Computing Bounding Volume Hierarchies Using Model Simplification

Tiow-Seng Tan[†], Ket-Fah Chong[†]

National University of Singapore

Kok-Lim Low[‡]

University of North Carolina at Chapel Hill

Abstract

This paper presents a framework that uses the outputs of model simplification to guide the construction of bounding volume hierarchies for use in, for example, collision detection. Simplified models, besides their application to multiresolution rendering, can provide clues to the object's shape. These clues help in the partitioning of the object's model into components that may be more tightly bounded by simple bounding volumes. The framework naturally employs both the bottom-up and the top-down approaches of hierarchy building, and thus can have the advantages of both approaches. Experimental results show that our method built on top of the framework can indeed improve the bounding volume hierarchy, and as a result, significantly speedup the collision detection.

CR Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – surface and object representations.

Additional Keywords: collision detection, hierarchical data structure, intersection searching, space partitioning, ray tracing, level of detail, multiresolution rendering, vertex clustering.

1 INTRODUCTION

Fast rendering and collision detection are two fundamental goals of many interactive 3D graphics applications. Many realistic-looking 3D models, containing millions of polygons where almost all are visible in a complex scene, can hardly be rendered at interactive frame rates. A solution for this is to use multiresolution or level-of-detail modeling, in which each object has a series of geometric approximations or simplified models with increasingly lower rendering cost, and they resemble the original models from all directions. Also, navigation in a complex scene requires extensive detection of collisions between models and thus needs support from very efficient collision detection data structures and algorithms to achieve interactive frame rates.

Though there are significant advances in constructing good bounding volume hierarchies, there still remains many open issues, such as the choice between the top-down and the bottom-up approaches, and the choice of bounding volume [Klosowski98]. Our work differs from all previous work by looking beyond the above issues to focus on the importance of the objects' shapes for constructing good bounding volume hierarchies.

The new framework proposed in this paper uses the outputs of model simplification to guide the construction of bounding volume hierarchies. Simplified models, besides their application to multiresolution rendering, can provide clues to the object's shape. These clues help in the partitioning of the object's model into components that may be more tightly bounded by simple bounding volumes. The framework naturally employs both the bottom-up and the top-down approaches of hierarchy building, and thus can have the advantages of both approaches.

A method, based on the proposed framework, has been built to experiment on architectural, mechanical and CAD models found in applications such as simulation, modeling, and virtual prototyping. Motivated by the interactive 3D design and modeling environment where the method may be used, we have adopted approaches that have efficient processing time.

The rest of the paper is organized as follows. Section 2 reviews some previous work in simplification and collision detection. Section 3 presents the basic idea of the proposed framework. Section 4 describes the details of a method built on top of our framework. Section 5 discusses the roles of model simplification in the framework, section 6 presents our experimental results, and section 7 concludes the paper.

2 PREVIOUS WORK

Recently, there has been much research in polygonal simplification algorithms. This is demonstrated by the explosion of papers lately, see for example, [Cohen98, Garland97, Hoppe97, Luebke97, Popovic97]. Heckbert and Garland have given a good overview of the area [Heckbert97]. These algorithms form a spectrum, ranging from fast, simple methods that yield outputs with moderate quality to slower, more sophisticated methods with superb quality outputs.

Similarly, collision detection has been studied extensively in the literature. A comprehensive discussion has been given by Klosowski *et al.* [Klosowski98]. Many popular algorithms are based on either the hierarchy of bounding volumes that successively approximate the parts of the object's model until the exact geometry of the model is reached (e.g. [Barequet96,

[†] School of Computing, National University of Singapore, Lower Kent Ridge Road, Singapore 119260.
Email: {tants | chongket}@comp.nus.edu.sg.

[‡] Department of Computer Science, University of North Carolina at Chapel Hill (on study leave from the School of Computing, National University of Singapore). Email: lowk@cs.unc.edu.

1999 Symposium on Interactive 3D Graphics Atlanta GAUSA

Beckmann90, Hubbard96, Palmer95, Klosowski98, Gottschalk96]) or spatial decomposition of the space occupied by the model (e.g. [Garcia94, Held95, Moore88, Naylor90, Noborio89]). Both approaches are aiming to reduce the number of pairs of objects or primitives that need to be checked for contact.

With our proposed framework, we are able to improve the OBB-trees constructed by RAPID [Gottschalk96] (a leading publicly available software library for collision detection). These better bounding volume hierarchies, in turn, speedup the detection of collisions during runtime. Apart from collision detection purposes, bounding volume hierarchies are also used to accelerate ray-tracing [Arvo90, Klimaszewski97].

