Eurographics Symposium on Geometry Processing (2004)
R. Scopigno, D. Zorin, (Editors)

Fast Collision Detection between Massive Models using Dynamic Simplification

Sung-Eui Yoon Brian Salomon Ming Lin Dinesh Manocha

University of North Carolina at Chapel Hill
{sungeui,salomon,lin,dm}@cs.unc.edu
http://gamma.cs.unc.edu/MRC

Abstract

We present a novel approach for collision detection between large models composed of tens of millions of polygons. Each model is represented as a clustered hierarchy of progressive meshes (CHPM). The CHPM is a dual hierarchy of the original model; it serves both as a multiresolution representation of the original model, as well as a bounding volume hierarchy. We use the cluster hierarchy of a CHPM to perform coarse-grained selective refinement and the progressive meshes for fine-grained local refinement. We present a novel conservative error metric to perform collision queries based on the multiresolution representation. We use this error metric to perform dynamic simplification for collision detection. Our approach is conservative in that it may overestimate the set of colliding regions, but never misses any collisions. Furthermore, we are able to generate these hierarchies and perform collision queries using out-of-core techniques on all triangulated models. We have applied our algorithm to perform conservative collision detection between massive CAD and scanned models, consisting of millions of triangles at interactive rates on a commodity PC.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Hierarchy and Geometric Transformations

1. Introduction

Recent advances in CAD and scanning technologies have resulted in geometric models of complex objects and structures consisting of millions of polygons. The availability of these models has stimulated research in model simplification, mesh compression, real-time rendering, and large-scale simulation. In this paper, we present a novel, fast algorithm for collision detection between complex, massive models composed of millions of geometric primitives. Collision queries frequently arise in various applications including virtual prototyping, dynamic simulation, interaction, navigation and motion planning.

Collision detection has been well-studied for more than three decades and some of the commonly used algorithms are based on spatial partitioning or bounding volume hierarchies (BVH). However, existing algorithms may not achieve interactive performance on large, complex models consisting of tens of millions of polygons. The memory requirements of these algorithms are typically very high, as precomputed BVHs can take many gigabytes of space. Moreover, the number of pairwise overlap tests between the bounding volumes can grow as a super-linear function of the model size, thereby slowing down the query performance.

In order to deal with the model complexity, algorithms using multiresolution representations or model simplification techniques have been proposed. These algorithms have been used to generate tight fitting BVHs [TCL99], to create static contact LODs [OL03], and to evaluate various factors affecting collision perception [OD01]. To the best of our knowledge, none of them have been applied to general, unstructured complex models composed of millions of triangles.

Main Contribution: We present a fast and conservative collision detection algorithm for massive models composed of millions of polygons. We use a novel model representation, a *clustered hierarchy of progressive mesh* (CHPM), which serves as a *dual hierarchy* of each model. We use this representation both as a bounding volume hierarchy to cull away cluster pairs that are not in close proximity and as a multires-

olution representation that adaptively computes a simplified representation of each model on the fly. Our algorithm utilizes the cluster hierarchy for coarse-grained refinement and progressive meshes (PMs) associated with each cluster for fine-grained local refinement. This allows us to rapidly compute a dynamic simplification and reduce the "popping" or discontinuities between successive collision queries associated with static levels-of-detail. We use GPU-based occlusion queries for fast collision culling between dynamically-generated simplifications of the original models.

We also introduce a new conservative collision error metric. Based on this error metric, we compute the mesh simplification and perform overlap tests between the bounding volumes and the primitives. Our overall algorithm is conservative and never misses any collisions between the original model, though it may return "false positive" collisions within an error bound. Moreover, we only load the cluster hierarchy in the main memory and use out-of-core techniques to fetch the progressive meshes at runtime. Our algorithm has been implemented on a commodity PC with an NVIDIA GeForce FX 5950 Ultra GPU and dual 2.5GHz Pentium IV processors and uses a memory footprint of approximately 250MB. It has been used for real-time dynamic simulation between two complex scanned models consisting of 1.7M and 28M triangles and interactive navigation in a CAD environment composed of more than 12 million triangles. Collision queries using our algorithm take about $15-40$ milliseconds to compute all the contact regions on these benchmarks. Some of the key benefits of our approach include:

