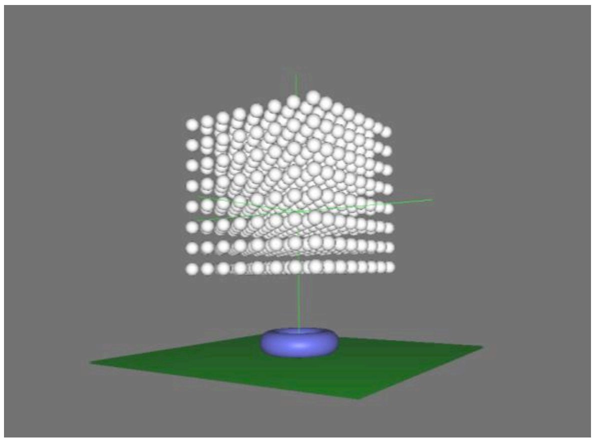
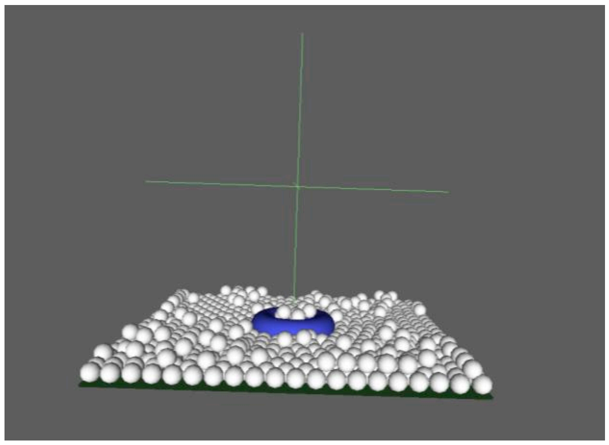
Bounding Sphere Images: A Parametric Bounding Volume Hierarchy for Collision Detection on the GPU.

By Myers Abraham Davis

Physical simulation performed using the Bounding Sphere Image structure introduced in this paper to perform collision detection between particles and a torus.

1.0 Introduction

The simulation of real-world conditions using computers has shown a great deal of success and an even greater potential for development in the past few decades [2,3,4,8,10,15,19]. Physical simulations have been successfully implemented in a wide variety of applications ranging from research by the National Nuclear Security Administration (NNSA) to the production special effects as seen in Hollywood films [5,19]. While computational models for experimentation can be less reliable than traditional methods, the speed and accuracy of simulations performed on computers have improved dramatically over time [4,7,8,13,14,15,16,20,22,25,26]. A large part of this improvement is due to the incredible growth of the computer industry and the development of faster, more sophisticated computer architectures [5,21,24]. The size and complexity of physical simulations may vary greatly depending on their application [2,10,16,19,22]. As such, the hardware necessary to carry out simulations varies by application [4,8,11,19]. While most computers have traditionally run on sequential architectures, supercomputers have been able to reach higher standards of performance through the use of parallel computation [6,12,19,24]. Until recently, commodity hardware has not had any programmable parallel components [5]. The introduction of programmable Graphics Processing Units (GPUs) in recent years has opened up the possibility of small-scale parallel processing on commodity hardware [5,21]. The enormous potential in using parallel computation will likely inspire a demand for parallel algorithms that can benefit from new multiprocessor architectures [5,21]. This paper introduces the concept of a *Bounding Sphere Image* (BSI), a novel data structure that can be used to store each level of a Bounding Volume Hierarchy (BVH) used for collision detection in a format that the GPU is optimized to handle. It is then shown that BSIs may not only be used for collision detection on both the CPU and GPU, but that they may be used to perform adaptive Level Of Detail (LOD) rendering in the same application.

2.0 Review of Literature

Collision detection is a fundamental component of just about any physical simulation involving interacting bodies and has been well studied on CPUs [1,2,4,7,8,13,14,16,23]([13] and [16] provide good overviews of existing research). The general case for 3D collision detection tests for static interference between two arbitrary bodies in a 3D environment. In general, an object's surface is represented with a number of polygons constructed by connecting points sampled along the surface of its actual geometry [5]. As the number of polygons approaches infinite, the polygonal mesh representation approaches the actual geometry. In order to apply physical characteristics to a mesh, there must be some way of determining what conditions are necessary for interaction with that mesh [13,16]. This is the purpose of collision detection. The simplest approach to performing collision detection is to test every triangle in the representation of one object for intersections with every triangle in the representation of another object. However, this approach requires $O(n^2)$ triangle intersection tests to determine collisions for just two objects[1]. Result is extremely inefficient and impractical in most applications. Therefore, various algorithms have been designed to increase the efficiency of the entire process [1,4,7,9,13,14,20,22,23,25,26]. Many approaches use a Bounding Volume Hierarchy (BVH) to avoid testing portions of geometry that are not in close proximity [1,4,20,22,23,26]. Various approaches to implementing BVHs have been taken including oriented and axis-aligned bounding boxes [4,26], bounding spheres [1], and K-Dops [22]. Often these approaches are implemented using octree structures, which are fairly simple to create in a 3D Cartesian coordinate system [13]. This paper describes a novel way to implement a BVH that is specifically tailored to capitalize on optimized characteristics of commodity graphics hardware.

[1] $O(n^2)$ is a notation meaning "on the order of n^2" where n is the number of triangles in a model.

2.1 The GPU

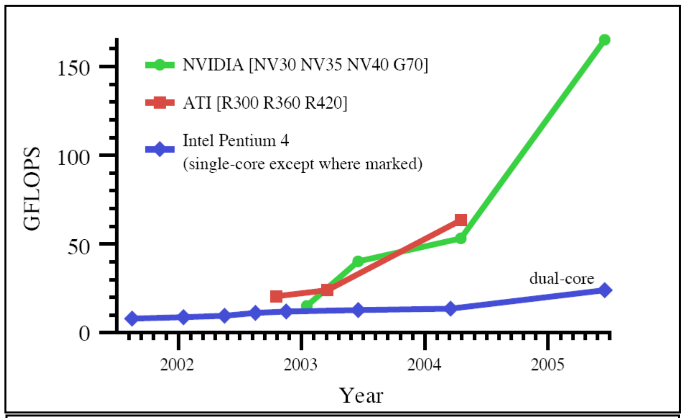


Figure 1: This graph, taken from a survey of general purpose computation on graphics hardware [21], clearly shows how the growth in processing power of GPUs (shown in red and green) has outpaced that of CPUs (shown in blue).

