Contact Detection Algorithms

S. Kockara[1], T. Halic[1], C. Bayrak[1], K. Iqbal[1], and R. A. Rowe[2]
University of Arkansas at Little Rock[1], University of Arkansas for Medical Sciences[2]

Abstract— **A process that determines whether two or more bodies make contact at one or more points is called contact detection or collision detection. Contact detection is inseparable part of the computer graphics, surgical simulations, and robotics etc. There are large of methods that are used for collision detection. We will review a few of the most common ones. Algorithms for contact determination can be grouped into two general categories - broad-phase and narrow-phase-. This paper provides a comprehensive classification of a collision detection literature for the two phases. Moreover, we have attempted to explain some of the existing algorithms which are not easy to interpret. In the process, we have tried to keep sections self-explanatory without sacrificing depth of the coverage.**

Index Terms—**contact detection, collision detection, deformation, bounding volumes, spanner**

I. BROAD-PHASE COLLISION DETECTION

When given the two models and their placements in the world space, the simplest brute force approach to perform a collision query is to test each of the primitive segments in object A against each of the primitive segments of object B; that process requires number of A's primitive segments times number of B's primitive segments overlap tests. This approach is reasonable for relatively small models. However, we cannot perform exhaustive pair-wise testing on models which have thousands of primitives since a collision query needs to be performed in every simulation step in order to detect colliding objects. Animations can have many objects, all of which may have a complex geometry such as polygonal soups of several thousands facets. Therefore, performing collision detection is computationally extensive task. Yet, it can be difficult to obtain real time interaction. Thus, to eliminate these computationally costly pair-wise tests different algorithms are proposed in the literature.

Hubbard [1] was the first to classify the collision detection into two parts e.g. broad-phase and narrow-phase. Those concepts of broad-phase and narrow phase collision detection reduce the computational load by performing a coarse test in order to prune an unnecessary pair test. Broad-phase collision detection identifies disjoint groups of possibly intersecting objects. On the contrary, pruning unnecessary primitive-pair test is narrow-phase collision detection which identifies disjoint groups of possibly intersecting objects' primitives e.g. polygons. Most of the literature uses Hubbard's broad and narrow phase collision detection scheme to classify collision detection algorithms [2]-[3]. The same classification technique will also be used throughout this survey along with introduction of new algorithms. Some of the methods such as bounding volumes or boxes are included in both broad and narrow phase collision detection.

II. BROAD-PHASE COLLISION DETECTION

From computational geometry [4], we know that broad phase collision detection can be achieved by answering these questions: which box contains a given point and/or which boxes overlap a given box. Therefore, for broad-phase collision detection, approximating objects with boxes make broad-phase collision detection easier. However, we also need to know which boxes will be overlapped in the next iteration because in physics-based animations objects can move and deform in time. To perform this, broad-phase collision detection can be done with three different kinds of algorithms: All-pair test (Exhaustive Search), sweep and prune (Coordinate Sorting), and hierarchical hash tables (multi level grids).

An exhaustive search is the brute-force approach which compares each object's bounding volume with others' bounding volumes. If algorithm finds colliding bounding volumes then it starts further investigation with narrow phase collision detection algorithms. Sweep and prune algorithm [5][6] projects every object's bounding volume's starting and ending points onto the coordinate axes. If there is intersection among the entire principal coordinate axes then there is a collision between those objects. Another approach for broad-phase collision detection is hierarchical hash tables, such as grid [7]. This algorithm divides the entire scene into the equal size grids along all the principal axes. All points overlapping with the given grid cell is identified by the algorithm. If there is more than one object sharing the same cell, those objects are possibly colliding objects.

III. NARROW-PHASE COLLISION DETECTION

Broad phase lists pairs of possible colliding objects and narrow phase inspects further each of these pairs and finally contact determination algorithms verify the exact collisions. Narrow phase algorithms usually return more detailed information such as separation distance, penetration depth, closest points etc. This information can be later used for the computation of time of impact, collision response and forces, and contact determination.

Narrow phase collision detection algorithms can be categorized into four groups [8]: feature-based, simplex-

based, volume-based, and spatial data structures (bounding volumes and spatial subdivisions).

Feature-based Algorithms

These kinds of algorithms directly work on the geometric primitives of the objects. Some of the well known feature-based algorithms are polygonal intersection [8], Lin-Canny [9], V-Clip[10], and SWIFT[11][12]. Another feature-based algorithm is image space based technique. Because image space based algorithms are suitable to implement on the graphical processing unit (GPU), they are currently more preferred techniques.

Lin-Canny algorithm is the first feature-based algorithm in literature. There are other feature tracking algorithms proposed based on Lin-Canny including Voronoi-Clip (V-Clip) and SWIFT. In real-time simulations objects tend to change their orientations or rotations by small amounts from one frame to another. This is called frame-to-frame coherence. This coherence assumes that the closest points between two non-intersecting objects are located in the near vicinity of the closest points between the same objects located at the previous frame. However, for a polyhedron, even a minute change in orientation can cause a big change in closest points' locations between consecutive frames. Therefore, for polyhedra, Lin et al. [9] proposed using the closest features (vertices, edges, or faces) rather than tracking closest points from one frame to another. Then the frame-to-frame coherence theorem for closest features becomes: "A pair of features from each of two disjoint polyhedra are said to be closest features if they contain a pair of closest points for the polyhedra." Lin-Canny has some drawbacks. It does not terminate when presented with penetrating polyhedra. The other drawback is that it sometimes exhibits poor convergence in degenerate situations. Another algorithm based on Lin-Canny is V-Clip which eliminates some serious defects of Lin-Canny.

The V-clip algorithm operates on a pair of polyhedra. It is based on the theorem which defines the closest points between the two polyhedra in terms of the closest features of the pair of polyhedra. Figure 1 shows pair of 3D polyhedra satisfying the theorem. *F(X)* and *F(Y)*are closest pair of features and *P(X)* and *P(Y)* are closest pair of points (not necessarily unique points) between two polyhedra X and Y. Red lines indicate Voronoi region for object X and yellow lines indicate Voronoi region of polyhedron Y. V-Clip starts with two features one from X and another from Y. In this example, feature F(X) is edge E and feature F(Y) is vertex V. If P(X) is in Voronoi region of Y and P(Y) is in the Voronoi region of X, the F(X) and F(Y) are closest pair of features. In each iteration of the algorithm, the features are tested to see if they are satisfying the conditions of the theorem. If they satisfy the theorem, algorithm terminates and returns nonintersecting between two polyhedra. If the theorem is not satisfied, one of the features is updated with a neighboring feature. Neighbors of a feature are defined as follows: the neighbors of a vertex are the edges incident to the vertex, the neighbors of a face are the edges bounding the face, and the neighbors of an edge are the two vertices and the two faces incident to the edge.

