Fast Intersections for Subdivision Surfaces

Aaron Severn and Faramarz Samavati

Department of Computer Science
University of Calgary

Abstract. Subdivision surface intersections can be costly to compute. They require the intersection of high resolution meshes in order to obtain accurate results, which can lead to slow performance and high memory usage. In this paper we show how the strong convex hull property can lead to a method for efficiently computing intersections at high resolutions. Consequently, the method can be used with any subdivision scheme that has the strong convex hull property. In this method, a bipartite graph structure is used to track potentially intersecting faces.

1 Introduction

Surface intersection is a common problem with a variety of applications. It is integral to algorithms for computing Boolean operations between surfaces, which are an essential tool in Computer Aided Geometric Design (CAGD). The somewhat simpler problem of detecting surface intersection is necessary for collision detection, which has a wide variety of applications in animation, simulations, video games, and other areas. Much work has been devoted to the surface intersection problem for analytical surfaces [4, 5, 18] and NURBS surfaces [10–12], however few previous works have attempted to solve this problem for subdivision surfaces. Subdivision surface intersection often involves subdividing the control meshes to a desired resolution and then applying a mesh intersection algorithm, without considering properties of subdivision [2].

Today, meshes and subdivision surfaces are used in high end modelling packages [1], animation production [3], and have become part of the MPEG4 standard [16]. Subdivision is considered to be a fundamental building block in digital geometry and mesh processing [19]. We can easily find many libraries and data sets for meshes that represent various kinds of objects. By applying subdivision schemes to these meshes we can create smoother surfaces, and it is also possible to include sharp features in the results. Efficient intersection of these surfaces makes it possible to apply Boolean operations on meshes, creating many useful and new objects from the current objects (Fig. 1). Consequently, computing intersections on the hierarchy of meshes resulting from subdivision is an important task. For low resolution meshes this problem can be solved efficiently using existing mesh intersection techniques. However, in order to produce high quality results we need to perform these intersections on high resolution meshes which can be slow and memory intensive. Subdivision schemes lead to exponential

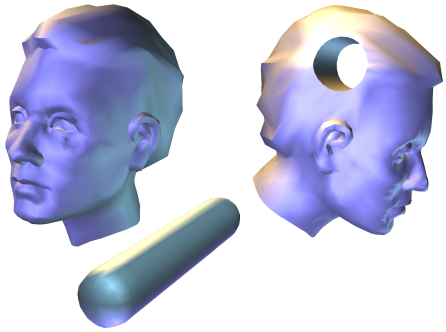

Fig. 1. Using subdivision surface intersection to compute a Boolean operation between two surfaces. The cylinder is subtracted from the man's head.

growth of the number of faces in a mesh and only a few iterations of subdivision can lead to meshes that are difficult to intersect due to their size.

In this paper we present a method for efficiently computing the intersection between two subdivision surfaces. The operations can be performed with arbitrary precision, but our focus is on intersecting high resolution surfaces efficiently. We will not consider the case of self-intersecting surfaces. Our main contributions are a fast face intersection exclusion test derived from the strong convex hull property of subdivision surfaces, a bipartite graph construction that makes it possible to identify and track which faces do not need to be tested for intersection, and a complete subdivision surface intersection algorithm built from these two results. We have implemented and tested our algorithm for Loop and Catmull-Clark subdivision. Our exclusion method can also be used for the case where the vertices are sent to the limit position.

Our method could be applied to any subdivision scheme where the strong convex hull property of subdivision surfaces holds. The method can be used for any existing meshes without additional pre-processing steps and could fit into the framework of existing CAGD systems that use subdivision surfaces.

2 Previous Work

Much work has been done on computing intersections between various types of surfaces. Several techniques have been developed for intersecting trimmed NURBS surfaces. Methods involving symbolic and numerical techniques have been used to trim the resulting surfaces [12], as well as mostly symbolic methods involving exact algebraic number representation [10, 11]. Hohmeyer [8] gives a framework for robust surface intersection based on loop detection.

The work of Epstein et al. [4], Goldfeather et al. [5], and Rappoport and Spitz [18] on efficient rendering of constructive solid geometry (CSG) objects presents a different approach to the problem of computing Boolean operations. For such surfaces, intersection is a trivial problem.

