further, in this experiment, we only guaranteed that the objects' axis-aligned bounding boxes intersected. If so, we had our technique perform a full collision test. RAPID, however, uses oriented bounding boxes. Therefore, in many cases, RAPID was able to immediately detect that no collision took place. Additionally, we did not have our technique perform gross culling of subobjects outside of the view volume—all geometry was transformed and clipped one polygon at a time by the rendering hardware. We purposefully introduced these discrepancies into our experiment so that the performance comparison of our technique would not be best case.

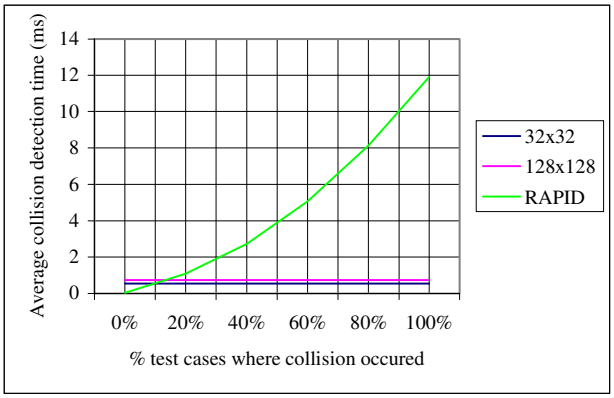

Figure 4. Performance comparison of our technique and RAPID. Note that collision time is averaged. The standard deviation of collision time with RAPID was very large, whereas it was very small with our technique. As can be seen, our technique is computationally stable. We did not perform any collision volume refinement with our technique.

As can be seen in figure 4, our technique is computationally stable, while RAPID grows significantly more expensive as collision becomes more common. Further, even when collision is somewhat rare, our technique is still faster than RAPID overall. Through the use of oriented bounding boxes and gross subobject culling, our technique might be as fast as RAPID for very low collision rates.

In figure 5 is shown a performance comparison of RAPID and our technique with respect to number of polygons.

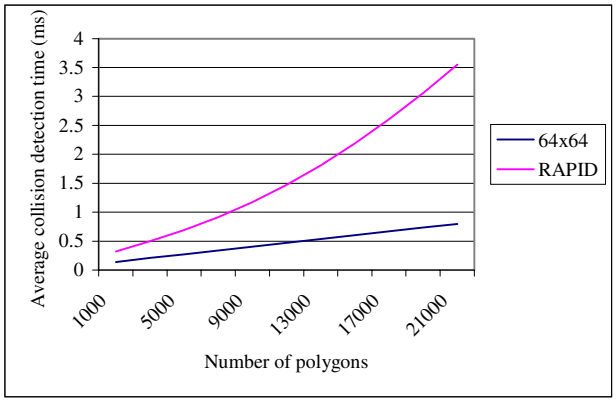

Figure 5. Average collision detection time of the jet airplane models tessellated to different numbers of polygons. As in previous experiments, the two objects were randomly positioned in close proximity. Collision occurred in 30% of test cases. The X axis is the total number of polygons between the two models. As can be seen, our technique is more computationally stable than RAPID.

5: Discussion

The technique presented in this paper has been shown to be very fast, highly customizable, general, and robust. It suffers from only one degeneracy (which is easily avoided), large flat surfaces orthogonal to the view plane. Further, this technique is stable at O(n). It requires no bounding volumes, and is therefore especially efficient for dynamic surfaces when compared to traditional techniques.

This algorithm may be implemented in other rendering systems, as long as bitwise OR of fragments and destination pixels is supported. It is also noteworthy that our technique will directly work with any geometric primitives supported by OpenGL (e.g. lines, complex polygons, NURBS, etc).

No practical resolution and number of slices used with our technique can make it as accurate as a floating-point exhaustive polygon-to-polygon test. However, the degree of error is determined by the descretization of rendering, and therefore is adjustable according to the tolerance of a specific application. If the greatest

accuracy possible is required, a traditional technique should be used.

Currently, our technique has no feature to detect exactly which polygons collided. However, this information is often required, since the collision volume specifies where on the models the collision took place. If polygon intersection information must be gathered, we recommend that a traditional collision detection technique be used to find these polygons. The collision volume can be used to clip polygons that are known to not collide, optimizing the search. Such clipping can be hardware accelerated on workstations using OpenGL's feedback feature.

Most rendering features in OpenGL are not needed to perform our technique. Because of this, extra-fast rendering is possible with most hardware. Further, unlike other hardware assisted techniques, only the color buffer is used, keeping video memory and bus usage down.

With the phenomenal advances in rendering hardware over the past few years, many applications no longer fully utilize this hardware. However, CPU has not advanced at anywhere near this pace, and is still a major bottleneck in many cases. Because of this processing discrepancy, rendering hardware-assisted collision detection is more attractive than ever.

References

[1] G. Baicu, W. Wong, and H. Sun. "RECODE: An Image-Based Collision Detection Algorithm." *Journal of Visualization and Computer Animation*, 10(4): 181-192 (1999).

[2] M. Bailey and D. Clark. "Using ChromaDepth to Obtain Inexpensive Single-Image Stereovision." *Journal of Graphics Tools*, 3(3): 1-9 (1998).

[3] M. Boyles and S. Fang. "Slicing-Based Volumetric Collision Detection." *Journal of Graphics Tools*, 4(4): 23-32 (1999).

[4] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments." In *Proceedings of Symposium on Interactive 3D Graphics 1995*, 189-218.

[5] S. Gottschalk, M. Lin, and D. Manocha. "OBBTree: A Hierarchical Structure for Rapid Interference Detection." In *Proceedings of SIGGRAPH 1996*, 171-180.

[6] K. Hoff III, T. Culver, J. Keyser, M. Lin, D. Manocha. "Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware." In *Proceedings of SIGGRAPH 1999*, 277-286.

[7] K. Hoff III, A. Zaferakis, M. Lin, D. Manocha. "Fast and Simple 3D Geometric Proximity Queries Using Graphics Hardware." In *Proceedings on 2001 Symposium of Interactive 3D Techniques*, 145-148.

[8] J. Lengyel, M. Reichert, B. Donald, D. Greenburg. "Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware." In *Proceedings of SIGGRAPH 1990*, 327-335.

[9] J. Lombardo, M. Cani, and F. Neyret. "Real-Time Collision Detection for Virtual Surgery." *In Computer Animation 1999*.

[10] M. Woo, J Neider, T. Davis, and D. Shriener. *OpenGL Programmer Guide Third Edition*. Reading, MA: Addison Wesley Developer's Press, 1999.

	16X16	32X32	64X64	128X128	256X256
No refining (1 rendering total)	2.97%	1.55%	1.63%	1.72%	2.07%
Refined once (2 renderings total)	3.02%	0.86%	0.45%	0.33%	0.26%
Refined twice (3 renderings total)	3.16%	0.85%	0.27%	0.16%	0.08%
Refined twice, 32 depth slices in last rendering (4 renderings total)	3.35%	0.81%	0.26%	0.12%	0.07%

Table 1. This table reports the percent of test cases where error occurred for collision detection of two complex models in very close proximity. Collision occurred in about 20% of test cases. The standard deviation of error for the different models tested was very low.