Consumers and scientists have enjoyed an impressive growth in the processing power of computers for decades now (compare the processing power of a "supercomputer" in 1989 [6] to the vastly more powerful personal computers of today [5,17]). However, this trend of growth in performance cannot continue indefinitely on a sequential architecture, particularly as transistors approach the atomic level [24]. Parallel processing has become one of the most promising strategies for maintaining the high standard of growth in computer performance [12,21]. In parallel machines, multiple processors work on large sets of data simultaneously. The GPU, commonly called a graphics card, is specialized commodity hardware that utilizes parallel processing to quickly perform many of the operations commonly involved in the production of 3D graphics [5,17]. GPU manufacturers have only recently developed models with programmable processors [5,17,18], so most research on GPU algorithms is limited to the past four years [5,21]. Current top-of-the-line GPUs have both fragment and vertex processor programmability [5,17,18]. Their parallel structure, coupled with a powerful demand by consumers for faster and more detailed Computer Graphics (CG) has led to a very impressive rate of increase in the processing power of GPUs (see Figure 1) [21]. Especially promising is the ability to increase the performance of GPUs by simply increasing the number of processors they contain, which has led to levels of growth roughly double that of CPUs in recent years [5,21]. Because GPUs can currently process information faster than CPUs, the CPU has become the bottleneck in the graphics pipeline [5,21]. A few efforts have been made to move computation traditionally performed on the CPU to the GPU in order to increase overall efficiency [21]. Much of the research in this area is dedicated to general-purpose algorithms not necessarily having to do with CG [21]. However, the parallel structure of GPUs makes them particularly difficult to program [5].

2.2 Collision Detection on the GPU

A small amount of research has gone into the development of programs that run collision detection, particle systems, or other components of physical simulation on the GPU [7,8,10,11,15,]. There is little material on the subject as it is very current technology [7,8,10,11,15, 21]. Most existing studies on the topic are related to the past three years [5, 21]. Furthermore, traditional algorithms for collision detection on the CPU do not generally translate well to parallel architectures. The difficulty in communicating between processors that are working in parallel makes it impossible to implement most traditional CPU algorithms without major revisions [5, 21].

Most current methods for performing collision detection on the GPU are carried out on the fragment processors and use sampled projections (sampled per fragment) to determine whether objects are intersecting [7,8,11,15]. While this approach has yielded fast results, it is view-dependent and therefore only an approximate method which cannot always guarantee that existing collisions will be detected. In contrast, the structure proposed in this paper is not view dependent and is constructed conservatively so that any existing collision will definitely be detected, making it a more reliable method. The main benefits of BSIs for application on GPUs are their implicit structure and the fact that they store BVH data in a format that GPUs are already optimized to handle – as an *image pyramid*. The motivation for using BSIs is the possible implementation of large-scale collision detection on the GPU in applications where accuracy is important and the program must be able to guarantee that a collision will not be missed. Evidence suggests that an algorithm using BSIs will provide an advantage in speed over current CPU collision detection and an advantage in reliability over existing algorithms for the GPU.

3.0 Methods

The algorithm uses a Bounding Volume Hierarchy (BVH) of bounding spheres to conduct proximity queries. The BVH is constructed from a *geometry image*[2]. Each BSI represents a single level in the hierarchy and is stored in parametric form as a two dimensional array of elements containing four data values per element. A given BSI and can be stored as a RGBA texture image on the GPU. To test the performance of BSIs and predict the feasibility of their implementation in large-scale simulations on the GPU, a number of tests are performed involving single and dual-hierarchy traversals. Finally, the algorithm is adapted to perform limited collision detection on the GPU, confirming the feasibility of BSIs as a way to implement BVH algorithms for collision detection on the GPU.

3.1 Geometry Images

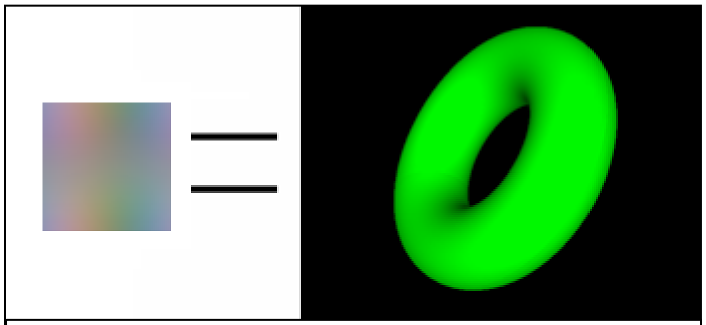


Figure 2: A 64x64 pixel geometry image (left) stores all of the geometry in a polygonal mesh (right) in a two dimensional array.

A geometry image is a parametric representation of a 3D mesh that can be stored as an image where every pixel represents a single vertex. This two-dimensional representation is possible because the connectivity of vertices is made implicit with respect to their adjacency. BSI construction is designed to work on any rectangular geometry image. A single hierarchy is created for each patch in the geometry image automatically. To maximize efficiency, multiple-patch hierarchies can be expanded manually to include bounding volumes around groups of patches. The creation of geometry images is a complex topic and the subject of cutting-edge research. While one other

[2] Geometry images are parametric representations of meshes that can be stored as image files. The connectivity of vertices in a geometry image is implicit based on a convention established through regular sampling performed in the construction of the image. This makes it possible to store the object without a triangle list or index array [9].

geometry image was used in testing, the only geometry image created explicitly for this project was of a simple torus (see Figure 2). This model was created using a common formulaic parameterization based on two angles. Readers are directed to [9] for more general approaches to creating geometry images from arbitrary polygonal meshes. The algorithms in this project are designed to compliment advances in research involving the creation and manipulation of geometry images.

3.2 Constructing the Hierarchy

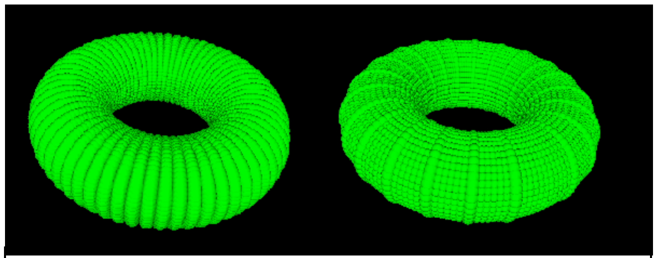


Figure 3: A torus (left) based on a 32x32 geometry image shows regularity in the size and placement of bounding spheres. The algorithm works with geometry images of other dimensions (right, a 150x40 model), with visible compensation.

Each BSI is constructed using the original geometry image data, meaning that any BSI in the hierarchy can be reconstructed independently. BSIs are best suited for images with $2^k \times 2^k$ dimensions. For images with non-power-of-two dimensions, bounding spheres are assigned extra vertices at regular intervals to compensate (see Figure 3). The hierarchy is created by first loading the geometry image into RAM and calculating the number of levels between the top level bounding sphere and the leaf nodes. This calculation may depend on how small the leaf nodes need to be in a given application. The default setting finds the number of levels necessary for the leaf nodes to bound a single triangle each. After establishing the number of levels in the hierarchy, the geometry image is partitioned into rectangular "blocks" of vertices specified by a set of intersecting rows and columns. The number of vertices in a given block is reduced by a factor of four in each subsequent BSI (see figure 4). Each block is passed to a function that creates one bounding sphere per block. Each BSI is arranged so that adjacent border vertices in any two given blocks within the BSI will belong to adjacent bounding spheres, maintaining the relative locations of bounding spheres with respect to the location of their vertices. This arrangement makes the relationip between parent spheres and child spheres implicit.