3 THE FRAMEWORK

In the proposed framework, the outputs of model simplification are used to guide the construction of a bounding volume hierarchy for the input model. We will also call a bounding volume hierarchy a *BV-tree*.

Here is the basic idea of the general framework. For each simplified model, which is usually drastically simplified from the input model, its primitives are grouped into *parts*. For each part, a *component* of the original model is formed by collecting all the original polygons that simplify into the primitives in the part. These disjoint components form a partition of the original model. Note that a component is a subset of the original model's polygons.

These first-level components can be further decomposed into sub-components in the same way by invoking the simplification process again with higher level of detail than the first simplification. There are mainly two ways of invoking the higher level of detail simplification. In the first, the simplification is separately applied to each first-level component, whereas in the second, the simplification is applied to the entire original model. For the second way, some sub-component may not be a subset of any first-level component, i.e. it contains polygons that come from more than one first-level component. This can be resolved by breaking the sub-component into smaller sets such that every one of them is a subset of some first-level component. Each sub-component thus formed is then linked to the first-level component that is a superset of the sub-component. The further decomposition can be recursively applied to the sub-components. The result of this recursive partitioning is a *component tree* of the original model. Figure 1(a) shows an example of a component tree.

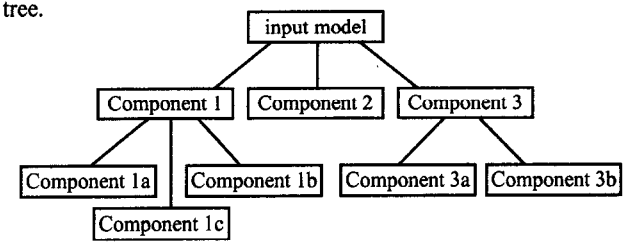


Figure 1(a): An example of a component tree.

We can view a component tree as the topmost levels of a bounding volume hierarchy. However, due to the possibly many number of components generated for each node in the component tree, the degree of the BV-tree can be high. Since it is generally agreed that high degree may be harmful to the performance of collision detection, we need to convert the original tree to a low-degree tree. For each node in the tree, we can combine its components in a bottom-up manner to reduce the tree's degree. Figure 1(b) shows a binary component tree converted from the original tree in Figure 1(a).

The topmost levels of the BV-tree (which we will refer to as the *upper BV-tree*) is then derived from the new component tree by computing a bounding volume for each node in the component tree. In practice, we do not perform the component decompositions down to the level where each leaf component is a single polygon. Therefore, in the next step of the framework, a traditional BV hierarchy building method is applied to each leaf component in the component tree. Each leaf component's BV-tree replaces the corresponding leaf in the upper BV-tree, and this completes the construction of the BV-tree for the input model. Figure 1(c) shows a BV-tree constructed from the example component tree in Figure 1(b).

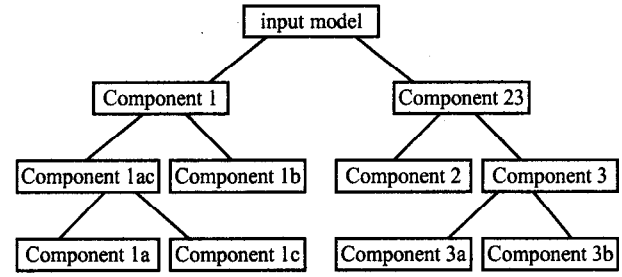


Figure 1(b): A binary component tree converted from the original tree in Figure 1(a).

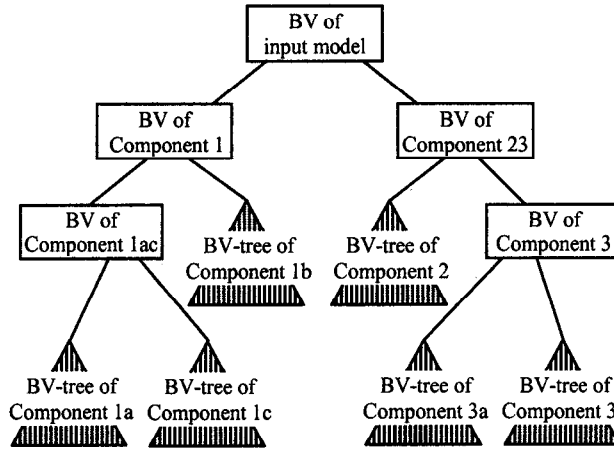


Figure 1(c): The final BV-tree constructed from the binary component tree in Figure 1(b).

