GI-COLLIDE — Collision Detection with Geometry Images

Bedřich Beneš *
ITESM CCM

Nestor Goméz Villanueva†
ITESM CCM

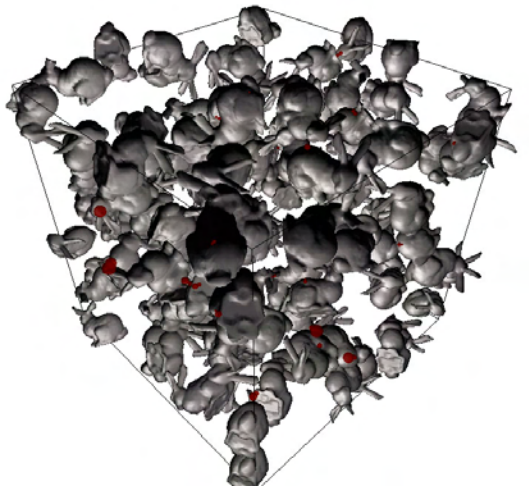

Figure 1: 512 bunnies tested for collisions

Abstract

A new collision detection algorithm GI-COLLIDE is presented. The algorithm works with geometry images and codes bounding spheres as a perfectly balanced mip-map–like hierarchical data structure. The largest geometry image in the hierarchy stores the bounding spheres of the quadruples of the vertices from the input geometry image. Center of the sphere is calculated as the center of the corresponding min-max AABB and stored as the pixel's RGB value. The alpha value stores the sphere radius. In this way each level represents bounding spheres of the previous level. The upmost level of the hierarchy is the bounding sphere of the entire object. Collisions between objects are detected by the standard tree traversing and checking the overlaps among the bounding spheres. The exact collision is detected by a triangle-triangle test at the lowest level. The bounding sphere coding is implicit. A sphere at any level can be found efficiently only by indexing the geometry image. Once objects are represented as geometry images the collision detection can be performed efficiently using directly this representation and it is not necessary to use any other. The results show that this method is efficient and works well even for large objects. We have compared GI-COLLIDE with the most important collision detection techniques.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation – Viewing Algorithms— [I.3.7]: Computer Graphics—Three Dimensional Graphics and Realism – Virtual Reality

Keywords: Geometry Images, Collision Detection, Real-Time Rendering, Time-Critical Rendering

*e-mail: bedrich.benes@itesm.mx
†e-mail: ngomez@itesm.mx

1 Introduction

Geometry images were recently introduced to Computer Graphics [Gu et al. 2002]. Their main advantage is the efficient way they code geometric information. The sampled geometry is stored in a perfectly regular form – as an image. A geometry image can be compressed by the compression schemes used for images; it can be efficiently stored and processed by graphics hardware, efficiently transmitted over network, etc. Many algorithms exploiting advantages of geometry images have recently been introduced, but we have not found any approach focusing on collision detection. There are collision detection algorithms that work very well with another representations, but we do not know any algorithm that works *directly* with geometry images. We present a new algorithm named GI-COLLIDE. The advantage of our approach is in exploiting the properties of geometry images. Suppose, we have objects represented as geometry images. If we want to detect collisions among them we do not need to use another representation, but we can use the geometry images directly.

We propose a mip-map–like [Moller and Haines 2002] hierarchical structure of the geometry images for an efficient collision detection. The input geometry image is used to calculate the bounding sphere tree (BS-tree). The deepest level holds in the RGB triplet the center of the sphere that is calculated as the center of the AABB of the four vertices from the input data. The alpha value stores the bounding sphere radius. This information is sufficient to create bounding sphere of the four vertices. The upper levels are computed and stored in the same way. The hierarchy can be either precomputed and stored together with the image or computed when the geometry image is read or created.

Geometry images will change if the object rotate and that is why we use spheres as the bounding volumes, keeping in mind that they, in general, do not provide a good fit to the objects.

There are two general approaches to detect collisions or interpenetrations between two objects: collision detection (also known

as interference detection or contact determination) and proximity queries (also known as penetration depth computation). The collision detection techniques automatically report a geometric contact that occurs or has actually occurred, giving the point of collision. Proximity queries algorithms report the collision as well, moreover they also provide extra information such as intersection detection, tolerance verification, exact or approximate minimum distance, or disjoint contact determination. GI-COLLIDE is a collision detection technique i.e., the algorithm report weather a collision between two objects has occurred or not.

GI-COLLIDE uses the precomputed hierarchical structure and always starts by comparing the upmost levels of the objects hierarchies, i.e., the bounding spheres of the entire objects. In the case of overlap, there is also potential collision and the structures of both objects are recursively traversed down. At the deepest level the exact triangle-to-triangle test is applied [Moller 1997]. Figure 2 shows an example of the bounding spheres hierarchies refinement as an object gets closer to another one.