Lin and Gottschalk [14] give an excellent survey of collision detection algorithms. Collision detection between subdivision surfaces was discussed as a part of character animation by DeRose et al. [3]. Bounding hierarchies are commonly used when intersecting meshes, some examples of these include Gottschalk et al. [6] and Zachmann [22]. Interference detection on subdivision surfaces has been considered in works by Wu and Peters [21] and Grinspun and Schröder [7].

Our work on intersecting subdivision surfaces is most closely related to that of Lanquetin et al. [13], who developed a method for intersecting subdivision surfaces using their control meshes and bipartite graphs to minimize the cost. It should be noted, however, that their method can incorrectly compute the intersection in some cases, as they explained in their paper. Our use of bipartite graphs was inspired by their work, although our graphs are constructed differently.

Biermann et al. [2] developed a technique for approximating Boolean operations on free-form solids bounded by subdivision surfaces. In their work, they intersected the surfaces by performing midpoint subdivision on the control meshes and using a standard mesh intersection technique based on spatial subdivision. Our intersection approach could be employed in their Boolean operations framework. Litke et al. [15] developed ways for trimming subdivision surfaces, such trimmed surfaces are produced naturally through surface intersections. Several different approaches for producing sharp features or creases have been described in the past [2, 9, 20], which can be produced from intersection curves.

3 Excluding Face Intersections

For the remainder of this paper, the neighbourhood of face *f*, denoted *N(f)*, refers to the vertices of face f and all vertices of faces that share at least one vertex with *f*. A child of face f refers to any face produced by the subdivision of *f*. The convex hull of a set of vertices V will be denoted *co(V)*.

3.1 Convex Hull Intersection

The strong convex hull property for subdivision surfaces states that, for any face *f*, the children of f will be contained within *co(N(f))* (Fig. 2). This property holds where all masks' weights are positive. B-spline based subdivision schemes, including Loop, Catmull-Clark, polyhedral, and Doo-Sabin, are subdivision examples with this property. As a rare case, the convex hull property does not hold for Butterfly subdivision.

We can apply this property to the process of intersecting subdivision surfaces with the following observation. If C_1 and C_2 are control meshes of two subdivision surfaces and $f_1 \in C_1$ and $f_2 \in C_2$ are two arbitrary faces in those meshes, then the children of f_1 will be contained in $co(N(f_1))$ and the children of f_2 will be contained in $co(N(f_2))$. Therefore, if $co(N(f_1))$ does not intersect $co(N(f_2))$ we know that none of the children of f_1 will intersect any of the children of f_2, no matter how often each surface is subdivided. We now state this formally.

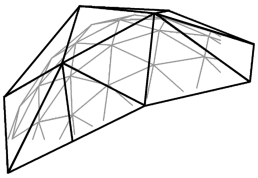

Fig. 2. The strong convex hull property illustrated for Loop subdivision. The fine mesh is contained within the convex hull of the coarse mesh.

Theorem 1: Let C_1^0 and C_2^0 be the control meshes of two subdivision surfaces, let C_1^i and C_2^i be the meshes resulting from subdividing C_1^0 and C_2^0 i times, and let $f_1^i \in C_1^i$ and $f_2^i \in C_2^i$ be two arbitrary faces produced from subdividing $f_1^0 \in C_1^0$ and $f_2^0 \in C_2^0$ i times respectively. If $co(N(f_1^0))$ does not intersect $co(N(f_2^0))$ then f_1^i does not intersect f_2^i.

This theorem gives us a strong condition for excluding the intersection of two faces. If $co(N(f_1^i))$ does not intersect $co(N(f_2^i))$ for $f_1^i \in C_1^i$ and $f_2^i \in C_2^i$ then we can exclude all children of f_1^i and f_2^i from intersection tests with each other. We show in our results that this can be applied to greatly reduce the number of face intersection tests when compared to other standard mesh intersection algorithms.

It should be noted that we do not need to compute any convex hulls in order to apply this result. If the distance between the convex hulls of the vertices of the neighbourhood is greater than zero then an exclusion has been detected, and this can be determined without computing the convex hulls. Rabbitz [17] gives an algorithm for computing the distance between two convex polyhedra using only the vertices of the polyhedra. If vertices of a non-convex polyhedra are given then the intersection is computed with the convex hull of those vertices. Since this algorithm takes only two sets of vertices as input it is well suited to our purposes. To determine if the children of faces $f_1 \in C_1$ and $f_2 \in C_2$ could potentially intersect, we can pass $N(f_1)$ and $N(f_2)$ to Rabbitz's algorithm.