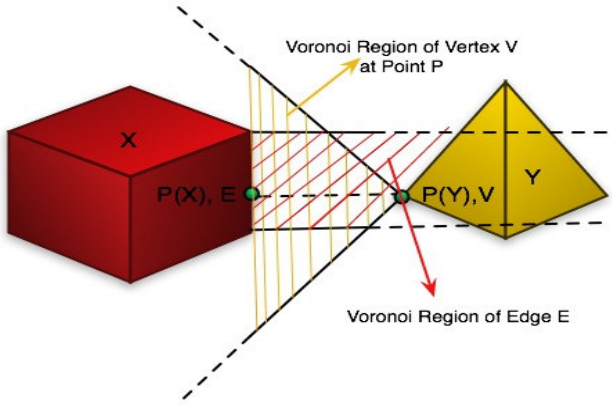

Figure 1 Closest pair of features and closest pair of points (P(X), P(Y)) for Vertex-Edge feature pair

The V-Clip become trapped in a local minimum in the vertex-face state where the vertex lies below the supporting plane of the face and at the same time lies inside all of the Voronoi planes of Voronoi region. That can cause objects penetrations before collision detection.

Simplex Based Algorithms

The simplex is the convex hull of an affinely independent set of points. The GJK (Gilbert-Johnson-Keerthi) [26] is the well known ancestor of this group of algorithms [27]-[30]. Simplex based algorithms works by incrementally improving over a simplex. Each incrementally improving step approximates the configuration space of the object.

GJK is a simplex-based that takes two sets of vertices as input and finds the Euclidean distance and closest points between the convex hulls. A simplex is a convex hull which has property that removing a point from it reduces the dimensionality of the simplex by one. Thanks to Gilbert et al. [31], GJK was generalized to be applied to arbitrary convex point sets, not just polyhedra. An important fact in GJK is that it does not operate on the two input objects; however, it operates on the Minkowski difference between the objects. Minkowski difference provides transformation of the problem from finding the distance between two convex sets to that of finding the distance between the origin and a single convex set. The GJK searches a sub-volume of the Minkowski difference object iteratively (each sub-volume being a simplex). We take a cue from the work of Ericson et al. [3] and clarify the GJK algorithm.

Let A and B be two convex point sets and *x* and *y* two position vectors corresponding to pairs of points in A and B respectively. Then Minkowski sum of two sets A and B in Euclidean space, $A \oplus B$ defined as $A \oplus B = \{x + y : x \in A, y \in B\}$ where $x + y$ is the vector sum of the position vectors *x* and *y*. The Minkowski difference also uses the same analogy which is defined as $A \Theta B = \{x - y : x \in A, y \in B\}$. The GJK algorithm based on the fact that separation distance between two convex polyhedra A and B is equal to the

distance between Minkowski sum and the origin as shown in Figure 2 [3] below.

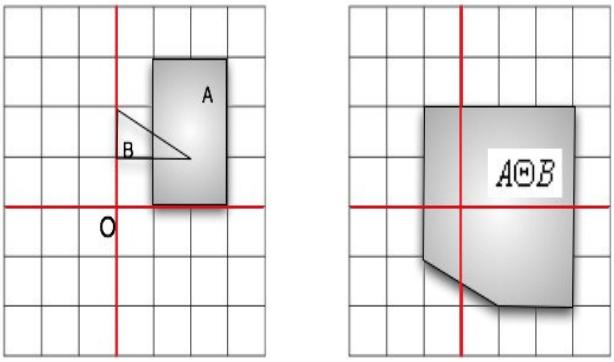

Figure 2 Minkowski Difference

Minkowski difference is important for collision detection point of view because two convex objects collide if and only if their Minkowski difference contains the origin as shown in Figure 2. Moreover, closest points' distance of two objects is equal to the minimum distance of Minkowski difference object to the origin. Therefore;

$$|AB| = \min\{\|x-y\| : x \in A, y \in B\} = \min\{\|c\| : c \in A \Theta B\}$$

Figure 3 [3] illustrates how GJK algorithm finds a point closest to origin *O*. In this case, the distance of closest point to the origin is equal to the minimum distance between two convex polyhedra due to the Minkowski difference being used as a convex hull.

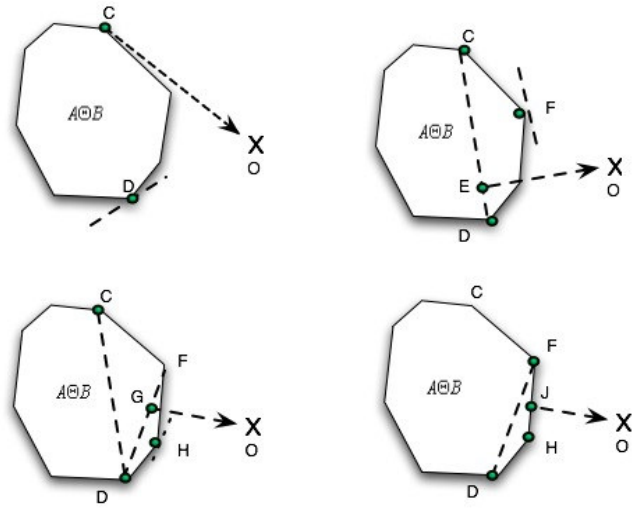

Figure 3 GJK Algorithm

In Figure 3, the algorithm arbitrarily begins with vertex C as the initial simplex set *Q={C}*. For a single-vertex simplex, vertex itself is the closest point to the origin *X*. Searching in the direction (from vertex C to the origin) leads to vertex D as a supporting point or extreme point at this direction. So, D is added to the simplex set *Q={C,D}*. The point in convex hull Q closest to the origin is E now. Since both C and D are needed to express point E, we keep these vertices in the simplex *Q={C,D}*. Now F is the extreme point in the direction from E to the origin. That results new convex hull Q, *Q= {C,D,F}*. The closest point to the origin from convex hull Q is now point G. Since representing point G is possible with only D and F, C is removed from the simplex, *Q={D,F}*. Now, direction of the supporting vector is from point G to the origin and new extreme point is H. H is added to the simplex Q, *Q={D,F,H}*. The point on Q closest to the origin is now J. Since F and H are smallest set of vertices to represent J, D is removed from Q, *Q={F,H}*. After this point, there is no vertex closer to the origin in direction from J to the origin. Therefore, J must be the closest point to the origin and the algorithm terminates.

Image-Space Based Algorithms

With the improvements of the GPU graphics hardware, the GPU can be used to accelerate collision detection. Image space base techniques are computed by image-space occlusion queries which are convenient to implement on the GPU. Therefore, image space base techniques are more preferred techniques to be implemented on the GPU. Contrary to common belief that they can be implemented only on the GPU, they can also be employed on the CPU such as [13]. Memory read-backs are expensive processes on the graphical units. [14] shows occlusion queries have lower bandwidth than buffer read-backs and thus are more convenient for GPU implementations. Some examples of image-space base techniques are based on stencil and depth testing which require memory read-backs and use graphics hardware. Frontiers of image space based methods include [15]-[18] and [19]-[23]. All image space based collision detection methods have several common drawbacks. They are much slower than hierarchical approaches. They usually have O(n) complexity. On the contrary, hierarchical approaches such as bounding volumes have O(log n) complexity. Since image space based approaches are rendering geometric primitives, these approaches introduce geometric errors. During the rendering, objects are discretized to the image space. This causes erroneous representations. These errors not only depend on the size of the viewport, but also the internal representation of the numbers, and the number of bits per pixel in the z-buffer. Therefore, the size of the viewport has significant impact on the performance.