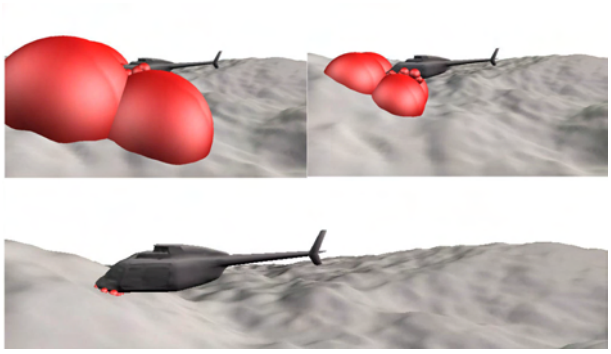

Figure 2: Sphere hierarchy refinement during the collision detection

Various stop-conditions can be applied. We can terminate the traversation when one collision is detected. Another choice is to continue and detect all possible collisions. We also describe a time-critical version of this collision detection algorithm. The difference to the non-time-critical version is that the collision test must be finished within a user-defined time. The test returns the best possible result that is detected within this time.

Geometry images are normally used to store one object, so we describe the object-to-object test in this paper. To detect collisions among more objects we have incorporated our test into the I-COLLIDE [Cohen et al. 1995] algorithm that uses an efficient space subdivision technique and frame to frame coherence.

2 Previous Work

2.1 Geometry Images

Gu et. al. [Gu et al. 2002] introduced geometry images; data structure that efficiently codes and stores the object's geometry. This representation captures geometry as a two-dimensional array of quantized discrete points. Other surface signals, like normal vectors and colors, are stored in similar arrays using the same surface parameterization. Texture coordinates information is implicit in the image. Methods for remeshing geometry images where introduced in [Gu et al. 2002] based on techniques for remeshing with (semi) regular connectivity [Eck et al. 1995; Lee et al. 1998]. The most important is the heuristics that cuts a surface into a disk that maps the surface onto a square.

Praun and Hoppe [Praun and Hoppe 2003] introduced a technique for direct parameterization of a genus-zero surface onto a spherical domain. This technique provides geometry remeshing, level of detail, morphing, compression, and smooth surface subdivision. The main limitation of these techniques is the restriction to a non-manifold geometry.

Sander et al. [Sander et al. 2003] presented an atlas construction that maps the surface piece wisely onto charts of arbitrary shape. Reduced parameterization distortion was achieved and a zippering algorithm that creates a watertight surface was presented. Up to this point, geometry images were used only to store geometry information, normal vectors, or animated geometry.

Geometry images with simple boundary symmetries were later used to store spline surface control points [Losasso et al. 2003]. A bicubic surface is evaluated using a subdivision scheme, and the regular structure of the geometry image makes this computation well suited for graphics hardware. This scheme also provides a smooth level of detail transitions from a subsampled base octahedron to a smooth model.

Various approaches for collision detection, proximity, and interpenetration between two objects exist. Next, we present the most important recently published algorithms and approaches.

2.2 Collision Detection

Collision detection and collision response generation are extremely important in many Computer Graphics applications, such as video games, Virtual Reality, interaction with CAD models, etc. Several techniques for objects collision detection have recently been introduced. Computing a test over all polygons in a model is exact, but very expensive in terms of computation. The majority of the algorithms exploit hierarchical data structures of bounding volumes reducing the time complexity of the algorithms.

OBB-trees were used in the fundamental paper of Gottschalk et al. [Gottschalk et al. 1996] for computing collision detection. A hierarchical bounding volume data structure uses tight-fitting oriented bounding box trees. Two such trees are traversed and tested for overlaps between oriented bounding boxes based on the separating axis theorem. A test for box overlap that takes about hundred operations was presented.

Klosowski et al. [Klosowski et al. 1998] introduced hierarchies of discrete orientation polytopes (k-DOPs) for efficient collision detection. The principal advantage of this method is a better fit of the k-DOPs compared to the previously used bounding volumes. The BV-tree traversation is similar to the previously mentioned.

He [He 1999] shows a combination of OBBs and k-DOPs called QuOSPOs. This approach provides a tight approximation of the original model at each level, and supports a conservative overlap-checking algorithm.

Bradshaw and O'Sullivan [Bradshaw and O'Sullivan 2004] presented a Sphere-Tree construction that approximates both convex and non-convex objects. Their technique is based on the Dynamic Medial Axis Approximation and the fit reported is better than the one reported by Hubbard [Hubbard 1996].

The algorithms mentioned so far work in the geometrical object-space. Baciu et al. [Baciu and Wong 2003] proposed a method for collision detection that works in the image space. This algorithm

breaks the object-space collision detection bottleneck by distributing the computational load onto the graphics pipeline. The image-space interference test is based on projecting the object geometry onto the image plane and performing analysis in a dimensionally reduced space. The hybrid collision detection algorithm first calculates the separating vector to find non-colliding objects. If no separating vectors are found, the image-based algorithm is invoked for further testing of interference.