Even though we have a relatively efficient way to test for intersection between $co(N(f_1))$ and $co(N(f_2))$, our experiments have found that this technique is only beneficial for extremely high resolution meshes. Although convex hulls are very good at reducing the number of face-face intersections performed, in most practical cases the expense of intersecting two convex hulls for each pair of faces results in a slower algorithm than a standard spatial subdivision approach. In order to make these ideas work at more practical resolution levels we need a much faster intersection exclusion test.

3.2 Alternate Intersection Tests

Any bounding volume around the convex hull can be used for excluding intersections. We have experimented with a variety of methods for bounding the convex hull of the neighbourhood of a face, including bounding spheres, oriented

bounding boxes (OBBs), and axis-aligned bounding boxes (AABBs). A tight radius for bounding spheres is hard to find and we have found that, with a larger radius, bounding spheres are not able to exclude enough face-face intersection tests. OBBs are very effective at excluding face-face intersection tests, however computing an OBB around the neighbourhood of a face is not trivial. OBBs are typically most effective when they can be pre-computed, but in our case they need to be computed on the fly each time a mesh is subdivided.

We have found that the best method of bounding the convex hull of the neighbourhood of a face, in terms of running time, is to use axis-aligned bounding boxes. For a face *f*, it is clear that the AABB of *N(f)* will contain *co(N(f))*, which leads to the following corollary.

Corollary: Let C_1^0 and C_2^0 be the control meshes of two subdivision surfaces, let C_1^i and C_2^i be the meshes resulting from subdividing C_1^0 and C_2^0 i times, and let $f_1^i \in C_1^i$ and $f_2^i \in C_2^i$ be two arbitrary faces produced from subdividing $f_1^0 \in C_1^0$ and $f_2^0 \in C_2^0$ i times respectively. If an AABB containing all vertices of $N(f_1^0)$ does not intersect an AABB containing all vertices of $N(f_2^0)$ then f_1^i does not intersect f_2^i.

This results in a highly efficient exclusion test with a tight enough bound to exclude the majority of face-face intersection tests.

4 Graph Based Intersection

We use a dynamic bipartite graph to keep track of only the faces that remain from the convex hull exclusion. In other words, this graph helps us to find all potentially intersecting faces. Our graph, $G = (V_1, V_2, E)$, consists of two sets of vertices, V_1 and V_2, and a set E of edges. Each vertex in V_1 represents a face in the control mesh C_1 and each vertex in V_2 represents a face in the control mesh C_2. An edge $e \in E$ connects $v_1 \in V_1$ to $v_2 \in V_2$ if the faces represented by v_1 and v_2 could *potentially* intersect. We call this the *potential intersection graph*. Using our result from the previous section, this means that if an AABB around $co(N(v_1))$ intersects an AABB around $co(N(v_2))$ then v_1 and v_2 are connected by an edge. Otherwise, they are not. The vertex sets V_1 and V_2 form the two partitions of the graph. As we are not considering self-intersection, there will be no edges between two vertices of the same set since two faces from the same control mesh will not need to be tested for intersection.

Upon subdividing C_1, each vertex of V_1 is split into n new vertices representing the new faces resulting from subdivision (in Loop or Catmull-Clark subdivision, for example, each vertex of V_1 is split into 4 vertices representing the 4 new faces resulting from the subdivision of each face). If an edge existed between $v_1 \in V_1$ and $v_2 \in V_2$ then edges will connect v_2 to each of the vertices resulting from splitting v_1 (Fig. 3). Subdivision of C_2 is handled the same way.

This dynamic graph structure is used to keep track of which faces need tested for intersection and which tests can be excluded. Each edge represents a face intersection test that has not yet been resolved. Using the quick exclusion test

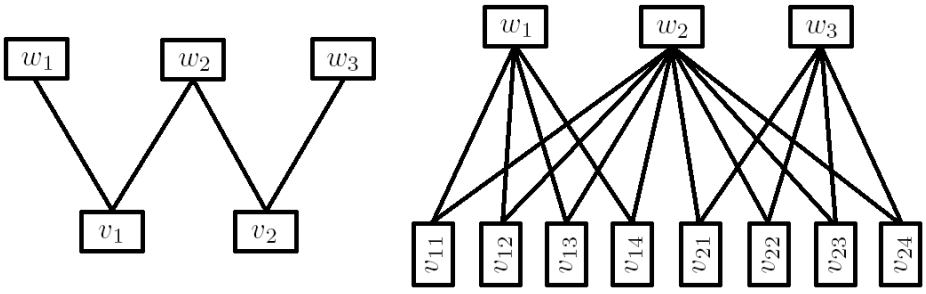

Fig. 3. A bipartite graph with potential intersections between v_1 and w_1, v_1 and w_2, v_2 and w_2, and v_2 and w_3 (left). The same graph after subdividing one partition (right).

described in Sect. 3, we can eliminate many edges from the graph each time a control mesh is subdivided using the algorithm described in the next section.