The framework has a few potential advantages. The information we get from the simplified models can allow us to better partition the original model into components that may be more tightly bounded by a pre-specified type of simple bounding volumes. Since the number of components is usually much smaller than the number of polygons in the input model, we can afford to use a more exhaustive bottom-up algorithm to perform the conversion of the component tree to one with a low specified degree. The above two factors allow us to compute a BV-tree that is good in its topmost few levels.

As the lower part of the BV-tree can be built using a fast traditional top-down algorithm, the overall framework can be very time-efficient, provided the simplifications are fast too.

4 DETAILS OF THE METHOD

This section describes the details of our method that materialize the proposed framework. In the discussion, we assume that the input model is made up of triangles only.

Let T be the set of triangles of the input model M for which we need to construct a BV-tree B(T). Each node v of B(T) corresponds to a subset T_v of T, with the root node being associated with the full set T. Each internal node v of B(T) has two or more children whose associated subsets form a partition of T_v. The maximum number of children for any internal node of B(T) is called the *degree* of B(T). Associated with each node v of B(T) is a bounding volume that is an outer approximation to the set T_v using a smallest instance of some specified class of shapes, e.g. axis-aligned bounding boxes (AABBs) [Beckmann90, Bergen98], spheres [Hubbard96], oriented bounding boxes (OBBs) [Gottschalk96] and discrete orientation polytopes (*K*-dops) [Klosowski98].

4.1 Simplification of Input Model

One of the objectives of our method is to be time-efficient. The overall time efficiency is very much affected by the speed of the simplification process. For this, we have chosen the vertex-clustering simplification algorithm described in [Low97] to do the simplifications. The simplification algorithm, which we will refer to as the *floating-cell simplification* (FCS) algorithm, is similar to the original vertex-clustering method proposed by Rossignac and Borrel [Rossignac93], but produces better approximation quality, and thus, is able to give a better "sketch" of the input model than the latter.

The FCS algorithm outputs simplified models that may contain triangles, edges and points. From our observations, in many cases, the edges give the most important clues to the basic shape of the model. An edge represents an elongated part in the model, and once the elongated part is identified, a good way to bound it is to use a bounding volume that can be oriented to fit along the length of the elongated part. Oriented bounding boxes (OBBs) and oriented cylinders are examples of such suitable simple bounding volumes. For our purpose, we have chosen the OBBs because their use in collision detection has recently been improved greatly by Gottschalk *et al.* [Gottschalk96].

4.2 Computing Component Tree C(T)

Let T be the set of triangles in a polygonal model M. After simplification, each triangle of T is simplified to a point, an edge or remains as a (possibly different) triangle. Let S be the set of triangles, edges, and points of a simplified model m of M. Notice that all elements in S are open sets, i.e., each triangle does not contain the three edges on its boundary, and each edge does not contain the two points at its ends. Define $F_m : T \rightarrow S$ be the mapping such that $F_m(t) = s$ iff triangle $t \in T$ is simplified to an element $s \in S$.

We first partition elements of S into *parts*, and then divide the triangles in T into *components* (with respect to m) so that t_1, t_2 belong to a same component iff $F_m(t_1)$ and $F_m(t_2)$ are in a same part of S. Intuitively, a part of S contains elements that are maximally connected among themselves as discussed next.

Define S_3, S_2, S_1 respectively as the set of triangles, edges, and points in S. Let us first consider the partition of S_3 into parts. Let R be a relation on S_3 such that $a\ R\ b$ iff a can reach b by traversing the triangles in S_3 where two adjacent triangles in the traversal must share an edge. It is easy to check that R is reflexive, symmetric and transitive thus an equivalence relation and defines a unique partition on S_3 (step **A1**). Then, for the purpose of having small total bounding volume of B(T), the edges of S_2 (step **A2**) and the points of S_1 (step **A4**) that exist in some triangles of S_3 are included into the respective parts in S_3. Also, each stray edge of S_2 (i.e. an edge not in any triangle of S_3) with both its endpoints in a same part is included into the part (step **A3**), since the bounding volume of the component corresponding to the part is likely to include the edge. Next, we consider the partition of the remaining elements in S_2. Each remaining stray edge is taken as a separate part (step **B1**). Also, each point that is an endpoint of a single stray edge can be grouped into the same part as the edge (step **B2**). Finally, each remaining point in S_1, isolated or incident upon by elements of different parts, forms an individual part (step **C1**). The following pseudo-code summarizes the above description.