Cinder [21] is a well known example of image space based algorithms that is founded on 3D version of Jordan Curve Theorem [4]. This theorem in computational geometry states that a semi infinite ray originating within a solid will intersect the boundary of the solid an odd number of times as in Figure 4. Cinder is handling both convex and non-convex geometries. The tests for collisions are performed in image space. The algorithm does not require any pre-processing or special data structures. It uses frame buffer operations to implement a virtual ray casting algorithm for every pixels that detect interference between objects. The edges of the objects are written to the depth buffer and the objects they penetrated each other are detected by using a virtual ray-casting algorithm. The virtual ray-casting is a technique that locates a point relative to an object by casting a ray from that point. The number of polygons that the ray passes

through is counted in such a way that if summation result for one ray is even then the point is outside the object. In contrast, if the summation result is un-even then the point is inside the object and there is collision. The algorithm uses a stencil buffer for counting the number of front and back facing polygons that the rays pass through. The values in the stencil buffer are increased for front-facing polygons and decreased for back-facing polygons. If at the end there is non-zero value in the stencil buffer, then the edge in the specific pixel is inside an object (meaning more front-facing polygons than back-facing ones). Colliding objects' identifications' numbers kept in color-buffer. The algorithm's running time is linear in the number of objects and the number of polygons existing in the objects. With this algorithm, collisions that are about to happen or have already occurred will not be detected. This occurs when objects' very thin parts pass through each other in space of one frame. There is a restriction on the object topology with this algorithm. The object must be closed. Figure 5 below shows an example of CINDER.

CULLIDE [24] is one of the prominent examples of these group of algorithms which uses occlusion queries and a hybrid approach. Here, the graphics hardware is used only to detect potentially colliding objects, while triangle-triangle intersections are performed in the CPU. CULLIDE uses clever but simple lemma to prune the non-colliding objects from possibly colliding objects' set. The lemma is: "An object A does not collide with a set of objects S, when A is fully visible with respect to S." CULLIDE keeps potentially colliding objects in a set which initially includes each and every object in the scene. Then it prunes the primitives from a potentially colliding set by rendering in a two-pass algorithm; first rendering front and then reverse order. Throughout the rendering, visibility (occlusion) queries remove objects from potentially colliding list if the object is not visible. This strategy continues iteratively until no more changes are made in potentially colliding set (PCS). The primitives in the final PCS are then made for exact collision detection. Boldt et. al [25] extended CULLIDE to handle self collision test. Even though this approach alleviates Cinder's restrictions on object topology; CULLIDE's effectiveness degrades dramatically when the density of the environment increases.

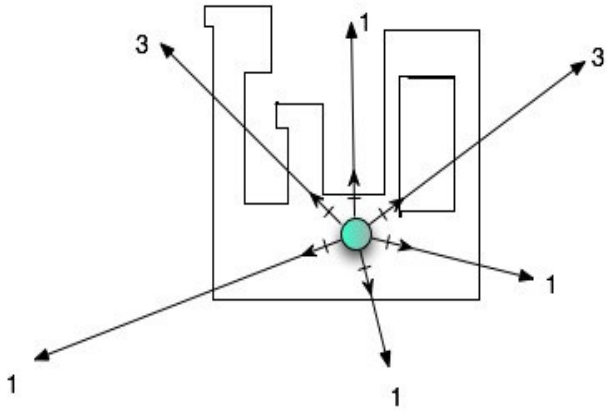

Figure 4. Jordan Curve Theorem

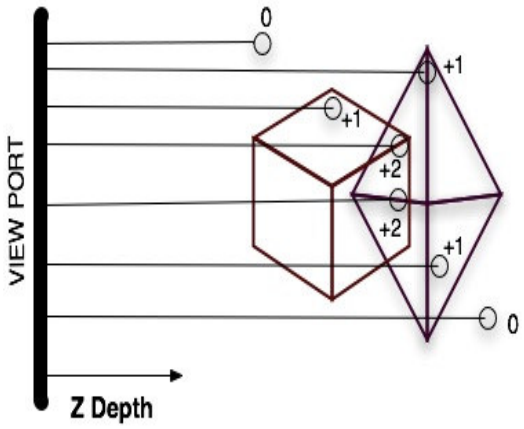

Figure 5. Cinder with Virtual semi-infinite ray casting

Volume Based Algorithms

Most of the volume based algorithms are conceptually founded on the same idea of the image space based techniques; however, they use different methods to compute such as layered depth images (LDI), distance fields etc. These groups of algorithms are also suitable for GPU implementations. Gundelman et. al [32] proposed one of the volume based collision detection algorithms. Assume A and B are objects in the scene and we are interested to know whether they are colliding or not. This algorithm works by taking vertices of A and looks them up in the signed distance function of B. After that, the vertices of B are looked up in the signed distance function of A. Each object is represented by a triangular mesh and the signed distance map. The algorithm in [32] is for rigid bodies. Thus, both the triangular mesh and signed distance grid are stored in an object space. This means that when vertices of A are looked up in B, then they must be transformed from object space A to object space B. The drawback of this algorithm is that it is not tailored for detecting edge-edge intersections.

Bounding Volume Hierarchies

There are two types of spatial data structures for collision detection: spatial division and bounding volume hierarchies (BVH). Spatial partitioning recursively divides the space in which objects are embedded. On the other hand, BVH recursively or iteratively partitions the object itself. With spatial partitioning splitting of polygons is unavoidable. This causes increase of depth of the tree and so performance is lost. In addition, since cell size of the spatial partitioning cannot cover objects' primitives tightly; determining contact status when objects are close is difficult.

BVHs are more applicable for general shapes than simplex based and feature based algorithms. They provide smaller and tighter hierarchies than spatial partitioning. This is very useful from the accuracy point of view. BVH can be called as discrete representation of level of details of objects. At first level, hierarchy includes one bounding volume and this is very coarse representation of an object. Further levels include more detail representations of the object. The leaf level or

finest level of the hierarchy generally includes the object primitives (lines, triangles, or tetrahedra). There is a parent-child relationship between succeeding levels with the topology of the tree. Bounding volume (BV) does not necessarily enclose its children's bounding volume; instead it must enclose the geometry of an object that is included in the children BVs.