- **Generality:** Our algorithm makes no assumption with respect to model complexity or topological structures. It can also handle "polygon soup" models.
- **Lower memory overhead:** In practice, the CHPM of a model takes $5-8$ times less memory as compared to a BVH. Moreover, our out-of-core algorithm uses a small runtime memory footprint.
- **Fast collision queries:** Our dynamic simplification algorithm bounds the size of the front in each hierarchy and computes all contacts between complex models in tens of milliseconds.
- **Error bounded and conservative:** Our algorithm is conservative in the sense that it detects all contacts. It may report "false positive" collisions within a user-specified error bound.
- **Integrated multiresolution representation:** The dynamic LOD reduces popping in simulation and the CHPM can also be used for interactive display of massive model [YSGM04]. Therefore, this new representation can be adopted for interactive display, real-time interaction, and physical simulation of massive models simultaneously.

Organization: The rest of the paper is organized in the following manner. Section 2 briefly surveys previous work. We present an overview of our approach and the model representation in Section 3. Section 4 describes the algorithm to compute the CHPM and the error metrics used for model simplification. We present our criteria to perform conservative and multiresolution collision queries in Section 5 and the overall collision detection algorithm in Section 6. We describe its implementation and performance in Section 7 and highlight some of the limitations in Section 8. Section 9 provides some concluding remarks and discusses future work.

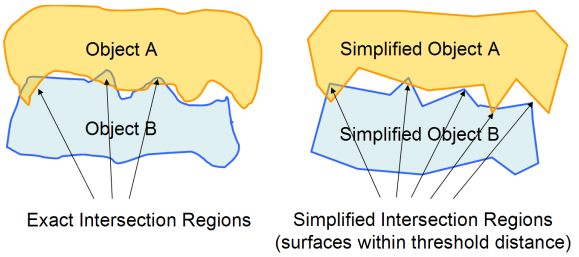

Figure 1: Collision Detection using Dynamic Simplification: *Collision detection between original objects is shown in left and collision between the corresponding simplified objects is shown on the right. All colliding regions between the original objects are detected by our algorithm and we compute a simplified representation of each colliding region. Moreover, "false positive" collisions are also reported within a given error threshold due to the conservativeness of our algorithm.*

2. Related Work

We give a brief overview on the related work in model simplification and collision detection.

2.1. Model Simplification

Simplification algorithms compute a reduced-polygon count approximation of a model, while attempting to preserve its shape. Most of the existing work in model simplification has been targeted towards rendering acceleration [LRC*02]. At a broad level, the simplification algorithms can be classified into static simplification algorithms or dynamic simplification algorithms.

The static approaches pre-compute a discrete series of levels-of-detail (LODs) in a view independent manner [CVM*96, GH97, EMB01]. At run time, the rendering application selects one of the static LODs based on the error threshold. As a result, the run-time overhead is relatively small. However, switching between different static LODs can result in "popping" artifacts or discontinuities in the simulation.

The view-dependent or dynamic algorithms pre-compute a hierarchical data structure that encodes a continuous range of detail. View-dependent simplification originated as an extension of the progressive mesh (PM) [Hop96]. A PM is a linear sequence of increasingly coarse meshes built from an input mesh by repeatedly applying edge collapse operations. Xia and Varshney [XESV97] and Hoppe [Hop97] organized the PM as a vertex hierarchy (or view-dependent progressive mesh (VDPM)) instead of a linear sequence.

Luebke and Erikson [LE97] developed a similar approach employing octree-based vertex clustering operations and used it for dynamic simplification. El-Sana and Varshney [ESV99] extended these ideas using a uniform error metric based on cubic interpolants and reduced the cost of runtime tests. Other view-dependent representation include the Multi-Triangulation [LDF97]. Pajarola [Paj01] improved the update rate of runtime mesh selection by exploiting properties of the half-edge mesh representation and applied it to manifold objects. El-Sana and Bachmat [ESB02] presented a mesh refinement prioritization scheme to improve the runtime performance. Yoon et al. [YSGM04] performed out-of-core simplification on large meshes consisting of a few hundred million triangles and presented a two-level view-dependent rendering algorithm.