/ Initializations */*

S_3, S_2, S_1 are initialized, respectively, to the set of triangles, edges and points in S;

/ Form parts due to triangles */*

do while $S_3 \neq \emptyset$

  $P := \{ t \}$ where t is any triangle in S_3;

  $S_3 := S_3 - \{ t \}$;

(A1) **do while** $\exists\ t' \in S_3$ where t' shares a side with some $t \in P$

    $P := P \cup \{ t' \}$; $S_3 := S_3 - \{ t' \}$;

  endDo

(A2) let $S'_2 \subseteq S_2$ whose every edge is a side of some triangle in P;

(A3) let $S''_2 \subseteq S_2$ whose every edge is an isolated edge with both endpoints incident to some triangles in P;

(A4) let $S'_1 \subseteq S_1$ whose every point is incident to some triangle in P but not any other triangle in S;

  Form a new part with $P \cup S'_2 \cup S''_2 \cup S'_1$;

  $S_2 := S_2 - S'_2 - S''_2$; $S_1 := S_1 - S'_1$;

endDo

/ Form parts due to stray edges */*

do while $S_2 \neq \emptyset$

(B1) let e be an edge in S_2;

(B2) let $S''_1 \subseteq S_1$ whose every point is an endpoint of e but not any other edge or triangle in S;

  Form a new part with $\{ e \} \cup S''_1$;

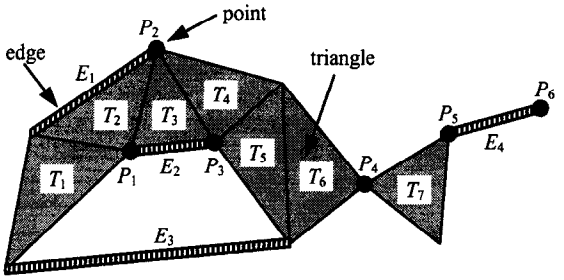
  $S_2 := S_2 - \{ e \}$; $S_1 := S_1 - S''_1$;

endDo

/ Form parts due to remaining points */*

(C1) Form a new part each with each remaining point in S_1;

Figure 2 shows the parts extracted from a simplified model by the above algorithm. Also, simple data structures are sufficient, in particular, on keeping information about the triangles incident to each vertex. When given this, the algorithm runs in time linear to $|S|$. The partition of S can be viewed as a *shape* of the input model. Shape, in general, is studied in various other areas such as computational geometry (see, for example, [Edelsbrunner94]) where input vertices are the basis to form shape. For our purpose, shape is defined from simplified model whose vertices are just approximation of the original. Furthermore, a shape is unlike, for example, the medial axis of an object [Sheehy96], which is expensive to compute and whose role is not clear for our purpose of object decomposition.



Parts formed due to triangles:

1. $\{ T_1, T_2, T_3, T_4, T_5, T_6, E_1, E_2, E_3, P_1, P_2, P_3 \}$
2. $\{ T_7, P_5 \}$

Parts formed due to stray edges:

3. $\{ E_4, P_6 \}$

Parts formed due to remaining points:

4. $\{ P_4 \}$

Figure 2: Computing parts from a simplified model.

A simple step after the application of the above algorithm in turn divides T into various components. The algorithm can be recursively applied, each time with a simplified model of higher level of detail, to obtain sub-components of components and so on. To divide a component containing triangles $T_c \subseteq T$ into sub-components using another simplified model m' of higher level of detail, we only need to consider elements $F_{m'}(T_c)$ in m'. Figure 4 (color plate) shows examples of components computed from some simplified models. In each picture, different colors are used to differentiate the different components and sub-components. Note that the edges in the simplified model are rendered using thicklines with variable thickness, as discussed in [Low97]. The components identified from T can naturally be arranged into a component tree C(T), in which each node represents a component, and its children, if any, represent the sub-components.

4.3 From C(T) TO B(T)

The component tree C(T) corresponds to the topmost levels of the whole bounding volume hierarchy of T, B(T). However, some nodes in C(T) may have many children and thus contribute to the high degree of the component tree. As we have already mentioned that high degree in a BV-tree may be ineffective for the purpose of collision detection, we next describe ways to convert C(T) into a binary tree before making it the topmost few levels of B(T).