5 The Intersection Algorithm

We begin with the control meshes C_1^0 and C_2^0 of two subdivision surfaces and an integer n, the depth of subdivision at which the intersection will be computed. The choice of n will determine the accuracy of the intersection, with larger values leading to a more accurate intersection. Larger values of n will also lead to a slower intersection due to the exponential growth in the number of faces resulting from subdivision.

In the first step of the algorithm, we must compute an initial potential intersection graph G^0, having the form described in Sect. 4. Since we begin with no previous information on whether or not any two faces of C_1^0 and C_2^0 intersect, we must test each face of C_1^0 with each face of C_2^0. For each face $f_1^0 \in C_1^0$ and $f_2^0 \in C_2^0$ we compute $N(f_1^0)$ and $N(f_2^0)$ and then test if an AABB around $co(N(f_1^0))$ intersects an AABB around $co(N(f_2^0))$. If there is no intersection, we can conclude that there will be no intersections between the children of these faces. Otherwise we add an edge to G^0 from f_1^0 to f_2^0 (a potential intersection in further levels). Note that the construction of the AABBs is trivial since they depend only on the vertices of $N(f_1^0))$ and $N(f_2^0))$ respectively.

At this point we can use spatial subdivision to make the process of constructing the initial potential intersection graph more efficient. We compute the AABB of $co(N(f))$ for each face f of a control mesh and then place these in an axis-aligned hierarchical bounding box structure [22], one for each control mesh, where the leaf nodes are the AABBs. The initial graph can then be computed by intersecting the two bounding box trees, adding an edge whenever two leaf nodes intersect. This optimization is unnecessary for extremely coarse control meshes having few faces, but can be beneficial in other cases.

After computing the initial graph, we proceed in a loop where, for each iteration, both meshes are subdivided and then a new graph is created from the

old one. The new graph G^{i+1} is created from G^i by splitting the vertices of G^i in the manner described in Sect. 4. This produces a much larger graph, but many of the edges can be removed using the intersection exclusion test once again. If there is an edge $e \in G^{i+1}$ from face f_1^i to face f_2^i then we compute $N(f_1^i)$ and $N(f_2^i)$ and test for intersection between an AABB around $co(N(f_1^i))$ and an AABB around $co(N(f_2^i))$. If they do not intersect edge e is removed from G^{i+1}. We can further prune the graph by removing any isolated vertices that do not have any edges connected to them. The loop terminates when the desired subdivision depth has been reached (Fig. 4).

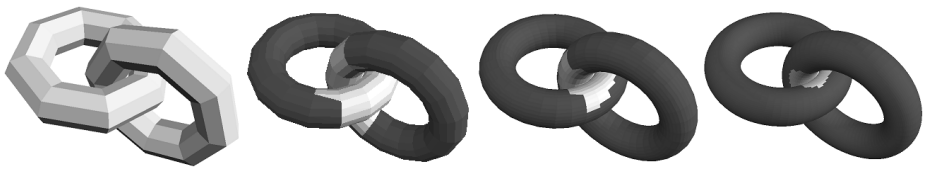

Fig. 4. The intersection algorithm proceeds from left to right, subdividing and updating the graph at each iteration. Faces still in the graph are highlighted. Note that most faces are excluded at a low resolution.

When the loop is complete we have two meshes, C_1^n and C_2^n, and a bipartite graph G^n, and we are ready to compute the final, precise intersection. To do this, we iterate over all edges of G^n and, if an edge connects f_1^n to f_2^n in the graph, we test f_1^n and f_2^n for real intersection.

One common method for producing a more accurate approximation to the limit surface after several applications of subdivision is to relocate the vertices of the mesh to the limit surface. Due to the fact that the limit surface is contained within the convex hulls of the neighbourhoods of the faces, this technique can be used with our algorithm before computing the real intersection without affecting the results of the computation.