2.2. Collision Detection

The problem of collision detection has been well-studied in the literature. See recent surveys in [JTT01, LM03]. Most of the commonly used techniques to accelerate collision detection between two objects utilize spatial data structures, including bounding volume and spatial partitioning hierarchies. Some of the commonly used bounding volume hierarchies (BVHs) include sphere-trees [Hub93], AABB-trees [BKSS90], OBB-trees [GLM96], k-DOP-trees [KHM*98], etc. These representations are used to cull away portions of each object that are not in close proximity. Recently, GPU-based accelerated techniques have also been proposed for fast collision detection [KP03, HTG03, GRLM03, KOLM02]. Their accuracy is governed by the frame-buffer or image-space resolution. Recently, Govindaraju et al. [GLM04] have presented a reliable GPU-based collision culling algorithm that overcomes these precision problems due to limited frame-buffer resolution.

Massive Models: There is relatively less work on collision detection between complex models composed of millions of polygons. The BVH based algorithms can be directly applied to these models. However, the memory overhead for the resulting algorithms can be substantial (e.g. many gigabytes). Wilson et al. [WLML99] presented an out-of-core collision detection algorithm for large environments composed of multiple objects. Their algorithm uses spatial proximity relationships between different objects for out-of-core data management. Niubo and Brunet [FNB03] have presented a K-dimensional data structure for broad-phase collision and proximity detection in large environments requiring external memory storage.

Approximate Collision Detection: In order to achieve interactive performance in complex algorithms, many approximate algorithms have been proposed. Hubbard [Hub93] introduced the concept of time-critical collision detection using sphere-trees. Collision queries can be performed as far down the sphere-trees as time permits, without traversing

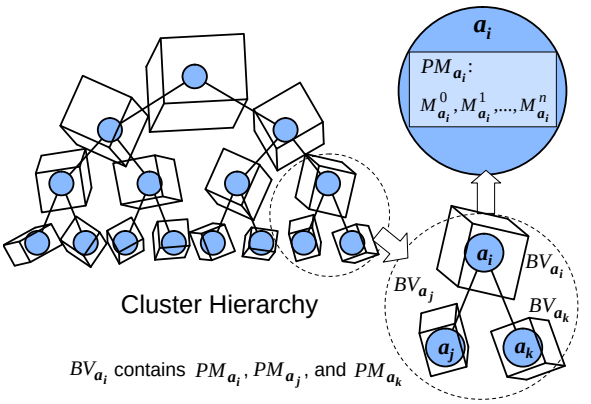

Figure 2: CHPM Hierarchy. *We represented the scene as a clustered hierarchy of progressive meshes (CHPM). The CHPM serves as a dual hierarchy: an LOD hierarchy for conservative error-bounded collision and as a bounding volume hierarchy for collision culling. Each cluster contains a progressive mesh and a bounding volume that encloses all geometry in its subtree.*

the entire hierarchy. This concept can be applied to any type of bounding volume hierarchy (BVH). However, no tight error bounds have been provided using this approach. O'Sullivan and Dingliana [OD01] studied LOD techniques for collision simulations and investigated different factors affecting collision perception, including eccentricity, separation, causality, and accuracy of simulation results. Otaduy and Lin [OL03] proposed CLODs, which are precomputed dual hierarchies of static LODs used for multiresolution collision detection. The runtime overhead of this approach is relatively small. However, switching LODs between successive instances may result in a large discontinuity in the simulation. Moreover, the underlying approach assumes that the input model is a closed, manifold solid and is not directly applicable to polygon soups.

3. Model Representation

In this section we introduce some of the terminology and representations used by our algorithm. We also give a brief overview of our approach for out-of-core hierarchical collision detection.

3.1. Dynamic Simplification of Massive Models

Our goal is to use dynamic or view-dependent simplification algorithms for fast collision detection between massive models. Most of the prior work on dynamic simplification has been motivated by view-dependent rendering and uses vertex hierarchies such as VDPMs [Hop96, LE97, XESV97]. These approaches augment each edge collapse with *dependency* information related to the local neighborhood at the time of the edge collapse during construction. This information is used to prevent "fold-overs" whereby a face normal is reversed at runtime. However, traversing and refining

an *active vertex front* across a vertex hierarchy composed of tens of millions of polygons can take hundreds of milliseconds per timestep. Resolving the dependencies can lead to non-localized memory accesses which can be problematic for out-of-core collision detection and rendering. Furthermore, these vertex hierarchies can be too fine-grained to serve as effective bounding volume hierarchies for collision detection.