We discuss the process with reference to the children v_i of each node v in C(T) for $i = 1, 2, \ldots, k$, where k is the number of children of v. Each child v_i has an associated bounding volume and a set of triangles $T_i \subseteq T$ in its represented component. We adopt the bottom-up approach to construct a binary tree with leaves v_i and root v. The method is similar to constructing a Huffman code tree. We first choose a pair, say v_i and v_j, to combine into v', which now contains the set of triangles $T_i \cup T_j$ and has an associated bounding volume. Then, we treat v' as a new child in place of v_i and v_j in the set of children to repeat the process until we are left with one child, which is the node v. It is natural to pick a pair where the resulting bounding volume of v' is minimum in the hope to minimize the total bounding volume of B(T). This can result in large components (generally with higher probabilities of collision) appearing higher in the B(T). Though such an approach requires searching of the smallest bounding volume, it remains efficient and effective as the number of children is normally small.

Let us discuss other possibilities. One other way is to use all triangles in the union of T_i (i = 1, 2, ..., k) to compute a partitioning halfplane by, for example, the RAPID system, and then assign each node v_i to the halfspace containing most of its triangles. This can be applied recursively to each halfspace until when there is only one or a few v_i in a subspace. The resulting tree obtained from the recursive partitioning will have nodes v_i as the leaves and node v as the root. Such top-down approach does not follow our principle of working with the components directly, and indeed does not show the best outcome in our experience.

Another possible top-down approach is as discussed in [Goldsmith87] for computing bounding volumes hierarchies for ray-tracing. This approach achieves efficient tree construction time of $O(n \log n)$, where n is the number of primitives in the input model, by working with estimated bounding area/volume at each stage of the computation. Though efficient, the method is sensitive to the order in which nodes are inserted into the tree. Our bottom-up approach seems more thorough in examining the possibilities and more direct in using the actual cost. Traditionally, bottom-up construction appears to be more difficult than top-down in that clusters of objects would have to be defined and grouped locally before they could be put together to form the bounding volume hierarchy. Such difficulty disappears in the proposed framework by the use of component hierarchy C(T). Notice that our method of computing the components for the topmost few levels of B(T) adheres to all the good properties (especially on the issue that BV-tree construction should concentrate on the nodes nearer the root of the tree) of BV-tree construction mentioned in [Kay86].

Once the topmost few levels of B(T) has been computed from C(T), the remaining computation to complete B(T) can be carried out next on each leaf of B(T) using traditional partitioning halfplane or spatial partitioning techniques. Our current implementation adopts the top-down partitioning method provided in the RAPID system [Gottschalk96].

With the complete B(T), checking for collisions can be performed as usual. In fact, in our implementation, the collision detection routine in RAPID is used, without modification, on the BV-trees created by our method.

5 ROLES OF SIMPLIFICATION

The success of our framework depends on the fact that simplification produces simplified models that show good sketches of the input model. Equally important, the simplified models should be consistent across different levels of detail, since these consistencies affect the goodness of the recursive partitioning of components into sub-components. From our experience, the floating-cell simplification (FCS) meets the above requirements, and moreover, it is very time-efficient. It is currently adapted for our implementation of the framework. Note that FCS requires an input parameter of the clustering cell width to determine the resolution of the simplified model. For many models, due to their locally dense but globally sparse nature, the frequency distribution of the lengths of the edges (sides of the triangles) indicates locations with drastic changes in frequencies. Figure 3 shows the histogram that depicts the frequency distribution of the lengths of the edges in the Enterprise model (shown in Figure 4). Such locations are good lengths for clustering cell widths. A straightforward algorithm has been designed to detect a few of such locations to generate simplified models.

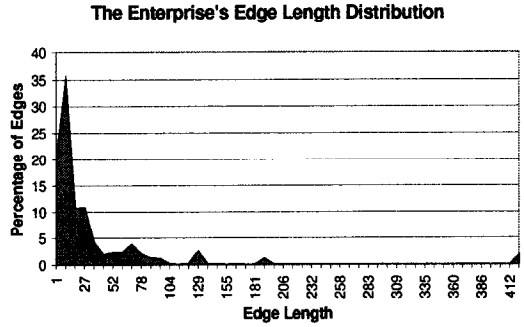


Figure 3: Edge length distribution.

In our implementation of the framework, we first compute a few (< 4) simplified models for the input model before the construction of B(T). One other possibility is to alternate between simplification and construction of a level of C(T) where simplification is performed to triangles of each component separately. The main advantage of the latter is the higher consistencies. However, it requires tight coupling between the simplification process and the construction of C(T). Nevertheless, our FCS has shown to produce consistent simplified models, and this latter approach is not necessary in our implementation.