CULLIDE [Govindaraju et al. 2003] is a GPU-based collision detection algorithm that also works in the image space. Objects that are tested for collision can be deformable and breakable. The collision detection algorithm calculates the potentially colliding sets that are obtained from visibility queries. This allows performing the detection only to parts and sub-parts of objects.

2.3 Proximity Queries

Collision detection algorithms report zero–one, or zero-more collisions. Proximity queries detect collisions and also distance of two objects, penetration depth, spanning distance, etc. Result of a proximity query can be exact, approximate, or Boolean. Moreover, the majority of the proximity query techniques share one common approach; they exploit frame-to-frame coherence. Here we present overview of the most important proximity query algorithms and techniques recently published.

I-COLLIDE [Cohen et al. 1995] is a two level approach based on pruning multiple object pairs using bounding boxes and performing exact collision detection between selected pairs of polyhedral models. To determine whether a collision has occurred or not, the algorithm uses the closest feature algorithm of [Lin and Canny 1991]. This algorithm reduces the number of required tests that would be calculated by checking for bounding boxes object overlaps. Once the collision is detected, the precise collision test is performed for all involucrate polygons.

An incremental proximity query detection algorithm between two general B-rep objects was published [Ponamgi et al. 1995]. The algorithm combines a hierarchical representation of AABBs with an incremental frame-to-frame computation. Coherence between successive instances determines the number of interacting object features. It localizes the interference regions on the convex hulls of each pair of objects. The features associated with the regions are stored in a precomputed hierarchy.

The V-Clip [Mirtich 1998] algorithm tracks the closest pair of features between convex polyhedra, using the approach similar to the closest feature algorithm of [Lin and Canny 1991]. It handles penetrating polyhedra, and can also be used to detect collision between both convex and concave polyhedra. The hierarchy helps to build concave polyhedra from several convex polyhedra. A convex hull of these subsets is constructed and, when a collision is detected; the subsets are tested to find the exact collision.

SWIFT++ shows unified approach to perform queries for checking objects intersections, tolerance verification, exact and approximate minimum distance computation, and (disjoint) contact determination. The methods involve a hierarchical data structure built upon a surface decomposition of the models. The incremental query algorithm takes advantage of coherence between successive frames. A minimum distance computation algorithm based on "Voronoi marching" to test the proximity of a pair of convex polyhedra was also described.

The paper continues with the description of the data structure that codes the BS-tree. We describe the algorithm itself in Section 4. Results are shown in the next section and Section 6 describes some opened questions and the future work.

3 Data Structures and the BS-trees

To generate geometry images we use objects that are modeled by hand in Maya, the Stanford Bunny, and a fractal surface patch. The objects are converted to sets of triangles and then to geometry images. This allows us to create a variety of shapes, both convex and concave. The geometry images can be also sampled with different precision, allowing images in higher resolution with more details and vice versa. Our geometry images are represented by 16-bits per channel and therefore have better precision than the standard 8-bits per channel representation. Example of an object and its corresponding geometry image is given in Figure 3.

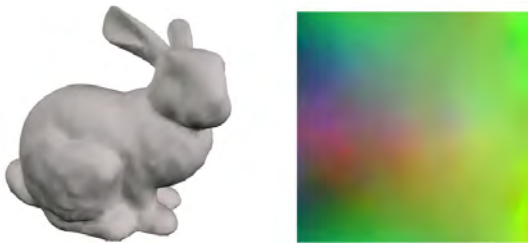

Figure 3: An object and its geometry image

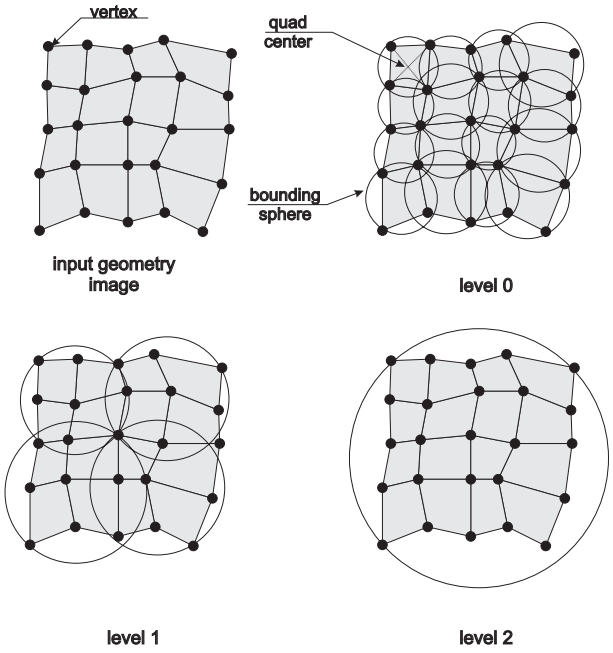

Figure 4: Calculating the hierarchy

From the given geometry image, the additional data structure that codes the bounding spheres hierarchy is directly generated. Geometry images are in our implementation, by now, limited to sizes of $(2^n + 1) \times (2^n + 1)$ vertices. Each quadrilateral is encompassed by a bounding sphere at the deepest level (see in Figure 4). The sphere center is defined as the center of the corresponding min-max AABB. Its radius is the half-length of the largest diagonal. The

geometry image stores in the RGB values the center of the sphere and the alpha stores the sphere radius (see in Figure 6). When the deepest level is stored the upper levels are calculated recursively and stored. Neighboring four spheres from the lower level are encompassed by a bounding sphere. Its center is stored as the RGB and the radius as the alpha value. The topmost level is the bounding sphere of the entire object and corresponds to the root of the tree of BSs.