BV Traversal: Traversal is testing whether two BVHs are overlapping or not. If the root BVs are not overlapping, then we conclude that two objects are not overlapping. Unfortunately, we cannot say the opposite because even though objects are not colliding, their BVs can collide. We must look further down the BVH to answer the question of whether objects are colliding or not. We do this by changing one of the root volumes by its children. This is called descending. Determining which root volume to descend is called traversal rule. Generally largest volume is chosen to descend to lower the chance of finding overlapping. If two volumes are equal then random choice is made. Non overlapping BVs are determined by the pair-wise tests between BVs and next, those non overlapping ones are discarded from further consideration. This is called pruning. At last in the traversal, if we reach the two leaf nodes from two distinct volumes, then we have two choices; whether testing two primitives are colliding or testing one primitive with the other's leaf bounding volume. The first one tests pair-wise primitives and returns precise description of the collision. This method has less iteration but more complexity. The second method skips the testing primitives rather does primitive leaf BV test. This method (primitive-volume test) has more iteration but has less complexity (if the objects are not actually colliding). This has an advantage when two objects' primitives are not colliding since skipping the expensive primitive tests. However, if objects' primitives are colliding, after second methods' primitive BV overlapping test, we have to test two primitives anyway. Thus, there is a tradeoff between the number of iterations and the complexity in the overlap testing.

Gottschalk et al. [33] states that recursive approach is often a bad choice since the number of primitives and of course the hierarchies can be quite large. Therefore, number of recursive calls would be huge that causes memory stack overflows. This problem can be solved by using iterative traversal technique with first-in-first-out queue. In this method, initially the root BVs are pushed into the queue then loop starts by popping two root volumes from the queue and standard overlap test is applied. If two root volumes overlap, descended volumes are pushed into the queue. Then those descendents are processed further in the loop and this process goes on iteratively. Alternative traversal orders such as breadth-first (pre-order) or depth-first (post-order or in-order) can be used .

The idea of using queue to escape from disadvantage of recursive nature of the algorithm is taken one step further by [34] and [35] by introducing a priority on the pair-wise BV tests. This is useful for time critical collision detection. The idea behind this is that there will be more than two objects possibly colliding in the scene. All those possibly colliding objects are determined in the broad-phase. Thus, all pairs of root passed from broad-phase are pushed to the queue and given a priority. Priority queue based traversal algorithm runs until a certain threshold time is reached. When the time is up, objects are determined as colliding if they are not pruned yet.

BV Update: Object space (model frame) is fixed with respect to the object but changing with respect to the world coordinate system by the movement. When objects are in the simulation environment, they are in the world coordinate system (WCS) which is fixed global coordinate frame. BVHs are generally created from object space; thus, are also represented in the model space. When we perform intersection tests between different objects, we need to bring those hierarchies to the common representation ground. For this purpose, there are two methods: the first is transforming one object's hierarchy to the other object's frame, the second is transforming both objects' hierarchy into the WCS. In model space update scheme, since there is only one bounding volume that needs to be updated, this scheme provides a performance and a fitting advantage. In contrary, since WCS update requires both BVs in the scene to be updated, it is costly. Therefore, model space updating is preferable to WCS updating.

BVH aligned to a specific frame such as axis aligned bounding box (AABB) needs to be realigned if rotation occurs. Some BVHs such as spheres and oriented bounding boxes are not aligned; therefore, do not require refitting. Spheres have a unique property that they are completely independent of the orientation. Only their positions need to be updated with the movement. Rotations turn AABB to an oriented bounding box (OBB). Therefore, to update AABB involves coordinate transformation and realignment. This problem can be solved in two ways as illustrated in Figure 6: one is placing AABB around OBB which is actually rotated version of the initial AABB (wrapping AABB around OBB middle image in the Figure 6). The second method is refitting the AABB for the rotated object. Notwithstanding, even though refitting provides tighter fit and early pruning capability, it is not preferred for computationally expensive simulations.

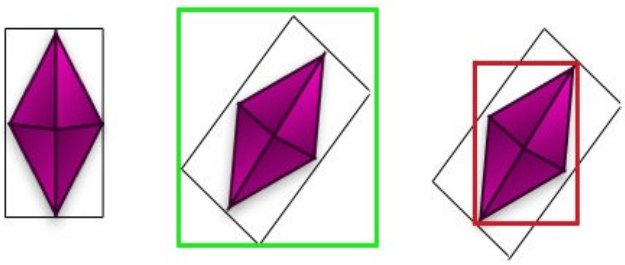

Figure 6 Updating Bounding box w.r.t. rotation, (a) AABB of the geometry, (b) Wrapping AABB around OBB, (c) Refitting of AABB

Different hierarchy update schemes need to be considered for rigid and deformable bodies. Deformable bodies usually represented at WCS cause misaligned

initial BVHs. Thus, deformable objects require different kinds of updates involving rebuilding, rebalancing, and refitting of BVH.

Complexity vs Performance: An ideal BVH preferred to be small size, to have small height, to have good pruning capabilities, and to be balanced. Small heights and balanced trees result in performance gains. Balanced trees provide a good overall worst-case seeking complexity. Good pruning capabilities imply good performance of a collision query. That means that BVs comparisons should have a capability of early rejection as much as possible. This early rejection criterion depends on the topology choice of the hierarchy (spheres, OBB, AABB etc.) and tighter fitting. Tighter fitting provides less space usage and early rejection. In overall, there is a tradeoff between BVH complexity and performance as shown in Figure 7 [33]. Complex topology choices establish tighter fitted BVs and so fewer overlap tests but causes performance lost. On the other hand, less complex hierarchies provide faster overlap test but less tight BVs. Unfortunately, the cost of performing an overlap test between two complex geometries increases with the complexity. Therefore, using a simpler geometry with cheaper overlap test is preferable for highly dynamic environments even though this will cause unnecessary overlap tests. The measurement of the tightness of the BV is presented by [36].

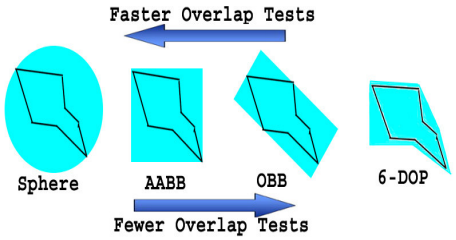

Figure 7 Tradeoff between complexity and performance for BVs [33]

The topology of the BVH has a large impact on the collision detection query. Usually balanced binary trees are desired since they have good search properties. However, Mezger et. al [37] proved that quad-trees and/or octrees act significantly faster than binary trees for collision queries. Gottschalk et. al [33] proved that OBBs are superior over spheres and AABBs for surface based BVHs. AABBs are preferred choice of deformable objects [29][38] since AABBs are cost effective for frequent refitting operations in deformations. Thus, AABBs overperform OBBs even though OBBs have better pruning capabilities. However, for volume based methods, spheres are the most suitable choice [34][39][40]. To increase the pruning capability of the BVH, BV should be short and fat rather than being long and thin. The long and thin topology increases the chance of overlapping BV with the other BVs which causes performance lost. In this sense, spherical or cubic BVs will reduce possible overlapping with other BVs which implies a better chance for pruning.