As mentioned, the speed, the quality and the smoothness in transition are the main reasons for our choice of the FCS algorithm. Though there are other vertex and edge contraction algorithms that guarantee extremely high quality simplified models (and usually slower in computation time), this is, however, unnecessary for our purpose of constructing C(T). More importantly, the simplification algorithm must produce drastically simplified models that contain useful clues, such as stray edges and triangles sharing no edges with others, so that the original model can be decomposed into its components.

Note also that the use of simplification in the proposed framework is very different from that described by Faverjon [Faverjon89], in which the simplified model is directly used in the bounding volume hierarchy to verify collisions.

6 EXPERIMENTAL RESULTS

In the experiments to test the performance of our method built on the proposed framework, we have used the original, unmodified RAPID as the basis of our comparisons. In our method, we have integrated RAPID into the algorithm to construct the lower portion of the complete BV-trees. The BV-trees constructed by our method are then used by the RAPID collision detection routine to check for collision between objects. Here, we will refer to our collision system as *U-Collide*.

We ran our experiments on a SGI Indigo^2 workstation with a 250MHz MIPS R4400 CPU/R4000 FPU and 128 MB of main memory. Table 1 shows the preprocessing times to build the BV-trees for some test models. In our experiment, we used only 2 to 3 simplified models to extract the components from each model. We observe that the simplification and the computation of the components can be done quite fast. Our implementation of the preprocessing routines has yet to be optimized. For example, on the bottom-up processing to convert C(T) to the top few levels of B(T), our current implementation simply computes the bounding volumes repeatedly using the routines in the RAPID system—this can be avoided by progressively removing the triangles not useful in the bounding volume computation.

One of the demos found in the RAPID package was adapted to be the benchmark for measuring the collision query time. With that, we specified the number of simulation steps, the number of instances of a model, and the size of the environment. The number of simulation steps was set to 5000, the environment was populated with models so that the number of triangles is over 200000, and the size of the environment was set to simulate sparse, normal or dense situations. In a sparse environment, the sum of the topmost bounding volumes of all the instances is about 25% of the volume of the environment; a normal environment, about 50%; and a dense environment, about 100%. Table 2 shows the statistics of our experiments. The overall result is consistent over a few runs on each model. For all our examples, we obtain better collision detection performance in all the different environments. The percentages of improvement over the performance of RAPID are shown within parenthesis. Table 3 shows a comparison on the number of nodes in B(T) traversed during the simulation. In general, U-Collide performs better than RAPID.

7 CONCLUSION

Our proposed framework exploits the outputs of simplification in the construction of bounding volume hierarchies. In the framework, simplified models are used to extract the components and sub-components of the input model, so as to construct efficient topmost few levels of the bounding volume hierarchy. The framework explores our belief that the set of primitives in a simplified model shows a sketch of the object, thus revealing its shape.

Our experiments have shown that the framework can be used to improve existing bounding volume hierarchy building algorithms. In particular, the RAPID system can be improved to generate more efficient bounding volume hierarchies with little extra preprocessing overhead and without user intervention. Collision detection performance, as a result, improves significantly for a number of test cases.

Model	# of instances	# of triangles per instance	U-Collide						RAPID	
			# of level of detail (LOD)	# of components from each LOD	simplification plus component computation time (sec)	B(*T*) bottom-up processing time (sec)	B(*T*) top-down processing time (sec)	bounding volume per instance	B(*T*) processing time (sec)	bounding volume per instance
Windmill	100	3526	3	4, 4, 11	0.36	0.57	0.35	276	0.36	436
Helicopter	35	6448	3	7, 3, 8	0.57	0.83	0.77	573	0.72	1025
Chair	100	2481	3	8, 11, 0	0.24	0.65	0.27	7810	0.25	10399
Spacestation	20	10235	3	28, 41, 51	1.10	8.08	1.10	1941	1.14	2705
Rat	200	1180	3	3, 9, 7	0.12	0.25	0.13	139	0.11	117
Triceratops	40	5660	2	4, 10	0.40	0.42	0.66	649	0.61	601
Skateboard	10	22842	2	11, 8	2.10	4.91	2.94	1650	2.66	1809
Forklift	80	2529	3	4, 7, 3	0.26	0.14	0.28	1.48e+06	0.26	1.66e+6
F16	50	4428	3	4, 2, 2	0.45	0.24	0.49	352	0.46	348
Enterprise	150	1422	2	8, 0	0.98	0.18	0.15	7.27e+07	0.13	1.02e+8

Table 1: Preprocessing time.