Mip-mapping cannot be used directly because to store the complete information four values are required. To save the memory, we store the hierarchy in the way that is depicted in the Figure 6. The deepest level is stored at the position $[0,0]$ and the remaining levels are rotated by 90^o and shifted by the size of the base image. The upper image shows the geometry and the lower the alpha channel displayed as the intensity of gray color. It can be seen that the alpha gets greater with the increasing level (displayed as the brighter color in gray scale), i.e., the spheres located higher in the hierarchy are bigger. The BS-tree generated in this way is always perfectly balanced that is one of the biggest advantages of the proposed technique.

Different levels of refinement corresponding to one object are shown in Figure 5. The bounding spheres form an object's shape approximation that improves with each level of the BS-tree .

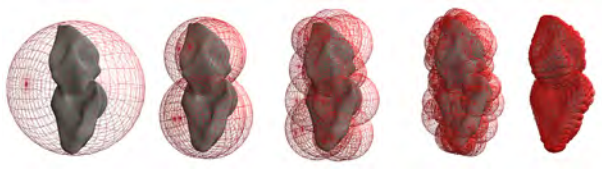

Figure 5: Different levels of bounding spheres tight to the object approximate its shape

The amount of space required to store the hierarchies and the images can be expressed easily. Suppose the geometry image has resolution $(2^n+1)\times(2^n+1)$. Each vertex has three coordinates that are stored as the RGB values. The required size of the geometry will be denoted by $s = 3(2^n+1)\times(2^n+1)$ and s is expressed in integers. The normal vectors need the same space as the vertices. The total size of the geometry image together with the normal vectors will be $2s$.

The lowest level of the hierarchy stores in each bounding sphere four vertices. The lowest level bounding spheres share vertices so each inner vertex belongs to four spheres. The deepest level is made up of $n \times n$ spheres. The upper level is always half of the previous one in the x and in the y direction. So the total size required for the hierarchy is $\sum_{i=0}^{\log_2 n} 2^i$. The geometry image is quadrilateral (see Figure 6) so the total size required is 25% bigger. The size required to store the center of the spheres is therefore $1.5s$. The alpha values are scalars and they are stored in the same way. The size required for the alpha values is $0.5s$ so the total size of the BS-tree hierarchy is $2s$. The entire size of the geometry image, normal vectors, and the BSs is $4s$, where s is the geometry image itself, one more s is required for the normal vectors and $2s$ for the BSs.

4 Collision Detection Algorithm

The collision detection algorithm of two objects is the classical double hierarchy traversation. First, the upmost-level spheres from both

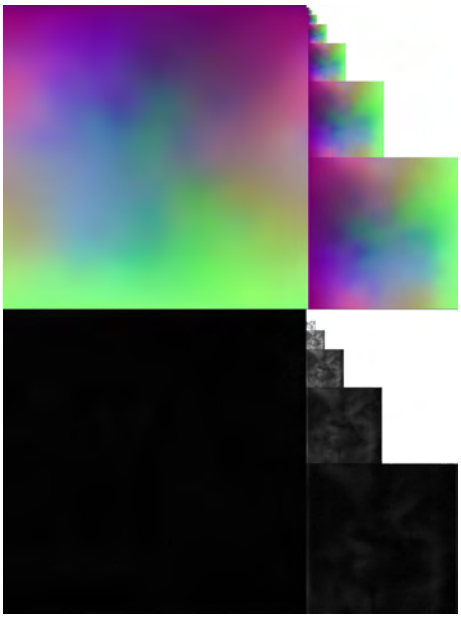

Figure 6: The hierarchy of bounding spheres stored as the geometry images. The upper image shows the average values, lower one shows how the bounding sphere sizes are coded. Since the spheres are small, the gray intensities are enhanced

BS-trees are tested for collision. If there is no collision, the algorithm quits. In the positive case, the algorithm recursively traverses the hierarchies down. It enters the first level of the first object and tests the upper level of the second object for collision with all four spheres. In the case of collision, it goes one level down in the hierarchy of the second object and tests the colliding sphere or spheres against all four spheres of the second object. Both hierarchies are traversed recursively in this zigzag way. At each level, one sphere from the upper level of one object is tested against four spheres of the lower level of the second object and vice versa. At the deepest level the precise triangle-to-triangle test [Moller 1997] is performed.