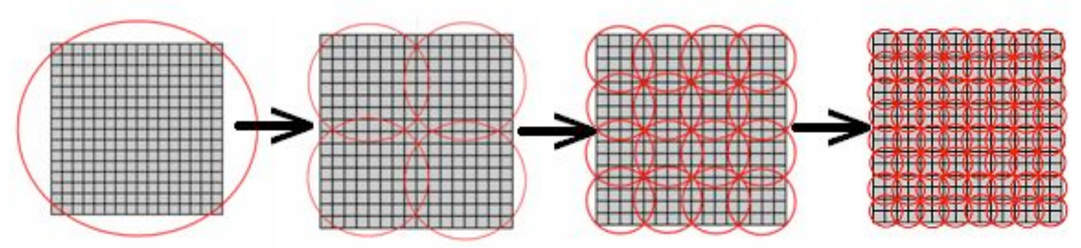


Figure 4: A conceptual representation of bounding volume subdivision in parametric space. Subsequent levels of the BVH divide the vertices of any given bounding sphere into four groups and creating new bounding spheres for each group.

To ensure that all geometry in a model is contained in each BSI and not just the vertices, each bounding sphere contains the vertices that border its assigned block.

3.3 Bounding Spheres

Bounding spheres offer a number of advantages to the algorithm. The primary reason for using bounding spheres is that they require a minimal amount of data to store. While a bounding box requires (at least) the *x*, *y*, and *z*-coordinates of two opposite corners in a given box, bounding spheres only require these coordinates for a center point in addition to a single value for the sphere's radius. The ability to represent a single bounding volume with only four numerical values allows for natural mapping of the BSIs in a BVH to image pyramids that can be stored on the GPU. Spheres also offer a simple way to test for intersection that should translate well to parallel architecture. This test only requires finding the distance between the transformed center points of two bounding spheres and comparing it to the sum of their radii. To speed up this process the comparison is done between the squared value of the distance and the square of the sum of the radii (multiplication is less computationally expensive than taking a square root).

3.4 Constructing a Bounding Sphere

Each bounding sphere is constructed using a list of contained vertices as an input. The center and radius of the sphere are then calculated under the constraint that all of the assigned geometry must be contained inside the sphere. This organization is somewhat general so as to give the algorithm enough flexibility to adapt to different applications. In applications where hierarchies are calculated as part of pre-processing, a relatively complex formula may be used to find the tightest bounding sphere; accordingly, in real-time applications where the hierarchies are formed at each time step (i.e. deformable body simulation), a faster formula may be used. Developers are given the freedom to decide whether the efficiency of tighter bounding volumes will outweigh the cost of a more sophisticated algorithm for calculating bounding spheres based on the specifics of a given application. The formula used as a default averages the minimum and maximum x, y, and z-coordinates to find the center point of the sphere then finds the greatest distance between that center and any of the contained vertices to calculate the radius.

3.5 Storing the Hierarchy

As noted earlier, one of the advantages to using bounding spheres is that they require minimal data to store. This low memory cost makes it possible to organize BVHs in a format that matches the texture memory on GPUs. This is especially advantageous because GPUs are already fairly well optimized in handling texture images. By storing the hierarchy data for a single object as an image pyramid made up of BSIs, it is possible for vertex or fragment processors to perform proximity queries by accessing the BVH through texture lookups. In a BSI, each bounding sphere is stored as a single texel of a RGBA image. The bounding spheres have been stored as the location of their center points (three floating point numbers each) and the lengths of their radii (one floating point number each), which adds up to precisely the number of channels in a RGBA texture image.

3.6 Quad-Tree Traversal on the CPU

The BVHs in this algorithm form a quad-tree structure. Tree traversal is addressed differently depending on whether the algorithm is running on the CPU or GPU. The quad-tree structure used is designed so that the relation between parent nodes and child nodes is implicit based on their respective positions in BSI's. For the CPU implementation, an index of every bounding volume is taken so that the relation between parent nodes and child nodes is implicit based on a numerical convention. The index follows conceptually from a graph of a basic quad-tree structure (see figure 5). Each level of the quad-tree is numbered from left to right with respect to the quad-tree graph. The corresponding index of texels in each BSI (figure 6) maintains the same numerical relation

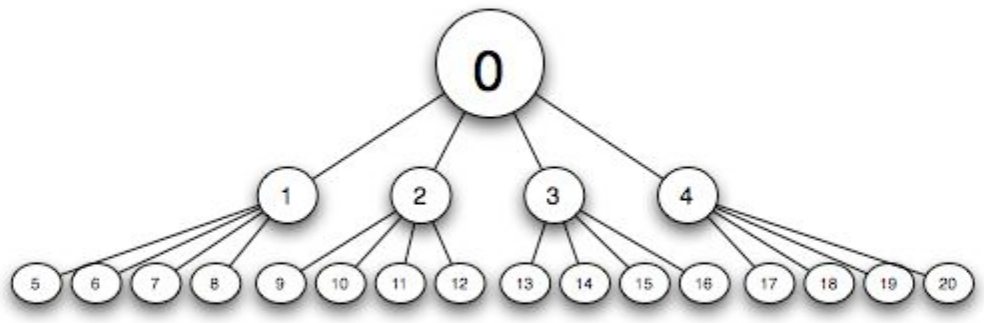


Figure 5: A graph of a basic quad-tree structure. In this case each node represents a different bounding sphere and each level represents a BSI.

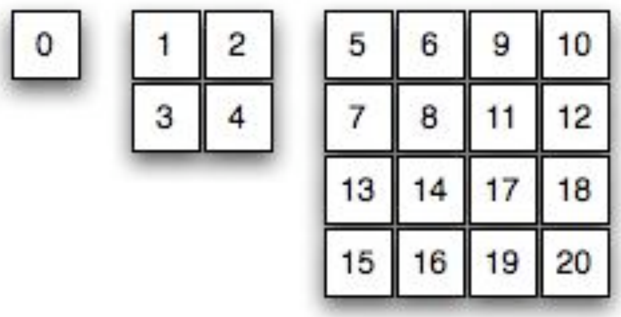


Figure 6: The indexing method for each bounding volume hierarchy follows from a graph of a basic quad-tree structure. Each parent node is then related numerically to each of its child nodes.

between parent spheres and child spheres. Given this arrangement, for any bounding volume B_n that is not a leaf node, its child nodes are defined as $Child_i(B_n) = B_{4n+i}$ where $1 \leq i \leq 4$.