3.2. CHPM Representation

We use a novel representation, a clustered hierarchy of progressive meshes (CHPMs) [YSGM04], for fast collision computation using dynamic LODs of massive datasets. The CHPM representation serves as a dual hierarchy for collision detection: as an LOD hierarchy for error-bounded collision detection and as a bounding volume hierarchy for collision culling. The CHPM consists of two parts (as shown in Fig. 2):

Cluster Hierarchy: We represent the entire dataset as a hierarchy of clusters, which are spatially localized mesh regions. As an LOD hierarchy each interior cluster contains a coarser representation of its children's meshes. As a bounding volume hierarchy (BVH) each cluster has an associated bounding volume (BV) which contains all the mesh primitives represented by its subtree. We use the oriented bounding box (OBB) as the BV representation.

Progressive Mesh: Each cluster contains a progressive mesh (PM) as an LOD representation. The PM representation is a linear sequence of LOD meshes that constructed using the edge-collapse simplification operation [Hop96]. The sequence is stored as a base mesh (M^0) and a list of vertex-splits. Each vertex split reintroduces the vertices and faces removed during simplification by an edge collapse. By applying the vertex splits at runtime, any mesh in the sequence $M^0, M^1, \ldots, M^n$ can be selected.

A CHPM representation is refined by first selecting a front in the cluster hierarchy. This coarse level refinement chooses clusters to represent the model with bounded errors in each mesh region. The PM contained in each cluster allows fine-grained control and local mesh refinement. The CHPM representation is a middle ground between the flexibility of a vertex hierarchy and the refinement speed of a static LOD (or hierarchical LOD) representation [EMB01].

To detect collisions between a pair of CHPM objects we perform cluster level culling between their cluster hierarchies. Once a set of colliding clusters is computed, PM refinement is performed on and exact collisions between the PM representation are computed. The PMs are used as a continuous LOD representation to alleviate simulation popping artifacts and satisfy the collision error bounds.

Figure 3: Cluster Decomposition. *This figures highlights the clusters on the Lucy model (28M triangles). The average cluster size is 1K triangles. Each cluster is represented by a progressive mesh for dynamic simplification and contains a bounding volume for collision culling.*

3.3. Dual Hierarchies

By combining an LOD hierarchy with a traditional BVH we are able to achieve a dramatic acceleration of collision detection between massive models. The CHPM hierarchy allows collisions to be computed using a dynamically generated approximate mesh and thereby reducing the number of overlap tests that need to be performed. Because we use a continuous LOD representation, LOD transitions are smooth and can meet an error bound without being overly conservative.

The collision test between two BVHs can be described by the *bounding volume test tree* (BVTT)[LGLM00], a tree structure that holds in each node the result of the query between two BVs. The overall cost of a collision test is proportional to the number of nodes in the front of the BVTT. The basic BVTT algorithm traverses down to the leaves of the BVHs, as long as each query reports a possible collision. However, when traversing the combined cluster hierarchy within the CHPM, the traversal stops when an appropriate LOD is reached. Therefore, the BVTT front size can be dramatically reduced by using LODs and thereby making it possible to perform collision queries between complex models at interactive rates.

4. Simplification and Error Values

An important issue in both mesh simplification for rendering and LOD-based collision detection is the choice of error metrics and their computation. In this section we briefly dis-

cuss the CHPM computation algorithm and the error metrics used for conservative error-bounded collision detection.

4.1. CHPM Computation

CHPMs for complex models are computed using an out-of-core clustering and simplification algorithm [YSGM04]. First, given an input mesh we compute a cluster decomposition. This is done in several passes over the faces and vertices to avoid loading the entire mesh in main memory at once. Then, a top down partition of the cluster decomposition creates a cluster hierarchy. Finally, a bottom up simplification process generates PMs for each cluster. An example of the cluster decomposition is shown in Fig. 3. More details are given in [YSGM04].

We also use the cluster hierarchy as a BVH and compute an OBB that encloses all the cluster triangles. Moreover, we ensure that the OBB not only encloses the triangles contained in that cluster, but also its descendant clusters. To guarantee this property each BV is computed as follows: after constructing a PM for the cluster, we use the covariance matrix algorithm [GLM96] to compute an OBB that contains all the vertices of the PM. To ensure that all the vertices of the descendant clusters are also contained, each dimension of the OBB is expanded by the maximum surface deviation between the base mesh of the PM and the original mesh.

4.2. Conservative Error Metric

Our collision detection algorithm dynamically computes a simplification of each model and checks for collisions between the simplified models. The accuracy of the algorithm is governed by the error function used to compute the simplification. An example of collision detection between simplified objects is shown in Fig. 1.