As mentioned above the collision detection test can finish with the first collision detected or it can report a list of all possible collisions. The first collision detection is fast and the algorithm has logarithmic complexity because of the tree hierarchy. Having two objects with n_1 and n_2 vertices the complexity of this kind of test is $\mathcal{O}(\log(max\{n_1,n_2\})$.

In the case of detection of the all-possible collisions the complexity is $\mathcal{O}(n_1 \times n_2)$. An example of this situation is two planar meshes laying one over another.

The collision detection can also be time critical. In this case it stops the detection when there is no collision or whet the time is up, returning the best possible fit that was found. The non-time-critical test stops testing either when there is no collision or on the deepest level i.e., with the triangle-to-triangle test.

In our implementation, and in all non-time-critical tests shown later in the paper, we have implemented the test that quits when the first collision is detected. It is possible to run the test completely and detect all collisions. We also generate the collision response.

As long as an object moves, the corresponding bounding spheres are translated, and the RGB values of the geometry images must be recalculated. The alpha that codes the distance remains unchanged.

The distance-offset calculation can be applied to all vertices and spheres blindly, but many calculations can be wasted without actually being used. We prefer to perform this step during the hierarchy traversation, just before it is actually needed.

Spheres do not provide very good fit to the objects in general and this is very true especially in the case of quadrilaterals. This can be observed in Figure 7, where the plane is refined by spheres. We have found that this is not an important limitation, because the sphere radius in the BS-tree decreases fast so possible false collision in the first level would be discarded with a high probability in the next level or levels.

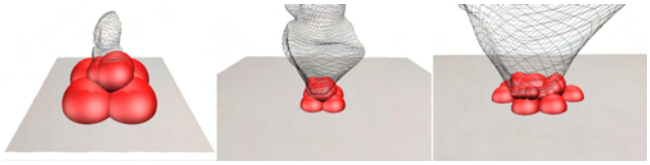

Figure 7: As the object gets closer to the plane the sphere hierarchies are refined (the camera is moved closer to demonstrate the sphere refinement)

The main advantage of our method over the previously published techniques is the way the tree hierarchies are stored. Instead of storing the tree in the dynamically allocated memory, it is implemented as indexing of a two-dimensional array. We simply access the data by dividing/multiplying the actual coordinates by two in each level of hierarchy. The regular sampling, that is the natural property of the geometry images, assures that we always construct a perfectly balanced tree of bounding spheres. Finding coordinates of a sphere within the hierarchy is done fast and is implemented only by the bitwise shifts.

5 Results

GI-COLLIDE was tested against I-COLLIDE [Cohen et al. 1995], the dynamically allocated BS-trees, and the RAPID [Gottschalk et al. 1996] algorithm. We also show the results of a time-critical implementation and a real application. In the comparisons the same conditions were used for all tests. A cube was filled by 2,4,8, ..., 256 randomly moving Stanford Bunnies that were eventually colliding. A collision was detected and the collision response was calculated. To detect the collision the precise triangle-to-triangle test was used. Each test was executed for 3000 frames and the number of collisions and the total time in milliseconds spent on collision detection were reported.

We do not explain the algorithms here and we refer the reader to Section 2.

5.1 I-COLLIDE vs GI-COLLIDE

To compare GI-COLLIDE with the I-COLLIDE we have taken the source code of I-COLLIDE and replaced the default Lin-Caney closest feature test by the GI-COLLIDE algorithm. We tried to make the test as fair as possible. We have changed only the parts of source code where the collision is detected. The space subdivision technique, used by I-COLLIDE to detect active pairs that are later tested for exact collision determination, was left intact.

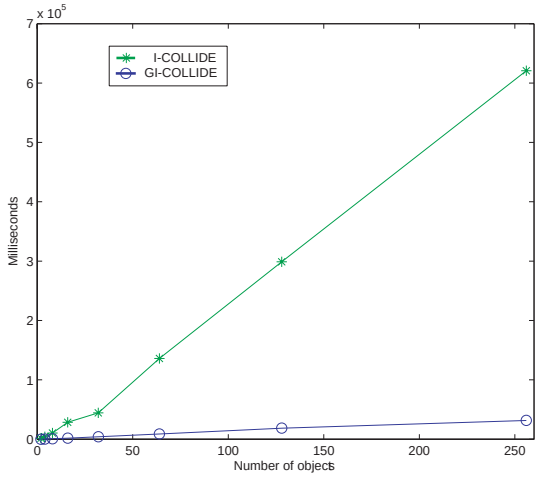

Figure 8: Time spent for collision detection with I-COLLIDE and GI-COLLIDE

To compare both tests we used an object having 8444 triangles for the original I-COLLIDE test and a 65 x 65 pixels geometry image that corresponds to 8450 triangles.

Results of this test are shown in Figure 8. The *x*-axis shows the number of objects colliding against each other and the *y*-axis the time in milliseconds spent on collision detection for 3000 frames. The time spent to detect the collisions grows with the number of objects in both cases. The graph shows that GI-COLLIDE needs significantly less time than the original algorithm used by I-COLLIDE. The difference is more significant especially for large number of objects.