Model	total number of collisions			time taken per simulation step (second)					
	Dense	Normal	Sparse	Dense		Normal		Sparse	
				RAPID	U-Collide	RAPID	U-Collide	RAPID	U-Collide
Windmill	572394	111548	16077	0.733000	0.335000 (54%)	0.113516	0.068434 (40%)	0.020152	0.012674 (37%)
Helicopter	107027	20388	3192	0.093428	0.063392 (32%)	0.019228	0.012998 (32%)	0.003700	0.002668 (28%)
Chair	510796	75046	10151	0.203400	0.160178 (21%)	0.032756	0.026536 (19%)	0.006324	0.006072 (4%)
Spacestation	70447	13692	2065	0.039104	0.027584 (30%)	0.008066	0.005604 (31%)	0.001528	0.001092 (29%)
Rat	1176528	233628	35116	0.627400	0.447760 (29%)	0.140146	0.099404 (29%)	0.033194	0.023194 (30%)
Triceratops	198962	40705	6775	0.096716	0.077646 (20%)	0.020610	0.017570 (15%)	0.004290	0.003668 (15%)
Skateboard	32697	11118	2558	0.013558	0.012842 (5%)	0.004908	0.004692 (4%)	0.001296	0.001254 (3%)
Forklift	565922	102249	14012	0.222600	0.220800 (1%)	0.043592	0.043034 (1%)	0.008172	0.007840 (4%)
F16	138417	28131	4511	0.077502	0.076230 (2%)	0.017474	0.016708 (4%)	0.003824	0.003588 (6%)
Enterprise	1677138	282050	38570	0.858909	0.543322 (37%)	0.137036	0.099162 (28%)	0.020814	0.018732 (10%)

Table 2: Comparison of the collision query performance.

Model	total number of B(*T*) nodes traversed					
	Dense		Normal		Sparse	
	RAPID	U-Collide	RAPID	U-Collide	RAPID	U-Collide
Windmill	330202542	213824501 (35%)	63894027	41451429 (35%)	9319980	5927499 (36%)
Helicopter	77569724	50280421 (35%)	15374727	9816672 (36%)	2484058	1572789 (37%)
Chair	144954471	112391953 (22%)	21555385	16794014 (22%)	2899858	2264126 (22%)
Spacestation	29767382	19985683 (33%)	5827253	3909421 (33%)	889271	581203 (35%)
Rat	397614854	316675111 (20%)	79774076	63546366 (20%)	11999895	9593104 (20%)
Triceratops	66255389	57398844 (13%)	13994163	12235287 (13%)	2341780	2045785 (13%)
Skateboard	10198879	9632627 (6%)	3555106	3427609 (4%)	833820	791443 (5%)
Forklift	164030935	163289124 (0.5%)	30329028	30193989 (0.4%)	4205738	4202565 (0.1%)
F16	57886338	57971941 (-0.1%)	11867579	11881557 (-0.1%)	1900567	1885667 (0.8%)
Enterprise	579426848	383526252 (34%)	99317494	65541615 (34%)	13191728	8922053 (32%)

Table 3: Comparison of the number of nodes traversed.

Acknowledgments

This work is supported by the National University of Singapore under grant RP960618. Also, the authors would like to thank Tong-Wing Woon for his helpful discussion and programming support to the project.

References

[Arvo90] J. Arvo and D. Kirk. *A Survey of Ray Tracing Acceleration Techniques.* In *An Introduction to Ray Tracing*, A.S. Glassner, ed., 201—262, Academic Press, 1990.

[Barequet96] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell and A. Tal. *BOXTREE: A Hierarchical Representation for Surfaces in 3D.* Proceedings Eurographics'96, Vol. 15(3), C-387—396, C-484, August 1996.

[Beckmann90] N. Beckmann, H. Kriegel, R. Schneider and B. Seeger. *The R*-tree: An Efficient and Robust Access Method for Points and Rectangles.* Proceedings ACM SIGMOD International Conference on Management of Data, 322—331, 1990.

[Bergen98] G. van den Bergen. *Efficient Collision Detection of Complex Deformable Models using AABB Trees.* To appear in *Journal of Graphics Tools* (http://www.win.tue.nl/cs/tt/gino/solid).

[Cohen98] J. Cohen, M. Olano and D. Manocha. *Appearance-Preserving Simplification.* Computer Graphics (SIGGRAPH'98 Proceedings), 115—122, July 1998.

[Edelsbrunner94] H. Edelsbrunner and E. Mücke. *Three-Dimensional Alpha Shapes.* ACM Transactions on Graphics, Vol. 13(1), 43—72, January 1994.