Given two original models, A^0 and B^0, and a minimum separation distance δ, a collision detection algorithm evaluates a function $Collide(A^0, B^0, \delta)$ that computes a set of triangle pairs (t_{A^0}, t_{B^0}) such that $t_{A^0} \in A^0$, $t_{B^0} \in B^0$, and $dist(t_{A^0}, t_{B^0}) < \delta$. For conservative LOD-based collision detection we modify this query. Instead, given the CHPM representations, A and B, we compute:

LodCollide$(A, B, \delta, \varepsilon)$: Determines all pairs (t_A, t_B) such that $t_A \in A$, $t_B \in B$, and $dist(t_A, t_B) < \delta$ with allowed error ε, or $dist(t_A, t_B) < (\delta + \varepsilon)$. The dynamic simplification used for LOD-based collision detection is determined by the user-specified error ε.

Note that this query is defined so that we compute all the triangle pairs within distance $(\delta + \varepsilon)$. Thus, our algorithm is a conservative algorithm which will not miss any collisions. We also use another proximity query in our algorithm:

ConservBVTest$(BV_i, BV_j, \delta, \varepsilon)$: Given two bounding volumes, BV_i and BV_j, this query conservatively determines whether the subset of the original model contained in these BVs are colliding (Sec. 5).

Notation	Meaning
$\boldsymbol{a}$	A cluster of object A
$PM_{\boldsymbol{a}} = (M^0_{\boldsymbol{a}}, M^1_{\boldsymbol{a}}, ..., M^n_{\boldsymbol{a}})$	The PM of cluster $\boldsymbol{a}$
$h(M^i_{\boldsymbol{a}})$	The directed Hausdorff distance between $M^i_{\boldsymbol{a}}$ and the original mesh
$\hat{h}(BV)$	The directed Hausdorff distance between a bounding volume, BV, and the original mesh
δ	The minimum separation distance for the global collision query. Triangles separated by less than this distance are in collision.
ε	The simplification error used for collision detection, specified as a directed Hausdorff distance
$dilate(BV, r)$	An operation that dilates a BV by distance r

Table 1: ***Notation.*** *This table highlights the notation used in the rest of the paper.*

Many error metrics have been proposed for approximate collision detection, including object size, object velocity, and constant frame-rate for time-critical collision detection [Hub93, OL03, OD01]. Our simplification algorithm is based on the maximum deviation error or the Hausdorff distance between the original mesh and the simplified mesh, M, denoted $h(M)$. By assuring that the total Hausdorff distance in regions of collision is less than the error threshold, ε, we can bound the simulation error. Other collision error metrics based on object size and velocity can be derived from the maximum deviation error [OL03]. In order to perform collision culling between cluster pairs at the cluster level using the CHPM representation, we also store the directed Hausdorff distance between each BV and the original mesh, $\hat{h}(BV)$.

A feature of the Hausdorff metric is that it adapts to the mesh in a contact-dependent manner. The contact forces computed will be more sensitive to simplification in areas with sharp features. However, simplification will be more restricted in such areas because of high deviation in the Hausdorff metric. In relatively flat regions, where the contact forces will be least affected by the simplification, the Hausdorff metric allows greater simplification [OL03].

5. Conservative Collision Formulation

In this section we present our conservative collision scheme which is used to guarantee that a query result using the CHPM representation does not miss any collision as com-

pared to an exact test on the original meshes within the distance error bound, ε. In Table 1, we highlight the notation used in the rest of the paper.

In performing LOD-based collision detection we take advantage of the fact that CHPM represents a dual hierarchy. $LodCollide()$ can be computed by performing a BVTT traversal between the BVHs of A and B, but a test is needed to check whether the original mesh regions represented by clusters $\boldsymbol{a}$ and $\boldsymbol{b}$ are within distance $\delta+\varepsilon$.

The $ConservBVTest()$ query relies on a dilated BV test that is applied to cluster BVs during BVTT traversal and performs overlap tests between the triangles of the PM.

5.1. Conservative Collision Metric

We transform the problem of checking whether the original meshes contained inside two BVs are within distance δ into an intersection test between the dilated BVs. Initially, consider the dilated OBB, $dilate(BV,d)$, to be defined as the Minkowski sum of BV with a sphere of radius d and represented as $BV \oplus d$. We use the following lemmas to check whether the original meshes contained inside two bounding volumes, BV_i and BV_j, are within distance $\delta+\varepsilon$.