The index is applied to each BSI using a recursive function that follows the rectangular arrangement of subsequent groups of bounding spheres connected to the same line of parent nodes. When an intersection test with a given node in the quad-tree returns positive, intersection tests are then performed on each of that node's children. If an intersection test with a leaf node returns positive then the program has confirmed that there is a possible collision. If the leaf node is small enough, then the point of intersection with the leaf node can be used as an approximation of an actual collision. For applications requiring more accuracy, triangle intersection tests can be performed between the triangles inside intersecting leaf node spheres and an exact point, edge, or surface of collision can be found.

4.0 Experimental Setups, Results, and Discussion

Several experimental setups were used to test the performance of BSIs in collision detection. Each setup tested for something different. The main questions addressed were:

- Does collision detection work? If so how efficient is it?
- Can it be applied to real-time physical simulations?
- Can the same hierarchy used for collision detection be used for adaptive LOD rendering?

4.1 Test 1: Dual-Hierarchy Traversal And Collision Query Times

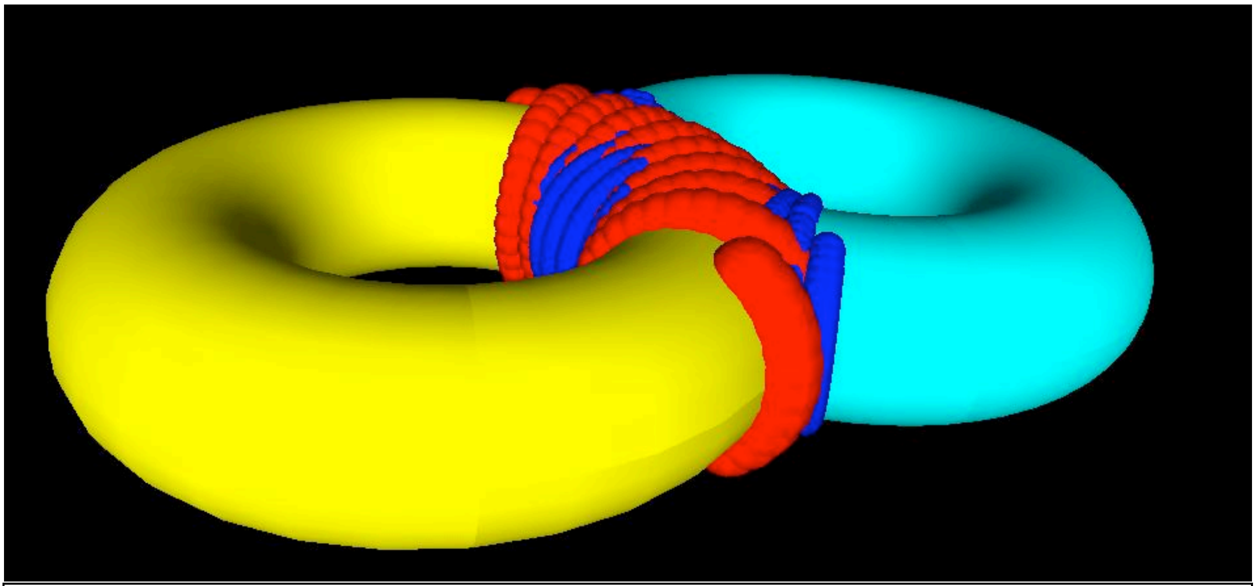


Figure 7: Collision detection between two models of tori at different resolutions. The intersecting leaf nodes of each tori are rendered. The red bounding spheres represent the yellow torus's leaf nodes and the darker blue bounding spheres represent the blue torus's leaf nodes.

The setup used to test the accuracy and efficiency of collision detection involved testing for collisions between two tori. The tori were first tested approaching one another from different angles to ensure that collision detection worked properly. This test confirmed that collision detection was successful and accurate from every angle.

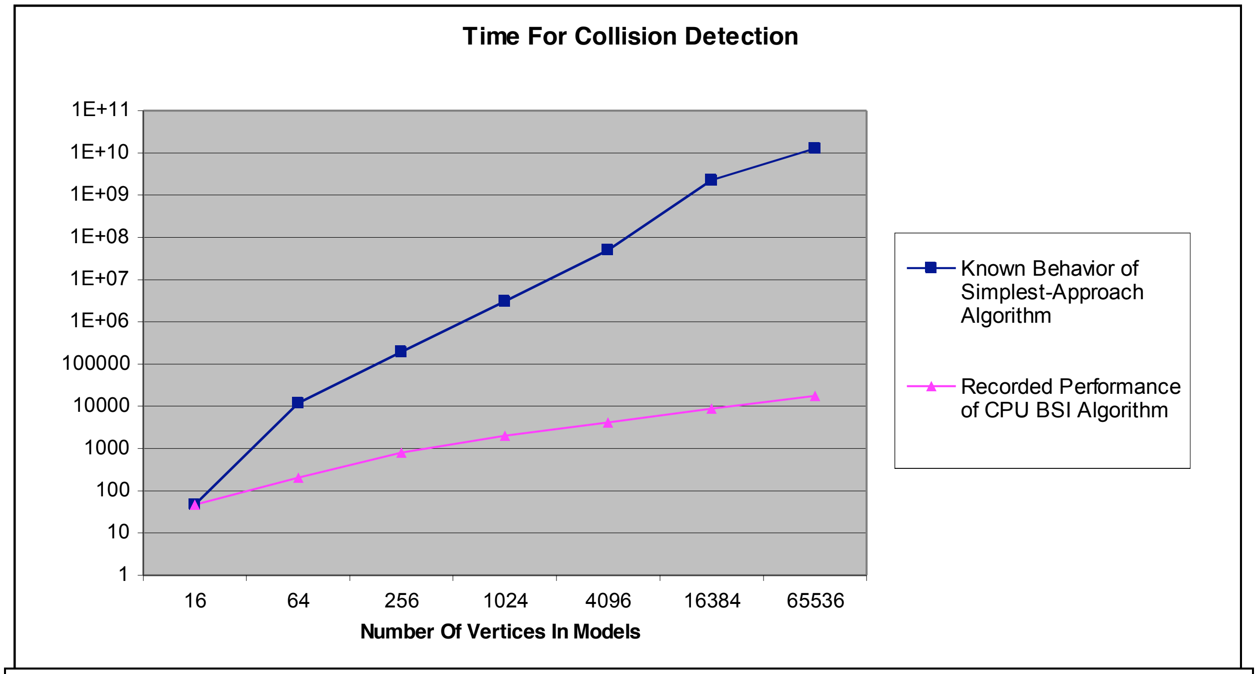


Figure 8: This graph of the time spent on collision detection for models of increasing complexity clearly shows that the BSI algorithm (shown in pink) offers a dramatic increase in the efficiency of collision detection when compared to the behavior of a simplest-approach algorithm (shown in blue).