6 Results

We have tested our algorithm on a wide variety of surfaces using both Loop and Catmull-Clark subdivision and have found that it leads to significant performance improvements, especially at high resolutions that are problematic in real applications. Table 1 gives a comparison of our method with a mesh intersection algorithm using spatial subdivision. The table gives the average number of face intersections that need to be performed for each algorithm. The number of subdivisions refers to the depth of subdivision at which the intersection is performed (the integer n in our algorithm). In the case of hierarchical bounding boxes, the control meshes were subdivided n times and the resulting surfaces were intersected. Fifty pairs of surfaces were intersected with the surfaces selected from a set of polygonal meshes. The meshes consisted of simple surfaces,

Subdivisions (n)	1	2	3	4
Catmull-Clark subdivision				
Hierarchical bounding boxes	48850	137330	397056	1179298
Bipartite graph with AABB exclusion test	52462	100827	198595	396781
Bipartite graph with convex hull exclusion test	13463	19899	25602	37905
Loop subdivision				
Hierarchical bounding boxes	48667	127348	357738	1036631
Bipartite graph with AABB exclusion test	82805	152086	288928	554449
Bipartite graph with convex hull exclusion test	23031	29522	39620	65865

Table 1. Average number of face-face intersection tests performed while intersecting 50 pairs of subdivision surfaces at various resolutions.

such as cubes and spheres, as well as arbitrary topology surfaces such as animals and human heads. The number of faces in the coarse meshes varied from 24 to 3336. Notice that, if we subdivide only once, spatial subdivision outperforms our method in the case of Loop subdivision and performs almost equally as well with Catmull-Clark subdivision. However, when subdividing four times our method is nearly twice as efficient in excluding intersection tests for Loop subdivision, and three times as efficient for Catmull-Clark subdivision. Our method also performs well with non-intersecting surfaces. In such cases all faces are excluded at a low resolution and no tests need to be performed on the high resolution meshes.

We have also included in the Table 1 the number of face-face intersection tests performed when using an exact convex hull intersection test to exclude faces from intersection. Clearly this gives a tighter bound than the AABB exclusion test, and this is reflected in the results. Using an exact convex hull test enormously reduces the number of face-face intersections that need to be performed. However, in practice, testing for intersection between two convex hulls is not fast enough for our purposes and leads to a slower algorithm.

We have experimented with a variety of different exclusion tests, including bounding spheres centred around faces, vertices, or edges, oriented bounding boxes, and exact convex hulls around face neighbourhoods, and we have found that most efficient method in terms of running time is the AABB test presented here. Figure 5 shows a comparison between the performance of our method and a hierarchical bounding box approach for a typical collision detection case resolved to various subdivision depths. As the resolution gets higher, our method performs significantly better due to the reduction in face-face intersection tests.

Also in Fig. 5 is a comparison of peak memory usage. Our method leads to a large reduction in memory usage at higher resolutions since we only need to store information about faces that could potentially intersect. After several subdivision steps, most faces have been removed from the graph.

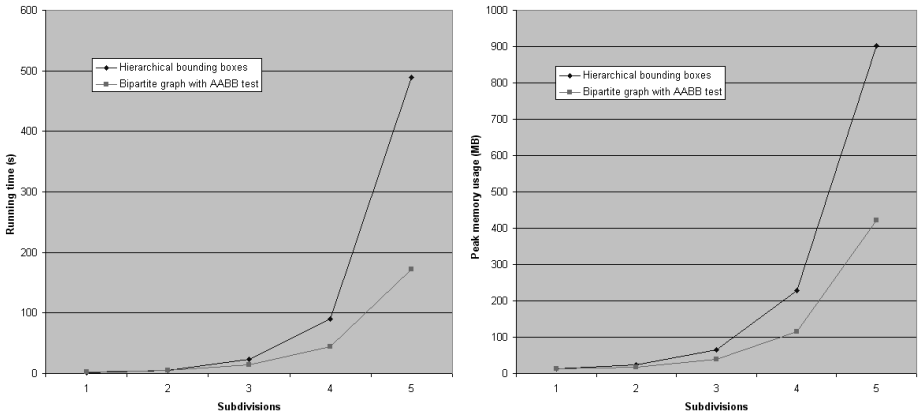

Fig. 5. A comparison of the running time (left) and peak memory usage (right) for a typical surface intersection at various subdivision depths using Loop subdivision. Two arbitrary topology surfaces, a dolphin (816 faces in the coarse mesh) and a duck (1818 faces in the coarse mesh) were used in the tests.