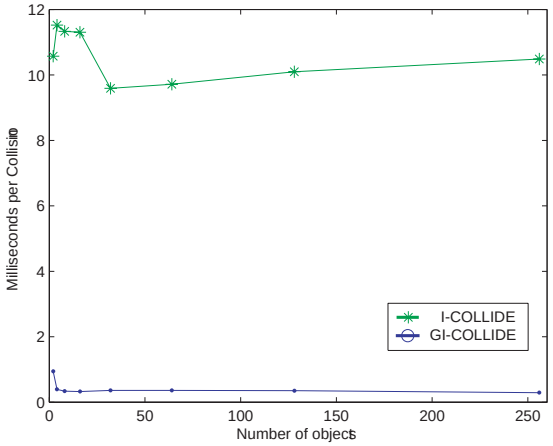

Figure 9: Average time needed to detect a collision for I-COLLIDE and GI-COLLIDE

Figure 9 shows the average time in milliseconds spent per collision detection. I-COLLIDE reported values between 9.5 to 11.5 milliseconds per collision whereas GI-COLLIDE reported times between 0.29 to 0.94 milliseconds.

5.2 BS-trees vs GI-COLLIDE

The second test does not use any space subdivision and the brute force one-to-one test is applied to each pair of objects. We compare the GI-COLLIDE against dynamically allocated (bounding spheres hierarchy) BS-trees. The BS-trees carry exactly the same information and the same hierarchy and structure of bounding spheres as GI-COLLIDE. Both hierarchies are the same, the objects and the tests as well, the dynamically allocated tree is perfectly balanced, and is constructed from the hierarchy stored in the geometry image. Both trees are traversed in the same order and both routines are implemented in the same way. Our intention is to compare the differences in performance between these two techniques. The speed difference comes from the pointer-based scheme used in the dynamically allocated BS-trees and the indexing scheme used by GI-COLLIDE.

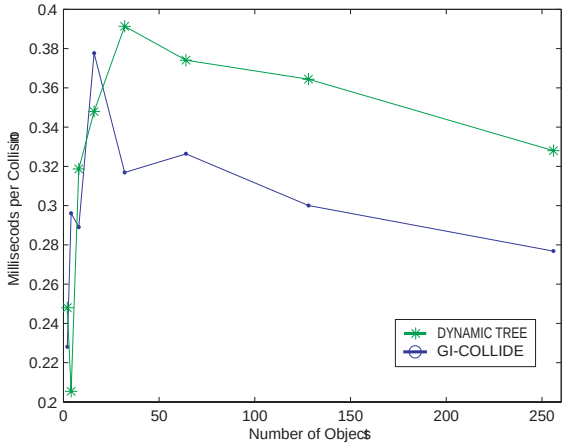

Figure 10: Average time needed for collision detection by BS-tree and GI-COLLIDE

Results of this test are shown in Figure 10. The average number of milliseconds spent on collision detection for different numbers of Stanford Bunnies is displayed. The numbers of milliseconds per collision are almost constant for both cases. On the other hand, for majority of the tests, GI-COLLIDE required less time to detect a collision, providing speedup up to a 19%. It shows that the indexing the geometry image is much more efficient than storing the spheres dynamically allocated in the memory. Apparently, the memory requirements for the geometry images are much smaller, because no pointers are required and the position of each sphere is calculated implicitly.

The dynamically allocated algorithm of the BS-trees based collision detection test was strongly improved. In reality it is really difficult to obtain perfectly balanced trees (something natural to GI-COLLIDE). Some time is also needed to construct and to balance a tree. A geometry image has all these properties inherited from the sampling scheme.

5.3 Rapid vs GI-COLLIDE

Another test we have performed was comparison of the RAPID [Gottschalk et al. 1996] and GI-COLLIDE. The collision detection was again performed with one-to-one comparison, and the time spent on collision detection and numbers of collisions were reported. The RAPID test can be terminated either when the first collision is detected or can be run to detect all collisions. We have used the first contact detection for both algorithms in our test.

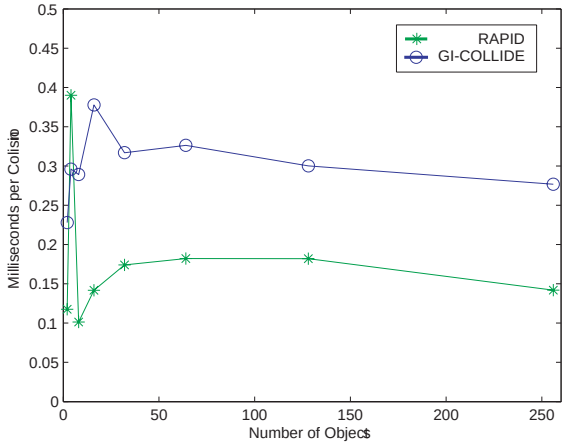

Figure 11: Average time needed for collision detection by RAPID and GI-COLLIDE

The results in Figure 11 show that RAPID provides faster collision detection than GI-COLLIDE. RAPID is a hierarchical representation using OBB-trees that provides better fit and include less empty space than BSs. That is why the OBBs report less number of false detections. We believe this to be the main reason why RAPID detects collisions faster. Comparing the graphics there is not a significant difference in the times reported. In average, RAPID is 0.148 milliseconds faster than GI-COLLIDE for detecting a single collision. To create the OBB-tree a significantly large amount of preprocessing time is required than for GI-COLLIDE.