BVH construction: There are three main approaches for hierarchy construction which are top-down, bottom-up, and incremental methods. Top-down methods are the most popular methods. They are especially useful when there are no geometry changes. Bottom-up approaches are more complicated to implement and have a slower construction time; however, they usually produce the best trees. On the other hand, incremental methods (or insertion methods) generate BVH incrementally. They are not widely used methods though. Tree is constructed by inserting one object at a time in the insertion method. Insertion methods have two main concerns: finding a place to insert a BV and then updating the hierarchy.

Top-down approach implies fit and split strategy where BV is fitted to cover all the primitives and the BV partitioned and this continues until each BV contains a single primitive. Fitting answers the question of "In a given BV topology and the geometry, how the smallest and tightest BV is found". Partitioning strategies answer "In a given degree, d, and the geometry, how the geometry is divided into d groups".

Partitioning Criteria

These methods have certain criteria which are min-sum, min-max, and longest side etc. *Min-max* stands for minimize the sum of the volumes of the child volumes. The probability of an intersection between a BV and either of the two child volumes are expected to be proportional to their volume. Hence, minimizing the sum of the volumes minimizes the likelihood of intersection. *Max-min* approach is choosing the axis that minimizes the larger of the volumes of the two resulting children. Although the previous approach results in one volume much larger than the other, this (max-min) strategy makes the larger volume as small as possible which leads to more equal volumes. Longest side method is selecting the axis along the longest BV. The last method is not suitable for spheres; however, suitable for OBBs, AABBs, k-DOPs BV types. We can extend Klosowski's criteria in [41] with Min-int, Max-sep, and Divide equally criteria. *Min-int* is another strategy that minimizes the volume of the intersection of the child volumes. This approach has the potential to decrease the probability of both children being overlapped and traversed into. *Max-sep* is a strategy of the separation of the child volumes. Even though children are not overlapping, separation of children can further decrease the probability of both children being traversed into. Another strategy is dividing primitives equally between the child volumes. This strategy is also called as median-cut algorithm. In this algorithm, the BV is divided in two equal size parts with respect to their projection along the selected axis which results in balanced trees.

Fitting

It is also called finding smallest enclosing volume. There are many types of bounding volumes presented in the literature. However, spheres, AABBs, and OBBs are most widely used ones.

Welzl algorithm to find minimum bounding sphere is presented in [42]-[44]. Welzl algorithm is applicable to both bounding circles in 2D and bounding spheres in 3D. Welzl algorithm does not compute the minimum sphere

bounding the other spheres. Solution to this problem is given in [44].

Splitting

BV splitting methods (or partitioning methods) are usually based on the idea of: first; determining splitting axis to indicate a direction of splitting, second; calculation of splitting point on the already determined splitting axis, third; defining splitting plane or dividing plane by making splitting axis' direction as normal vector of the splitting plane and containing splitting point on the plane. Splitting stops or sometimes fails when some particular stop or fail criterion is reached. Stop criteria are for instance; the node that contains only a single primitive, the BVs that reach the certain threshold volume, or depth of the hierarchy that reaches the predefined depth. Fail criteria results in early termination of the algorithm they include situations where all primitives fall on one side of the split plane, child volumes or one of them ends up with nearly as many primitives as the parent volume, or child volumes are as large as the parent volume.

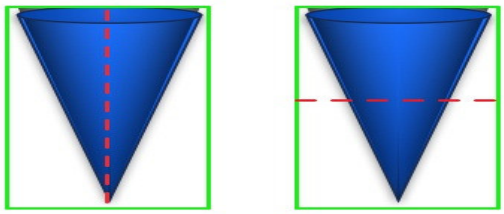

Figure 8 a) Bad splitting (left), b) Good Splitting (right) examples

Splitting Axis

Klosowski et. al [41] proposed to choose splitting plane orthogonal to local x, y, or z coordinate axis. This is very simple and fast approach. Longest side method is another approach for finding splitting axis by selecting the axis along which BV is longest. Another simple method is using already existing axis of bounding volume as splitting axis. In this case BV cannot be sphere since spheres do not have any associated axis. The last and the most accurate method is statistics based method which aligns the splitting axis along the axis where the covariance is the largest.

Splitting Point

Subsequent to splitting axis determination split point needs to be chosen since there are infinite numbers of possible splitting points along the splitting axis. Again the choice for the point must be restricted to a small set of points with the following strategies: object median, object mean, spatial median, BV projection, and centroid coordinates. Object median method is splitting at the object's median calculated from primitives' centers. Cormen et. al [45] reduced the cost of median finding by using a sophisticated method. Klosowski et. al [41] reports that using the object mean is superior to using the object median since resulting smaller volume trees with a lower number of operations performed and better query times. Splitting at the spatial median is another method which is equal to splitting the volume to two equal parts. In this technique, split point is found in constant time because this method only deals with the BV not the data included in the BV. This approach is used when the axis is selected from the parent volume such as that used in the longest side rule. BV projection method splits BV projection into evenly spaced points and instead of spending the time for intelligent point guessing, it spends the time for finding best splitting point among those evenly spaced points by using brute force methodology. The centroid coordinates method finds splitting point between random subset of the centroid coordinates. Figure 9 shows some of these methods.

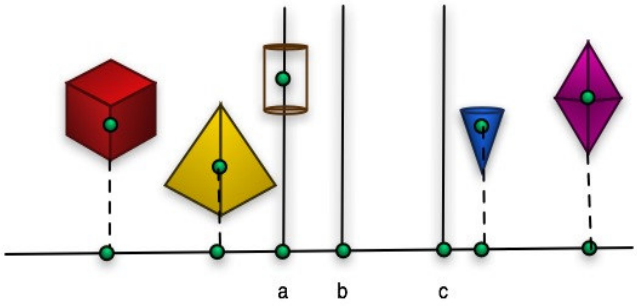

Figure 9 Splitting at (a) the object median, (c) the object mean, (c) the spatial median

BVH for Deformable Objects

Deformable objects are very challenging for BVH. Deformations can make representation of the object's geometry with BVH useless. Since it is impossible to reconstruct the BVH for each time step from scratch for complex objects and deformations, it is necessary to find better approaches. [46]-[50], and [37] investigate faster approach to refit the currently misaligned BVH tree by using bottom-up update scheme. There are certain drawbacks of these studies. One is coming from the nature of the bottom-up scheme that an update requires one to traverse the entire BVH. The second problem with the bottom-up update is that not all BV types can be updated very fast. The third problem with bottom-up scheme is that pruning capabilities of BV can be damaged during the update due to a possible increase among sibling BVs. Instead of bottom-up scheme, top-down scheme can be used in certain situations; however, both have deficiencies at times. Therefore, Larsson et. al [38] proposed hybrid approach to benefit from both schemes' advantages. Recently, deformable spanners [53] have been proposed to encode all proximity information which may turn out to be very useful for deformable and rigid bodies' collision detections. Deformable spanners are geometry independent (means that the geometry can be open, close, convex or concave) proximity queries with different resolutions that makes the deformable spanners very useful for surgical simulations. However, it may be difficult to get real time performance out of the deformable spanners without further algorithmic improvements.