Lemma 1: *If the dilated BVs, $dilate(BV_i,\delta/2)$ and $dilate(BV_j,\delta/2)$, do not intersect, the distance between the original meshes contained in the two BVs is greater than δ.*

Proof: Because each BV fully contains a portion of the original mesh, the minimum distance between the two meshes contained in the BVs is at least the sum of dilation amounts, δ.

Lemma 2: *If there is an intersection between dilated BVs $dilate(BV_i,\delta/2)$ and $dilate(BV_j,\delta/2)$ the distance between the original meshes contained in the BVs has an upper bound of $\delta+\hat{h}(BV_i)+\hat{h}(BV_j)$.*

Proof: Due to the conservativeness of the BVs, the BVs may intersect even though the meshes may not be colliding. By definition of directed Hausdorff distance, every point of each original BV is within distance $\hat{h}(BV)$ of the original mesh. Furthermore, the dilated BVs are within distance $\delta/2$ of the original BV. Therefore, the maximum total distance between the original meshes is $\delta/2+\hat{h}(BV_i)+\delta/2+\hat{h}(BV_j) = \delta+\hat{h}(BV_i)+\hat{h}(BV_j)$.

These Lemmas lead directly to the definition of $ConservBVTest()$:

$$ConservBVTest(BV_i,BV_j,\delta,\varepsilon) = \begin{cases} NoCollision, & \neg isect(dilate(BV_i),dilate(BV_j)) \\ Collision, & isect(dilate(BV_i),dilate(BV_j)) \\ & \text{and } \hat{h}(BV_i)+\hat{h}(BV_j) \le \varepsilon \\ PotentialCollision, & isect(dilate(BV_i),dilate(BV_j)) \\ & \text{and } \hat{h}(BV_i)+\hat{h}(BV_j) > \varepsilon \end{cases}$$

where *isect* is a bounding volume intersection test and the shorthand $dilate(BV)$ simply indicates $dilate(BV,\delta/2)$.

If the dilated boxes do not intersect then we know that the original meshes are not colliding by Lemma 1. However, if these boxes overlap we use the Hausdorff distances $\hat{h}(BV_i)$ and $\hat{h}(BV_j)$ to determine whether we can conclude that the original models are colliding. When $\hat{h}(BV_i)+\hat{h}(BV_j) \le \varepsilon$ then by Lemma 2 we can conclude that the distance between the original meshes must be within $\delta+\varepsilon$.

Rather than computing the exact Minkowski sum, we instead compute $dilate(BV,d)$ as an approximation of $BV \oplus d$ by extending each dimension of the OBB by $d/2$ from the center of the OBB. To satisfy Lemma 2, the $\hat{h}$ value associated with BV is extended by the maximum deviation between $dilate(BV,d)$ and $BV \oplus d$.

5.2. Cull and Refine Operations

To compute $LodCollide()$ we first refine the mesh for each object such that the sum of mesh deviations of each model is less than ε in regions of collision. Next, we check whether the selected LOD representations are within distance δ. Both parts of this computation use the $ConservBVTest()$ query through two operations:

- **Culling operation:** BV pairs whose distance is greater than δ are culled. To conservatively perform this culling step, we apply the $ConservBVTest()$ test by dilating the BVs of the two approximate mesh portions and checking for intersection between the dilated BVs. BVs for which $ConservBVTest()$ finds no collisions cannot be intersecting and are culled away.
- **Refining operation:** $ConservBVTest()$ can determine when the LOD resolution must be increased. The BV pairs, for which the $ConservBVTest()$ query reports a collision, has total simplification error less than ε and the triangles within the BVs are in collision. On the other hand, when $ConservBVTest()$ reports a potential collision the total Hausdorff distance is too high and further refinement needs to be performed on one of the BVs. We guarantee that refinement always decreases the $\hat{h}$ values. Once the total Hausdorff distance is below ε, $ConservBVTest()$ becomes an exact collision test.

By recursively performing these two operations, we can compute the triangle pairs from dynamic LODs whose distance is less than δ. More importantly, their counterparts in the original meshes are separated by less than $\delta+\varepsilon$.

6. Fast Collision Detection

In this section, we present a hierarchical collision detection algorithm based on the CHPM. We also present several culling techniques to improve its performance.