5.4 A Real Application

To give an example of a real application we have implemented a test of collisions between a model of a moving helicopter and a fractal terrain. The helicopter moves in a realistic way, pitching up and down and rolling left and right, getting close and far from the terrain surface.

Each test was performed on models in different resolutions. We used geometry images in resolution 513×513, 1025×1025, 2049×2049, 4097×4097, and (where possible) 8193×8193 providing objects with $256k$, $1M$, $4M$, $16M$, and $64M$ vertices. All possible resolutions were tested against each resolution that gives maximum of 90.10^9 possible sphere-to-sphere and 68.10^9 polygon-to-polygon tests.

Each test was performed twelve times, both extremities were discarded, and the average values were used.

The geometry images are uncompressed in the main memory. When stored on the disk, they are saved in the TIFF format with 16-bits per channel. The largest geometry image occupies 512MB of the disk space.

Figure 12 shows the result of the collision detection and the results are in tune with the expectations. The number of collision tests corresponds to a logarithmical function of the number of vertices of the object. It is important to notice that two objects in the resolution 4097×4097 pixels occupy more than 368MB of memory (184MB for the geometry and the normal vectors, and 184MB for

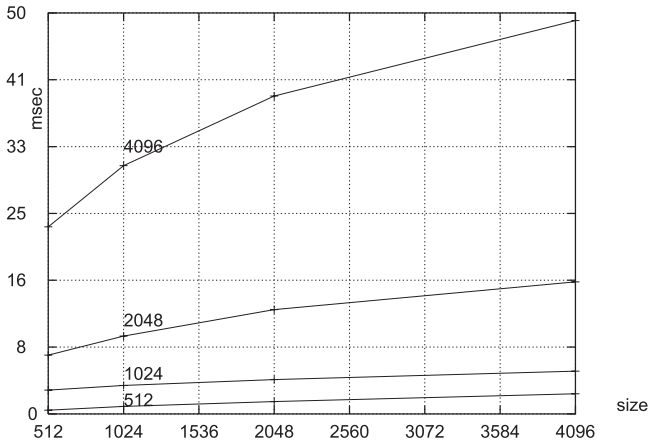

Figure 12: Number of collision tests as the function of object resolution. Two equal objects in varying resolution are colliding and the collision detection times are reported. The lines show the tests performed with geometry images in resolution 513^2, 1025^2, 2049^2, and 4097^2

the bounding spheres). The time for collision detection test was, for the worst case of 68.10^6 polygons, under 60ms. Non-extreme cases, for objects in resolution 1025×1025, were around four milliseconds. The GI-COLLIDE works really efficiently especially for huge amount of data.

5.5 Time Critical Collision Detection

The next example, in Figure 13, shows results of a time-critical test. We have tested an object in resolution 2049^2 against 513^2, 1025^2, ..., 8193^2. The framerate was kept constant on 60fps (leaving 15ms per collision test). The figure shows the actual timing of the performed tests. The number of actual collision follows the logarithmical function but up to the certain level. From this level the amount of tests remains constant.

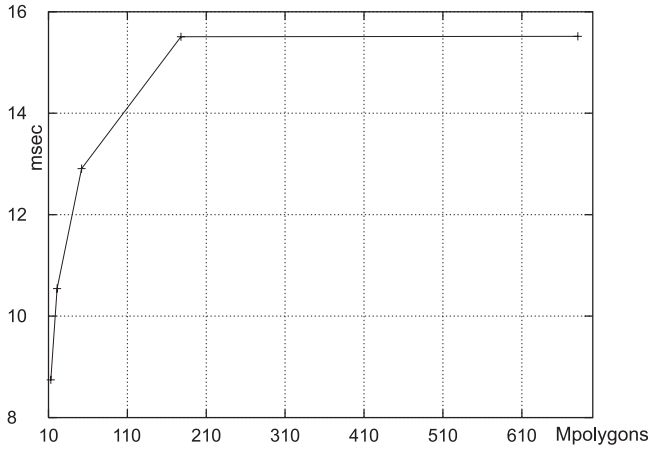

Figure 13: Time required for the collision test as the function of object resolution in the case of time critical collision detection. Object in resolution 2049^2 was tested against objects in resolution 513^2, ..., 8193^2. The framerate was kept on 60 fps. The *x*-axis shows the number of millions of polygons tested, the *y*-axis the time in milliseconds required to find the first collision

6 Conclusions and Future Work

We have presented a new approach for efficient collision detection using geometry images named GI-COLLIDE. The algorithm is based on an efficient coding of the bounding volume hierarchy within the geometry image. From the original geometry image a BS-tree is calculated. The deepest level of the hierarchy stores in its pixels the RGB values of the centers of the bounding spheres of the quadrilaterals. The alpha value stores the sphere radius. This calculation is performed for all possible levels. The highest pixel from the hierarchy corresponds to the bounding sphere of the entire object represented in the geometry image. Traversing both BS-trees performs the collision test and checking the bounding spheres collisions.