The setup was then modified so that two identical tori would remain in a consistent location with respect to one another while the number of vertices in each model was increased. The time spent on collision detection was then taken for over one thousand passes per pair of tori. To evaluate the performance of the algorithm, the average times for collision detection on each pair of models is graphed and the behavior of this graph is compared to the known behavior of simplest-approach collision detection in which every possible triangle-pair would be tested (figure 8). The starting value of the simplest-approach graph had been raised to ensure that the resulting graph is a conservative estimate. The difference between the two models is a close approximation of the actual increase in efficiency obtained through the use of the BVH. As the graph clearly shows, the BSI algorithm provided a dramatic increase in performance. When compared to existing CPU algorithms for accelerating collision detection, this result can be considered average [13, 16]. The main benefit offered by the use of BSIs, however, is the potential for use with graphics hardware. Therefore, average performance on the CPU can be considered a significant success. If the same standard of efficiency can be maintained on GPU implementations then this algorithm could likely perform several times faster than CPU algorithms in short time. Furthermore, the algorithm would presumably benefit from the stunning rate of growth in processing power observed by GPUs. According to some estimates, this would mean doubling its advantage over its CPU counterparts roughly every year [21].

4.2 Test 2: Real-Time Physical Simulation Between Particles and Torus

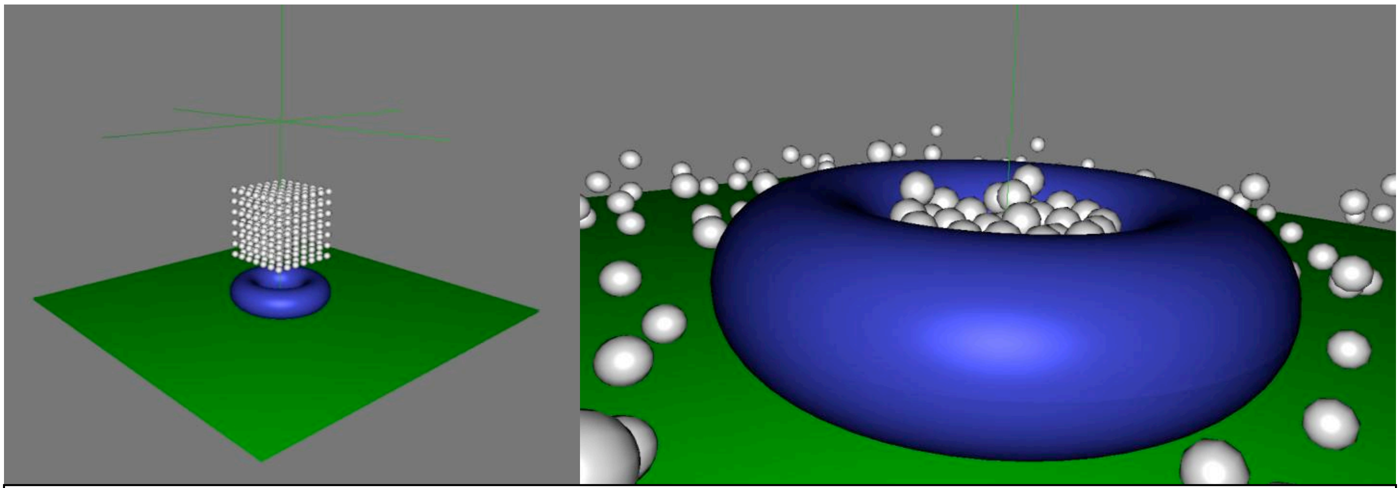


Figure 9: BSI collision detection in a physical simulation on the CPU. Collisions between particles and the torus are detected using a BSI hierarchy. Given the initial rest conditions shown on the left, after some time in the simulation a number of particles gather in the hole of the torus as shown on the right.

The second test was to see if the BSI hierarchy could be applied in an actual real-time physical simulation. For this test, a particle system was developed which runs on fairly straightforward 3D vector mechanics. The results of this test are more subjective than for the other setups, as its purpose was primarily to give an example of application. By studying the behavior of the particles in the simulation as they interact with the torus and with one another, the accuracy of the physical model can be assessed. The simulation also serves as an example of what benefits computer simulations offer over their real-world counterparts in general. The user in this program has control over many factors that are difficult to control in the real world such as gravity, air resistance, friction, the speed at which time passes, and the elasticity of collisions.