Ultimate goal for collision detection researchers is development of new algorithms that handle rigid and deformable bodies, self collisions, convex and non-convex or open and close geometries and process all in

real-time. To that end, deformable spanner can be a good solution for abovementioned requirements. Thus, we will introduce an example construction of deformable spanner in the next section.

IV. DEFORMABLE SPANNER CONSTRUCTION

In order to have concrete grasp of deformable spanner's hierarchy (called balls hierarchy in this study) we need to exploit hierarchy construction steps which are illustrated by step-by-step example where minimum radius r =0.3, expansion ratio ξ =3, and neighbor distance η = 3.2 are selected. For simplicity of explanation, 3 static 2D points are chosen which are {4,4}, {4,5}, and {5,5} respectively. Node IDs are given for avoiding confusions on nodes at each level.

Construction process is started with an arbitrarily chosen point. Each arbitrary chosen point leads to a different hierarchy structure; in other words hierarchy is canonical, meaning that with different insertion orders equally good deformable spanner can be constructed. Figure 10 shows insertion process of the first point P_0={5,5} and corresponding tree hierarchy.

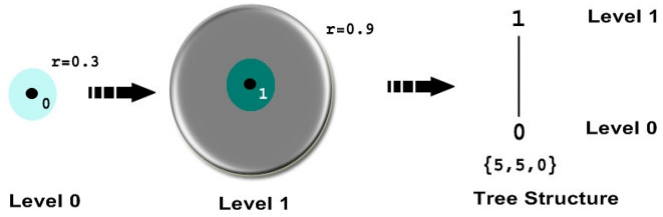

Figure 10. Insertion of first point

Since there is no other point inserted in the hierarchy yet, P_0 is parent of itself in the first level (Level 1) with 0.9 radius (minradius* ξ^{Level}). Bottom level (Level 0) with radius 0.3 includes P_0 itself. Second point to be inserted is P_1={4,4}. First, the perimeter of radius 0.3 is searched (see Figure 11). Due to the absence of points in this perimeter, P_1 goes one level up. Now, the new perimeter for P_1 is 0.9. This is the new range that will be searched. Since there is no point involved in this perimeter of P_1, P_1 will be (node id 4) parent of itself. This creates virtual edge which is represented as vertical straight line in the illustrations. Following the insertion, edge between nodes 1 and 2 is created since distance between nodes with IDs 1 and 2 is less than neighbor coefficient η times the radius at that level which is 0.9 (<=η *0.9) (see Figure 11, white straight line). Hierarchy construction process is still incomplete for two points. After second points' insertion, root of the tree must be determined. When the situation like equality holds between two nodes which are competing to be new root or parent, one of them is arbitrarily chosen. Whichever has fewer children , can be chosen as a root or parent of the other one. In this situation, importance of Lemma 4, 5, 6, and 7 become more obvious. For instance, a node which has already reached or almost reached its maximum number of children can be skipped in selection process as a root or parent in the implementation.

In our illustrations in Figure 11 and Figure 12, node 2 is arbitrarily chosen as a new root. Node ID of this new virtual node is 3. Figure 12 shows insertion of P2={4,5}. First, vicinity of P2 with radius 0.3 is searched. Since there is no point in the perimeter, P2 is allowed to go to upper level (Level 1). It is now parent of itself with node ID 5 and radius 0.9. At that level, node 5 is neighbor of nodes 2 and 1 since distance among them is smaller than η*0.9. There will be neighbor edge among nodes 1,2, and 5 which are illustrated as dashed lines in Figure 13. Corresponding tree representations are also shown in the same figure. Node 5 is included in the node 3's radius which is 2.7 (0.3*ξLevel 2 = 0.3*(3)2 = 2.7); thus, it is not permitted to go to the upper level.

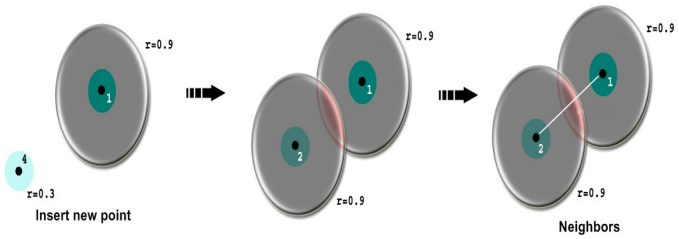

Figure 11. Insertion of second point and neighbor relation

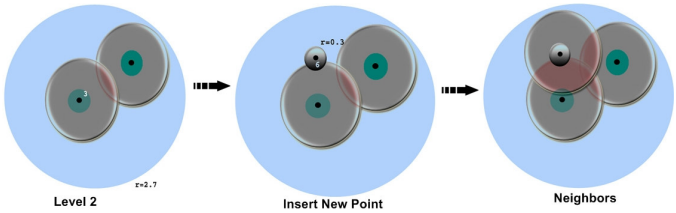

Figure 12. Insertion of third point

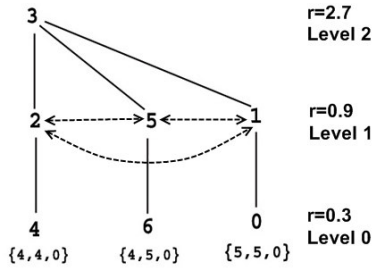

Figure 13. Final tree structure

An efficient neighborhood coefficient η, expansion ratio ξ, and minimum radius r selection process has profound impact on the balls hierarchy. These parameters will be application specific; therefore, these parameters must be selected with careful analyses or using trial and error procedure. A small value for the neighborhood coefficient makes hierarchy enormous and inefficient; on the contrary, a large value of the coefficient adds many useless edges to the hierarchy. In order to perform a spanner construction, a user must be able to estimate the minimum radius of the hierarchy at the bottom level

which will affect the efficiency of the application. Smaller radius values result in larger hierarchies; however, it provides better accuracy and pruning capabilities during hierarchy traversal. Expansion ratio (ξ) has also very important impact on the hierarchy. It affects the efficiency of the spanner. Smaller values of expansion ratio also increase the unnecessary number of edges included in the hierarchy and makes hierarchy extremely large. However, larger values of expansion ratio can result in faster construction time and lower the accuracy and pruning capabilities. If point sets are geometrically close to each other, hierarchy level will decrease. However, number of children belonging to a parent will increase. If points are far away from each other, hierarchy level and virtual edges (vertical lines in) increase whereas average number of children decrease.