[Faverjon89] B. Faverjon. *Hierarchical Object Models for Efficient Anti-collision Algorithms.* Proceedings IEEE International Conference on Robotics and Automation, 333—340, 1989.

[Garcia94] A. Garcia-Alonso, N. Serrano and J. Flaquer. *Solving the Collision Detection Problem.* IEEE Computer Graphics & Applications, Vol. 14, 36—43, May 1994.

[Garland97] M. Garland and P. Heckbert. *Surface Simplification Using Quadric Error Metrics.* Computer Graphics (SIGGRAPH'97 Proceedings), 209—216, August 1997.

[Gottschalk96] S. Gottschalk, M.C. Lin and D. Manocha. *OBBTree: A Hierarchical Structure for Rapid Interference Detection.* Computer Graphics (SIGGRAPH'96 Proceedings), 171—179, 1996.

[Goldsmith87] J. Goldsmith and J. Salmon. *Automatic Creation of Object Hierarchies for Ray Tracing.* IEEE Computer Graphics & Applications, 14—20, 1987.

[Heckbert97] P. Heckbert and M. Garland. *Survey of Polygonal Surface Simplification Algorithms.* School of Computer Science, CMU, Pittsburgh, May 1997, http://www.cs.cmu.edu/~ph.

[Held95] M. Held, J. Klosowski and J. Mitchell. *Evaluation of Collision Detection Methods for Virtual Reality Fly-throughs.* Proceedings 7th Canad. Conf. Computational Geometry, 205—210, 1995.

[Hoppe97] H. Hoppe. *View-Dependent Refinement of Progressive Meshes.* Computer Graphics (SIGGRAPH'97 Proceedings), 189—198, August 1997.

[Hubbard96] P. Hubbard. *Approximating Polyhedra with Spheres for Time-critical Collision Detection.* ACM Trans. Computer Graphics, Vol. 15(3), 179—210, July 1996.

[Kay86] T. Kay and J. Kajiya. *Ray Tracing Complex Scenes.* Computer Graphics (SIGGRAPH'86 Proceedings), 269—278, 1986.

[Klimaszewski97] K. Klimaszewski and T. Sederberg. Faster Ray Tracing Using Adaptive Grids. IEEE Computer Graphics & Applications, Vol. 17(1), 42—51, 1997.

[Klosowski98] J. Klosowski, M. Held, J. Mitchell, H. Sowizral and K. Zikan. *Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs.* IEEE Transactions on Visualization and Computer Graphics, Vol. 4(1), 21—36, 1998.

[Low97] K-L. Low and T.S. Tan. *Model Simplification Using Vertex-Clustering.* Proceedings of 1997 Symposium on Interactive 3D Graphics, 75—81, April 1997.

[Luebke97] D. Luebke and C. Erikson. *View-Dependent Simplification of Arbitrary Polygonal Environments.* Computer Graphics (SIGGRAPH'97 Proceedings), 199—208, August 1997.

[Moore88] M. Moore and J. Wilhelms. *Collision Detection and Response for Computer Animation.* Computer Graphics (SIGGRAPH'88 Proceedings), 289—298, August 1988.

[Naylor90] B. Naylor, J. Amatodes and W. Thibault. *Merging BSP Trees Yields Polyhedral Set Operations.* Computer Graphics (SIGGRAPH'90 Proceedings), 115—124, August 1990.

[Noborio89] H. Noborio, S. Fukuda and S. Arimoto. *Fast Interference Check Method using Octree Representation.* Advanced Robotics, Vol. 3, 193—212, 1989.

[Palmer95] I. Palmer and R. Grimsdale. *Collision Detection for Animation using Sphere-trees.* Computer Graphics Forum, Vol. 14(2), 105—116, June 1995.

[Popovic97] J. Popovic and H. Hoppe. *Progressive Simplicial Complexes.* Computer Graphics (SIGGRAPH'97 Proceedings), 217—224, August 1997.

[Rossignac93] J. Rossignac and P. Borrel. *Multi-Resolution 3D Approximations for Rendering Complex Scenes.* In Modeling in Computer Graphics, B. Falcidieno and T. Kunii, Eds., Springer-Verlag, 1993, 455—465.

[Sheehy96] D. J. Sheehy, C. G. Armstrong and D. J. Robinson. Shape Description by Medial Surface Construction. IEEE Transactions on Visualization & Computer Graphics, Vol. 2(1), 62—72, March 1996.

Figure 4: Components identified from simplified models.