4.3 Adaptive LOD Rendering using BSIs

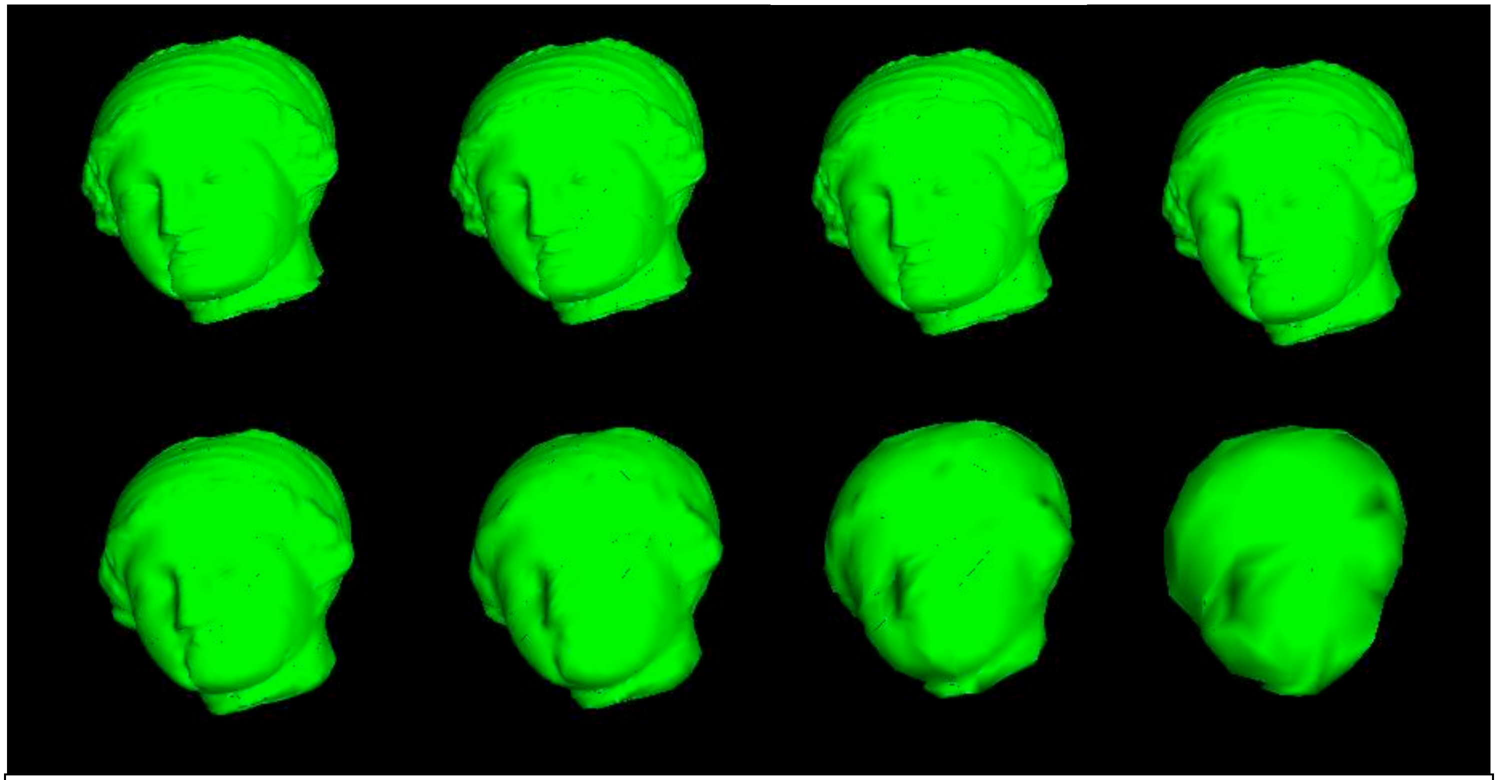


Figure 10: Adaptive LOD rendering of a 48-patch geometry image using BSIs. The original model is courtesy of Budirijanto Purnomo and the Johns Hopkins Computer Graphics lab. Each LOD representation is rendered using a different BSI in the hierarchy. The ability to use BSIs for both collision detection and adaptive LOD rendering is one of the advantages that this algorithm has over previous algorithms for collision detection using the GPU.

Because of the way that bounding spheres are constructed in this algorithm, the set of points that make up the centers of all bounding spheres in a particular BSI are very close to a sampling of the original geometry. This is because the function used to determine the center point of a bounding sphere is similar to some algorithms used for averaging the position of several vertices into one vertex for mesh simplification. Also, because both the original geometry image and the BSIs share the same rectangular shape and implicit spatial relationship between adjacent members of their respective arrays, it is possible to use a given BSI to create a simplified representation of the original geometry image. In models that consist of multiple patches (like the Igea model above), the BSI approximations for the original geometry leave large cracks in the surface model. To solve this problem, the BSI being used for rendering is “sewn” to the original geometry image at sampled points along the borders of the original patch which contain replicated vertices. The result is surprisingly effective set of multi-resolution renderings for the model (see figure 10). The effectiveness of BSIs for use in LOD rendering is assessed both visually in figure 10 and in terms of

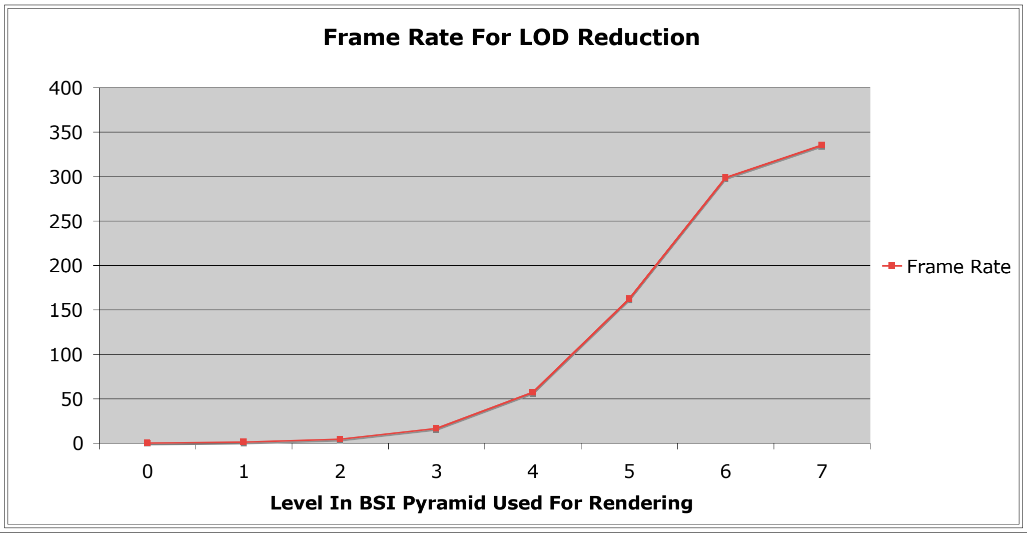


Figure 11: This graph shows how reducing the LOD for a model using the BSI hierarchy affects frame rate. Being able to trade off between increased rendering speed and more detailed representations of a model allows a program to adapt to a wider variety of applications.

how it affects the application’s frame rate in figure 11. While this method of performing LOD reduction is by no means as effective as the some of the cutting edge standards, it is surprisingly effective considering that BSIs were not originally intended for this purpose. The ability to perform collision detection and adaptive LOD rendering using the same structure could save on the amount of information that needs to be passed to the GPU at execution. If the LOD algorithm could somehow be made more efficient and if the few small holes that appear between patches in reduced meshes could be filled then BSI’s could prove to be very useful in performing LOD on top of their intended purpose.

5.0 GPU Implementation[3]

The primary goal of the GPU implementation of this algorithm was to prove that collision detection could be performed with BSIs on the existing hardware. To monitor exactly how the collision detection played out, a simplest case scenario was set up to test only a single particle against the torus. The bulk of the computation is performed on the fragment processors because they are more numerous than the vertex processors and capable of processing information faster. Due to the lack of any existing and commonly available debugger for GPU programs, the result of each pass is printed out to the frame buffer and read back by the CPU so that it can be checked to ensure that the program is functioning properly. Future implementations will try to minimize the amount of information that needs to be read back by the CPU before rendering each frame. Traversal is structured differently than on the CPU and relies solely on the implicit structure of the BSIs rather than an index of bounding volumes. This unique aspect of BSIs is what made it possible to perform collision detection on the GPU. The relationship between parent spheres and child spheres is based on location in the BSIs. For a given bounding sphere $\boldsymbol{B}_{(u,v)}$ with texture coordinates at texels $\boldsymbol{u}$ and $\boldsymbol{v}$ that is not a leaf node, four children spheres $\boldsymbol{B}_{(2u,2v)}$, $\boldsymbol{B}_{(2u+1,2v)}$,