7 Conclusion

We have presented a method for computing the intersection of two subdivision surfaces with arbitrary precision that takes advantage of the strong convex hull property to produce an efficient algorithm. Our method can be used with any subdivision scheme where the strong convex hull property holds, and we have shown that it leads to a significantly more efficient intersection computation than existing methods for both Loop and Catmull-Clark subdivision. This method could be applied to improve the efficiency of several algorithms that involve intersecting subdivision surfaces, such as performing Boolean operations or collision detection.

Although we have investigated several intersection exclusion tests, it may be possible to find a more efficient test that improves these results. Our results indicate that many more face intersection tests could be excluded using the strong convex hull property if an appropriate exclusion test could be developed, however such a test must be quite quick in order to work effectively in this context. It is not clear if such a test exists and therefore merits further investigation.

References

1. ALIAS|WAVEFRONT: Maya. www.aliaswavefront.com (2002)
2. Biermann, H., Kristjansson, D., Zorin D.: Approximate Boolean operations on free-form solids. Proceedings of SIGGRAPH 2001 185–194
3. DeRose, T., Kass, M., Truong T.: Subdivision surfaces in character animation. Proceedings of SIGGRAPH 1998 85–94
4. Epstein, D., Gharachorloo, N., Jansen, F., Rossignac, J., Zoulos, C.: Multiple depth-buffer rendering of csg. Technical report, IBM Research Report (1989)

5. Goldfeather, J., Hultquist, J. P. M., Fuchs, H.: Fast constructive solid geometry in the pixel-powers graphics system. Proceedings of SIGGRAPH 1986 107–116
6. Gottschalk, S., Lin, M. C., Manocha D.: OBBTree: A hierarchical structure for rapid interference detection. Computer Aided Geometric Design **3(4)** (1986) 295–311
7. Grinspun, E., Schröder, P.: Normal bounds for subdivision-surface interference detection. Proceedings of the conference on Visualization 2001 333–340
8. Hohmeyer, M. E.: Robust and efficient intersection for solid modeling. PhD thesis, UC Berkeley (1992)
9. Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., Stuetzle, W.: Piecewise smooth surface reconstruction Proceedings of SIGGRAPH 1994 295–302
10. Keyser, J., Krishnan, S., Manocha, D.: Efficient and accurate b-rep generation of low degree sculptured solids using exact arithmetic: I-representations. Computer Aided Geometric Design **16(9)** (1999) 841–859
11. Keyser, J., Krishnan, S., Manocha, D.: Efficient and accurate b-rep generation of low degree sculptured solids using exact arithmetic: II-computation. Computer Aided Geometric Design **16(9)** (1999) 861–882
12. Krishnan, S., Manocha, D.: An efficient surface intersection algorithm based on lower-dimensional formulation. ACM Transactions on Graphics **16(1)** (1997) 74–106
13. Lanquetin, S., Foufou, S., Kheddouci, H., Neveu, M.: A graph based algorithm for intersection of subdivision surfaces. ICCSA 2003 **3** 387–396
14. Lin, M., Gottschalk, S.: Collision detection between geometric models: A survey. Proceedings of IMA Conference on Mathematics of Surfaces (1998)
15. Litke, N., Levin, A., Schröder, P.: Trimming for subdivision surfaces. Computer Aided Geometric Design **18(5)** (2001) 429–454
16. MPEG 4 Committee: MPEG 4 Standard. (2002)
17. Rabbitz, R.: Fast Collision Detection of Moving Convex Polyhedra. Graphics Gems IV (1994) 83–109
18. Rappoport, A., Spitz, S.: Interactive Boolean operations for conceptual design of 3-d solids. Proceedings of SIGGRAPH 1997 269–278
19. Schröder, P.: Subdivision as a fundamental building block of digital geometry processing algorithms. Journal of Computational and Applied Mathematics **149(1)** (2002) 207–219
20. Schweitzer, J. E.: Analysis and application of subdivision surfaces. PhD Thesis, University of Washington (1996)
21. Wu, X., Peters, J.: Interference detection for subdivision surfaces. Computer Graphics Forum **23(3)** (2004) 577–584
22. Zachmann, G.: Minimal hierarchical collision detection. Proceedings of the ACM symposium on Virtual reality software and technology (2002) 121–128