V. CONCLUSION

We have classified collision detection algorithms into two groups; broad and narrow phase. Spatial partitioning and bounding volumes are well-known examples of the broad-phase. Feature based (FB), simplex based (SB), ISB, volume based, and bounding volume based algorithms belong to a group of algorithms in narrow-phase. FB approach only works for closed objects and it is not known how algorithm behaves in degenerate conditions. Problem with FB algorithms is that it does not terminate when presented with penetrating polyhedra. SB methods are only for convex objects. ISB techniques require close objects in the scene and cannot detect self collisions. Also, collisions that are about to happen or have already occurred will not be detected in ISB. Therefore, FB, SB, and ISB are not considerable for dynamic simulations and deformations e.g. surgical simulations. Those methods cannot handle open objects which occur during the surgical procedures such as incision. With spatial partitioning, splitting of polygons is unavoidable and determination of cell size is very difficult. Therefore, when objects are close, determining contact status is difficult. BVHs are generally most suitable for collision detections but they are highly dependent to the topology choice of the hierarchy. There is a trade off between complexity of the topology of BV and construction cost. For deformable objects, BVs require different kind of update involving rebuilding, rebalancing, and refitting of BVH. There is still no well-informed hierarchy update scheme for deformations in that deformable spanners may introduce.

ACKNOWLEDGMENT

The authors wish to thank Dr. N. Samadi for his valuable comments and feedback.

REFERENCES

[1] P. M. Hubbard, "Interactive Collision Detection", In Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality, 1993, pp.24-32.

[2] Physics-Based Animation, Kenny Erleben et al. Charles River Media, 2005

[3] Real-time Collision Detection, Christer Ericson, Morgan Kaufman, 2005.

[4] Computational Geometry in C, O'Rourke, 2005.

[5] D. Baraff, A. Witkin, J. Anderson, and M. Kass, "Physical Based Modelling", SIGGRAPH Course Notes, 2003.

[6] M. C. Lin, and D. Manocha, "Efficient Contact Determination between Geometric Models", Technical Report TR94-024, The University of North Carolina at Chapel Hill, Dept. of Computer Science, 1994.

[7] B. Mirtich, "Impulse Based Dynamic Simulation of Rigid Body Systems", Phd. Thesis, University of California, Berkley, 1996.

[8] M. Moore, and J. Williams, "Collision Detection and Response for Computer Animation", In Computer Graphics, vol. 22, pp. 289-298, 1988.

[9] M. Lin and J. Canny, "A fast Algorithm for Incremental Distance Calculation", Proc. of the 1991 IEEE International Conference on Robotics and Automation, pp. 1008-1014, 1991.

[10] B. Mirtich, "V-Clip: Fast and Robust Polyhedral Collision Detection", ACM Transactions on Graphics, vol. 17(3), pp. 177-208, 1998.

[11] S. A. Ehmann, and M. Lin, "Accelerated Proximity Queries between Convex Polyhedra by Multi-level Voronoi Marching", IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2000, vol. 3, pp. 2101-2106, 2000.

[12] S. A. Ehmann and M.C. Lin, "Swift: Accelerated Proximity Queries between Convex Polyhedra by Multi-level Voronoi Marching", Technical Report, Computer Science Dept., University of North Carolina at Chapel Hill, http://www.cs.unc.edu/~geom/SWIFT/, 2000.

[13] B. Benes and N. G. Villanueva, "GI_COLLIDE- Collision Detection with Geometry Images", In SCCG 2005, Proc. of the Spring Conference on Computer Graphics, pp.95-102, 2005

[14] oss.sgi.com/projects/ogl-example/registry/ARB/occlusion_query.txt, SGI 2005

[15] M. Shinya and M.C. Forgue, "Interference detection through rasterization", Journal of Visualization and Computer Animation 2, pp. 132-134, 1991

[16] K. Myszkowski, O.G. Okunev, and T. L. Kunii, "Fast Collision Detection Between complex Solids using Rasterizing graphics Hardware", Visual Comput. 11, pp. 497-512, 1995

[17] J.-C. Lombardo, M.-P. Cani, and F. Neyret, "Real-time Collision Detection for Virtual Surgery", in proc. of Computer Animation, Geneva, Switzerland, pp. 82-90, 1999

[18] G. Baciu and W.S.-K. Wong, "Hardware Assisted self collision for deformable surfaces", in Proc. of the ACM Symposium on Virtual Reality Software and Technology (VRST), ACM press, pp. 129-136, 2002

[19] G. Baciu and W.S.-K. Wong, "Image based techniques in a hybrid collision detector", IEEE Trans. On Visualization and Computer Graphics 9, pp. 254-271, 2003

[20] K. E Hoff, A. Zaferakis, M. Lin, and D. Manocha, "Fast and Simple 2D Geometric Proximity Queries Using Graphics Hardware", In Proc. Of ACM Symposium on Interactive 3D Graphics, pp. 145-148, 2001.

[21] D. Knott and D. Pai, "Cinder: Collision and Interference Detection in Real–time Using Graphics Hardware", In Proc. of Graphics Interface '03, 2003.