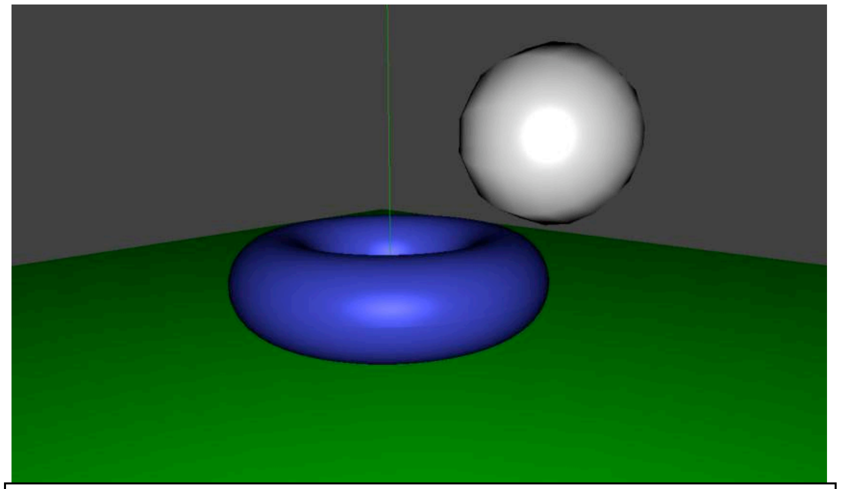


Figure 12: Collision detection between a single particle and the torus performed on the GPU was successful, proving that collision detection can be performed on the GPU using BSIs.

[3] This section may require the reader to have a more in-depth background on the components of a GPU and their functions to be fully understood. As any effective introduction to GPUs would require considerable length, in the interest of space I will instead direct the reader to [5] for a lengthy, more complete explanation. The main things that the reader should know are that the programmable components of a GPU are the vertex processors and the fragment processors. For those who are unfamiliar, a fragment can be thought of as a potential pixel.

$\boldsymbol{B}_{(2u,2v+1)}$, and $\boldsymbol{B}_{(2u+1,2v+1)}$ are defined in the next level BSI. Each pass tests for all possible bounding volume intersections on a given level of the hierarchy, meaning that all of the tests against a given BSI are performed in parallel during one pass. This batching of intersection tests is a departure from the sequential computation of traditional methods for collision detection that takes advantage of the parallel structure of GPUs. The result of a given pass is written to the back buffer and read back by the CPU. The process of reading back this information is by far the current bottleneck in the application. However, as the read back from the GPU to the CPU becomes faster on new hardware it can be assumed that this problem will be significantly reduced. Testing was successful in simulating the collisions between a single particle and the torus on the GPU (see figure 12). While the algorithm is not yet optimized, it is believed to be the first algorithm for collision detection using a BVH on the GPU.

6.0 Conclusions and Future Work

I have introduced a novel structure for collision detection using a BVH on the GPU. I have named this structure a *Bounding Sphere Image*. Through a series of experiments I have proven the efficiency of BSIs in a number of settings. I collected data to support both the reliability and efficiency of the algorithm and proved that implementation on the GPU is possible. The application of this research could potentially mean a dramatic increase in the performance of physical simulations on commodity hardware. The algorithm could not only benefit from the already impressive speed of current GPUs, but if current trends in hardware development continue then it could benefit from the rapid pace of development in GPUs making its advantage over CPU counterparts double every few years [21]. Before the algorithm will be ready for commercial use it will need to be optimized and expanded to perform both dual-hierarchy traversals and single hierarchy traversals on the GPU. Commercial application will also become more feasible if it can be demonstrated that the algorithm will work on simulations dealing with many objects in a scene at once. Goals for the near future include adapting the GPU implementation for dual hierarchy traversal and minimizing the information that needs to be read back by the CPU in each pass.

Acknowledgements:
Jonathan Cohen, Charlotte Saylor, The Johns Hopkins University Computer Graphics Lab for advice and support, and to Budirijanto Purnomo for providing the Igea atlas model.

7.0 Citations

[1] BRADSHAW, G., O'SULLIVAN, C.: Sphere-tree construction using dynamic medial axis approximation. In *Proceedings of the 2002 ACM Siggraph/Eurographics Symposium on Computer Animation* (2002). ACM Press, pp. 33-40.

[2] CARLSON, M., MUCHA, P. J., TURK, G.: Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Transactions on Graphics* (2004). ACM Press, pp. 377-384.

[3] CHEN, Y., COHEN, J., KUMAR, S.: On the Visualization of Time-Varying Structured Grids Using a 3D Warp Texture. In *Proceedings of the Conference on Visualization '04* (2004). IEEE Visualization. IEEE Computer Society.

[4] COHEN, J. D., LIN, M. C., MANOCHA, D., PONAMGI, M.: I-COLLIDE: an interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 Symposium on interactive 3D Graphics* (1995), ACM Press, pp. 189-ff.

[5] FERNANDO, R., KILGARD, M. J.: *The Cg Tutorial: The Difinitive Guide to Programming Real-Time Graphics.* NVIDIA Corporation (2003), Addison-Wesley.

[6] FUCHS, H., POULTON, J., EYLES, J., GREER, T., GOLDFEATHER, J., ELLSWORTH, D., MOLNAR, S., TURK, G., TEBBS, B., ISRAEL, L.: Pixel-planes 5: a heterogeneous multiprocessor graphics system using processor-enhanced memories. In *Proceedings of the 16th Annual Conference on Computer Graphics and interactive Techniques* (1989). ACM Press, pp. 79-88.

[7] GOVINDARAJU, N. K., LIN, M. C., MANOCHA, D.: Fast and reliable collision culling using graphics hardware. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (2004), ACM Press, pp. 2-9.

[8] GOVINDARAJU N., REDON S., LIN M., MANOCHA, D.: CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings of the ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2003), pp. 25-32.

[9] GU, X., GORTLER, S. J., HOPPE, H.: Geometry images. In *Proceedings of the 29th Annual Conference on Computer Graphics and interactive Techniques* (2002). ACM Press, pp. 355-361.

[10] HARRIS, M. J., COOMBE, G., SCHEUERMANN, T., LASTRA, A.: Physically-based visual simulation on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2002). Eurographics Association, pp. 109-118.

[11] HOFF, K. E., ZAFERAKIS, A., LIN, M., MANOCHA, D.: Fast and simple 2D geometric proximity queries using graphics hardware. In *Proceedings of the 2001 Symposium on interactive 3D Graphics* (2001), ACM Press, pp. 145-148.

[12] IGEHEY, H., STOLL, G., HANRAHAN, P.: The design of a parallel graphics interface. In *Proceedings of the 25th Annual Conference on Computer Graphics and interactive Techniques* SIGGRAPH '98. ACM Press, pp. 141-150.

[13] JIMENEZ, P., THOMAS, F., TORRAS, C.: 3d Collision Detection: A Survey. *Computers and Graphics*, 25(2), pp. 269-285, 2001.