Our approach inherits all advantages of the geometry images. The manipulation is fast, we can store extremely precise and huge data representations, etc. The main advantage of GI-COLLIDE is the way the BS-tree is represented in the mip-map-like structure. This structure is always perfectly balanced, can be computed very fast, and is accessed just by indexing the 2D array. Compared to the previously published methods minimum of additional data structures is needed. This allows us to represent objects up to 8193×8193 vertices in our current implementation. The algorithm is fast and one of its main advantages is that it is really easy to implement so it positions itself to computer games, VR applications, and to all time-critical applications where the memory requirements are limiting and the high speed is required. Since we store the geometry images on the disk as TIFF images we can use all the advantages of this representation. We can compress the stored data using LZW algorithm that helps to reduce the space required for the storage.

Our results show that the algorithm is faster or almost of the same speed as I-COLLIDE, RAPID, dynamically allocated BS-trees. It works for real applications even for huge data and can be easily implemented in a time-critical version.

We are limited to genus zero objects in our implementation. More topologically complicated objects require more geometry images [Sander et al. 2003]. Another disadvantage is that geometry images are designed to represent objects, so the test is suited to object-to-object collision detection. A technical disadvantage is that our actual implementation is limited to 2^n+1, $n=1,2,\ldots$ geometry images.

There is some future work left. A possible application is a collision detection together with a level of detail (LOD). When the object is displayed we would select the corresponding LOD and detect the collisions up to the certain level by performing the collision detection on the selected geometry image and not on the original one. This could significantly reduce the computational time namely in complex scenes. An important property of the algorithm is that we do not need necessarily to test a geometry image against another geometry image. It would be interesting to compare a test of a geometry image against k-DOPs [Klosowski et al. 1998], against sphere trees or OBB-trees [Gottschalk et al. 1996], or to apply some image space techniques [Baciu and Wong 2003; Govindaraju et al. 2003]. Another possible future work is an implementation of the collision detection using graphics hardware.

References

BACIU, G., AND WONG, W. 2003. Image-based techniques in a hybrid collision detector. *IEEE Transactions on Visualization and Computer Graphics 9*, 2, 254–271.

BRADSHAW, G., AND O'SULLIVAN, C. 2004. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transaction on Graphics 23*, 1, 1–26.

COHEN, J. D., LIN, M. C., MANOCHA, D., AND PONAMGI, M. 1995. I-collide: an interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM Press, 189–ff.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. *Computer Graphics 29*, Annual Conference Series, 173–182.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. Obbtree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, 171–180.

GOVINDARAJU, N. K., REDON, S., LIN, M. C., AND MANOCHA, D. 2003. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, 25–32.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 355–361.

HE, T. 1999. Fast collision detection using quospo trees. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, 55–62.

HUBBARD, P. M. 1996. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph. 15*, 3, 179–210.

KLOSOWSKI, J. T., HELD, M., MITCHELL, J. S. B., SOWIZRAL, H., AND ZIKAN, K. 1998. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics 4*, 1, 21–36.

LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. 1998. MAPS: Multiresolution adaptive parameterization of surfaces. *Computer Graphics 32*, Annual Conference Series, 95–104.

LIN, M. C., AND CANNY, J. F. 1991. A fast algorithm for incremental distance calculation. In *IEEE International Conference on Robotics and Automation*, 1008–1014.

LOSASSO, F., HOPPE, H., SCHAEFER, S., AND WARREN, J. 2003. Smooth geometry images. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, Eurographics Association, 138–145.

MIRTICH, B. 1998. V-clip: fast and robust polyhedral collision detection. *ACM Transaction on Graphics 17*, 3, 177–208.

MOLLER, T. A., AND HAINES, E. 2002. *Real-Time Rendering*. A K Peters.

MOLLER, T. 1997. A fast triangle-triangle intersection test. *J. Graph. Tools 2*, 2, 25–30.

PONAMGI, M., MANOCHA, D., AND LIN, M. C. 1995. Incremental algorithms for collision detection between solid models. In *Proceedings of the third ACM symposium on Solid modeling and applications*, ACM Press, 293–304.

PRAUN, E., AND HOPPE, H. 2003. Spherical parametrization and remeshing. *ACM Transaction on Graphics 22*, 3, 340–349.

SANDER, P. V., WOOD, Z. J., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, Eurographics Association, 146–155.