[22] B. Heidelberger, M. Teschner, and M. Gross, "Volumetric Collision Detection for Deformable Objects", Technical Report #395, Computer Science Dept., ETH Zurich, 2003.
[23] B. Heidelberger, M. Teschner, and M. Gross, "Detection of Collisions and Self-collisions Using Image-space Technique", In Proc. WSCG, pp. 145-152, Plzen, Czech Republic, 2004
[24] N. Govindraju, S. Redon, M. Lin, and D. Manocha, "CULLIDE: Interactive Collision Detection between Complex Models in Large Environments Using Graphics Hardware", ACM SIGGRAPH/Eurographics Graphics Hardware, 2003.
[25] N. Boldt and J. Meyer, "Self-intersections with CULLIDE", DIKU project # 04-02-19, the department of Computer Science at the University of Copenhagen, 2004.
[26] E. Gilbert, D. Johnson, and S. Keerthi, "A Fast Procedure for Computing the Distance Between Complex Objects in Three-dimensional Space", IEEE Journal of Robotics and Automation, vol. 4, pp. 193-203, 1988.
[27] S. Cameron, "Enhancing GJK: Computing Minimum and Penetration Distances Between Convex Polyhedra", IEEE Int. Conf. Robotics and Automation, vol. 4, pp. 3112-3117, Albuquerque, NM, USA,1997.
[28] G. v. d. Bergen, "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects", Journal of Graphics Tools, vol. 4(2), pp. 7-25, 1999.
[29] G. v. d. Bergen, "Proximity Queries and Penetration Depth Computation on 3D Game Objects", Proc. Game Developers Conf., 2001.
[30] G. v. d. Bergen, "Collision Detection in Interactive 3D Environments", Interactive 3D Technology Series, Morgan Kaufmann, 2003.
[31] E. Gilbert and Chek-P. Foo, "Computing the Distance between General Convex Objects in Three-dimensional Space", IEEE Transactions on Robotics and Automation, vol. 6, no. 1, pp.53-61, 1990.
[32] E. Gundelman, R. Bridson, and R. Fedkiw, "Nonconvex Rigid Bodieswith Stacking", ACM Transaction on Graphics, Proc. of ACM SIGGRAPH, 2003.
[33] S. Gottschalk, "Collision Queries Using Oriented Bounding Boxes", Phd. Thesis, Dept. of Computer Science, University of N. Carolina at Chapel Hill, 2000.
[34] C. O'Sullivan, and J. Dinglina, "Real-time Collision Detection and Response Using Sphere-trees", 1999.
[35] J. Dingliana and C. O'Sullivan, "Graceful Degradation of Collision Handling in Physically Based Animation", Computer Graphics Forum Proc. Eurographics, vol. 19, no. 3 , pp 239-247, 2000.
[36] G. Zachmann and E. Langetepe, "Geometric Data Structures for Computer Graphics", SIGGRAPH 2003 Course Notes, 2003.
[37] J. Mezger, S. Kimmerle, and O. Etzmuss, "Hierarchical Techniques in Collision Detection for Cloth Animation", Journal of Winter School of Computer Graphics (WSCG) vol. 11, no. 2, pp. 322-329, 2003.
[38] T. Larsson and T. Akenine-Moller, "Collision detection for continuously deforming bodies", in Proc. of Eurographics, pp. 325-333, 2001
[39] P. M. Hubbard, "Approximating Polyhedra with Spheres for Time-critical Collision Detection", ACM Transactions on Graphics, vol. 15, no. 3, pp.179-210, 1996.
[40] G. Bradshaw and C. O'Sullivan, "Adaptive Medial-xis Approximation for Sphere-tree Construction", ACM Transactions on Graphics, vol. 23, no. 1, pp. 1-26, 2004.
[41] J. T. Klosowski, "Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments", PhD thesis, State Univ. of New York at Stony Brook, 1998.
[42] E. Welzl, "Smallest Enclosing Disks (balls and ellipsoids)", In H. Maurer, editor, New Results and New Trends in Computer Science, LNCS, Springer 1991.
[43] M. D. Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, "Computational Geometry: Algorithms and Applications", Springer-Verlag, 1997.
[44] K. Fischer and B. Gartner, "The Smallest Enclosing Ball of Balls: Combinatorial Structure and Algorithms", Proc. of 19th Annual Symposium on Computational Geometry (SCG), pp. 291-301, 2003.
[45] T. Cormen, C. Leiserson, and R. Rivest, "Introduction to Algorithms", MIT Press, 1990.
[46] G. v. d. Bergen, "Efficient Collision detection of Complex Deformable Models Using AABB Trees"1997,
[47] P. Volino, and N. M. Thalmann, "Collision and Self-collision Detection: Efficient and Robust Solutions for Highly Deformable Surfaces", Technical Report, MIRALab, 1998.
[48] P. Volino and N. M. Thalmann, "Virtual Clothing: Theory and Practice", Springer-Verlag Berlin Heidalbarg, 2000.
[49] R. Bridson, R. Fedkiw, and J. Anderson, "Robust Treatment of Collisions, Contact and Friction for Cloth Animation", Proc. of ACM SIGGRAPH, vol. 21, no. 3, pp. 594-603, 2002.
[50] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, W. Strasser, "Collision Detection for Deformable Objects", Eurographics State-of-the-Art Report (EG-STAR), 2004, pp 119-139.
[51] B. Heidelberger, M. Teschner, and M. Gross, "Detection of collisions and self-collisions using image-space techniques". Journal of WSCG, vol. 12, no. 1-3, 2004.
[52] T. Larsson and T. Akenine-Moller , "A dynamic bounding volumehierarchy for generalized collision detection", Computers & Graphics, vol 30, no 3, pp 451-460, Elsevier Ltd, 2006.
[53] J. Gao, L. J. Guibas, and A. Nguyen, "Deformable spanners and applications", Computational Geometry: theory and applications, vol. 35, issue 1, pp. 2-19, 2006.
[54] K. Elissa, "Title of paper if known," unpublished.

S. Kockara was born in Sivas, Turkey. He received the B. Sc. in 2002 from Computer Engineering Department at Dokuz Eylul University, Izmir, Turkey and a Ph. D. degree from Department of Applied Science at University of Arkansas at Little Rock, AR, USA, in June 2008. He has been with the University of Central Arkansas at Computer Science Department since August of 2008 as a tenure track Assistant Professor.

His research interests include surgical simulations, collision detection, virtual and augmented reality, immersive environments, computer graphics, image processing, and biomedical engineering. He haspublished over 15 technical papers including 2 in IEEE Transactions. He also has a book chapter in Data Engineering: Mining, Information, and Intelligence published by Springer (2008). He has been a member of Alpha Epsilon Lambda, the National Honor Society for Graduate Students since May 2008.

T. Halic was born in Manisa, Turkey. He received the B. Sc. in 2002 from Computer Engineering Department at Marmara University, Istanbul, Turkey. He is currently a Ph. D. student at Department of Mechanical, Aerospace and Nuclear Engineering in Rensselaer Polytechnic Institute, Troy, NY, USA. He has been with the University of Arkansas at Little Rock, Applied Science Department since August of 2005 as a graduate student.

His research interests include surgical simulations, finite element modeling, and physics based soft tissue deformation by using finite element model, mass-spring model, and geometric models. He delved into details of general purpose graphical processing unit computations. He was graduated as an honor graduate student from the University of Arkansas at Little Rock.

C. Bayrak was born in Torul, Turkey. He holds a BS from Slippery Rock University in 1985, and a MS from Texas Tech University in 1989, and Ph.D. from Southern Methodist University in 1994 in Computer Science, TX, USA.

Currently is a professor in the department of Computer Science at the University of Arkansas at Little Rock. He also has been serving on the Faculty of Applied Science since 2000. His primary research is in the intersection of software engineering, component based development, data mining, and Biomedical Engineering. However, he also has interest in modeling and simulation, cellular automata, and monitoring and control. Dr. Bayrak has published over forty research articles in scientific conferences and journals, given tutorials at major conferences, and served on program committees for numerous international conferences and symposiums.

He is a senior member of IEEE and ACM professional organizations

K. Iqbal was born in Pakistan. He holds BE from Avionics Engineering at NED university of Engineering and Technology., Pakistan. He has Ph.D. and MS from Electrical Engineering at Ohio State University, Ohio, USA. He also holds MBA from the same institute. He is currently Associate Professor at Department of Systems Engineering in University of Arkansas at Little Rock. His research interests include control systems, biomechanics, motor control, postural stability, movement coordination.

He is a senior member of IEEE.

R. A. Rowe was born in Georgia, USA. Richard A. Rowe, M.D., is an associate professor for the Department of Neurosurgery in the College of Medicine and a neurosurgeon at Arkansas Cancer Research Center at the University of Arkansas for Medical Sciences (UAMS). He earned his medical degree at Emory University School of Medicine in Atlanta, Georgia and served a surgical internship at Emory University School of Medicine Affiliated Hospitals. He completed a research fellowship and a residency in neurosurgery at the Medical College of Georgia, Department of Surgery, in Augusta, Georgia. He also completed a fellowship in skull base neurosurgery at UAMS. He has investigated computer simulation technology in depth and has presented widely on the benefits of virtual reality simulators in various aspects of neurosurgery. He has also developed several software packages, including Microvascular Atlas of the Head and Neck and Neuroanatomy: An Electronic Atlas. He is certified by the American Board of Neurological Surgery and is a member of the American Medical Association and the Congress of Neurological Surgeons.