[14] KIM, D. J., GUIBAS, L. J., SHIN, S. Y.: Fast collision detection among multiple moving spheres. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*. ACM Press, pp. 373-375.

[15] KOLB, A., LATTA, L., REZK-SALAMA, C.: Hardware-based simulation and collision detection for large particle systems. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2004), Eurographics Association, pp. 123-131.

[16] LIN, M., GOTTSCHALK, S.: Collision detection between geometric models: A survey. In *Proceedings of the IMA Conference on Mathematics of Surfaces*, 1998.

[17] LINDHOLM, E., KLIGARD, M. J., MORETON, H.: A user-programmable vertex engine. In *Proceedings of the 28th Annual Conference on Computer Graphics and interactive Techniques* SIGGRAPH '01 (2001). ACM Press, pp. 149-158.

[18] MARK W. R., GLANVILLE R. S., AKELEY K., KILGARD M. J.: Cg: A system for programming graphics hardware in a c-like language. *ACM Transactions on Graphics* 22, 3 (2003), ACM Press, pp. 896-907.

[19] “NNSA Computers Lead Global List.” National Nuclear Security Administration Press Release, July 10, 2001.

[20] OTADUY, M. A., LIN, M. C.: CLODs: dual hierarchies for multiresolution collision detection. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2003), Eurographics Association, pp. 94-101.

[21] OWENS, J., LUEBKE, D., GOVINDARAJU, N., HARRIS, M., KRUGER, J., LEFOHN, A., PURCELL, T.: A Survey of General-Purpose Computation on Graphics Hardware. In *Proceedings of the EUROGRAPHICS State of the Art Reports* (2005), Eurographics Association, pp. 21-51.

[22] RAABE, A., BARTYZEL, B., ANLAUF, J. K., ZACHMANN, G.: Hardware Accelerated Collision Detection - An Architecture and Simulation Results. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 3* (2005). IEEE Computer Society, pp. 130-135.

[23] TAN, T., CHONG, K., LOW, K.: Computing bounding volume hierarchies using model simplification. In *Proceedings of the 1999 Symposium on interactive 3D Graphics* (1999). ACM Press, pp. 63-69.

[24] TOSIC, P. T.: A perspective on the future of massively parallel computing: fine-grain vs. coarse-grain parallel models comparison & contrast. In *Proceedings of the 1st Conference on Computing Frontiers* (2004), ACM Press, pp. 488-502.

[25] YOON, S., SALOMON, B., LIN, M., MANOCHA, D.: Fast collision detection between massive models using dynamic simplification. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2004).vol. 71. ACM Press, pp. 136-146.

[26] ZACHMANN, G.: Minimal hierarchical collision detection. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (2002). VRST '02. ACM Press, pp. 121-128.

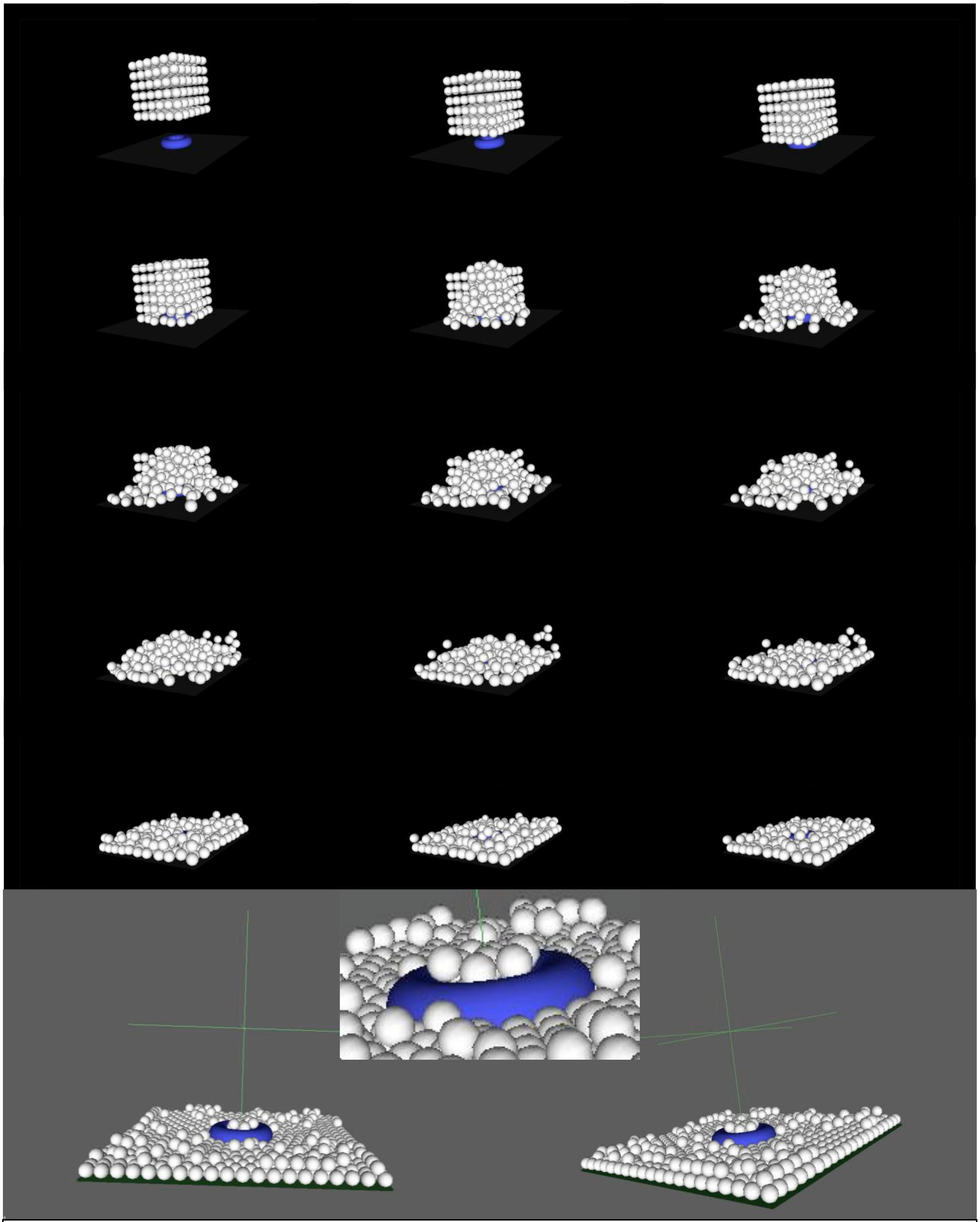


Figure 13: Real time simulations of a particle system interacting with a donut (torus) using a BVH of BSIs. Particles stack around the torus and inside of its hole as they would in real life.