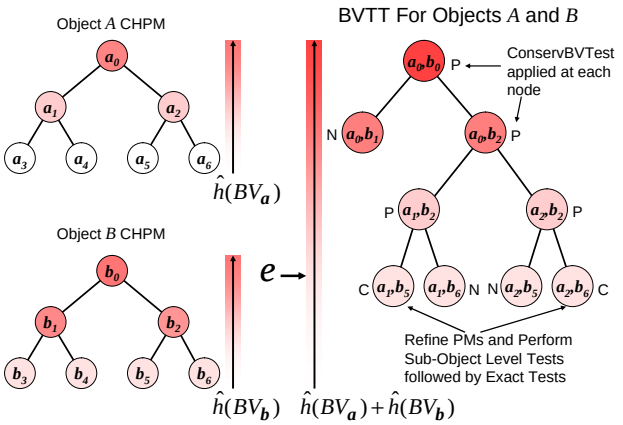

Figure 4: BVTT. *Each node of the bounding volume test tree (BVTT) represents a test between a cluster from each of two colliding objects. The test, ConservBVTest(), uses the clusters' bounding volumes to determine whether the cluster pair is not colliding (N), colliding (C), potentially colliding (P). The distinction between colliding and potentially colliding depends upon the sum of the clusters' associated errors (indicated by error bars) being below the error-bound,* ε.

6.1. Overall Algorithm

The overall algorithm for collision detection between two CHPM objects is shown in Alg. 1. We compute the colliding front of the bounding volume test tree (BVTT) using the culling and refining operations presented in Sec. 5.2. The colliding front contains pairs of clusters from the two objects that are in collision. For each of these cluster pairs, we perform an exact collision test after refining their PMs. This provides the fine-grained control of the simulation error. The cluster collision test uses a further culling algorithm that relies on GPU occlusion queries. Exact collision tests are performed after this additional culling step.

Algorithm 1 Compute collisions between two objects (*LodCollide*())
Input: *A*, *B*: Objects' δ: min. separation distance; ε: LOD error bound
Output: triangles of *A* and *B* in collision

```
ComputeLodCollide(A,B,δ,ε)
  tris ← ∅
  Front ←ComputeBVTTFront(A,B,δ,ε)
  for all (a,b) ∈ Front do
    tris ← tris∪ClusterCollide(a,b,δ,ε)
  end for
  return tris
```

6.2. Bounding Volume Test Tree (BVTT)

We use the concept of the bounding volume test tree (BVTT)[LGLM00] to accelerate the computation of *LodCollide*(). In the CHPM representation, the cluster hierarchy is also a BVH. We traverse the BVHs of both the objects and compute the BVTT.

A node $(\boldsymbol{a},\boldsymbol{b})$ in the BVTT represents a test between clusters $\boldsymbol{a}$ and $\boldsymbol{b}$ from objects *A* and *B*, respectively. If the test determines that the objects are non-colliding then the node is a leaf of the BVTT and no further tests are needed between the subtrees of *A* and *B* rooted at $\boldsymbol{a}$ and $\boldsymbol{b}$. Otherwise, there is a potential collision between $\boldsymbol{a}$ and $\boldsymbol{b}$. If the total Hausdorff error of $\boldsymbol{a}$ and $\boldsymbol{b}$, $(\hat{h}(BV_{\boldsymbol{a}})+\hat{h}(BV_{\boldsymbol{b}}))$, is less than ε, an exact test is performed to determine the triangles in collision; otherwise the cluster with greater error is refined (see Fig. 4). As shown in Alg. 2, we use the *ConservBVTest*() query to traverse the hierarchies of *A* and *B*, which implicitly computes the BVTT. The BVTT traversal effectively performs coarse-grained LOD refinement by selecting the clusters from objects *A* and *B* used for exact collision detection.

Algorithm 2 Perform BVTT traversal and compute the colliding BVTT front
Input: *A*, *B*: Objects, δ: min. separation distance, ε: LOD error bound
Output: triangles of *A* and *B* in collision

```
ComputeBVTTFront(A,B,δ,ε)
  return BVTest(Root(A),Root(B),δ, ε)

BVTest(a,b,δ,ε)
  t ← ConservBVTest(BV_a, BV_b, δ, ε)
  if t = NoCollision then
    {Culling: contained original meshes are further than δ}
    return ∅
  else if t = Collision then
    {Bounding boxes in collision, total error is less than ε}
    {These nodes are part of the colliding front}
    return (a,b)
  else {t = PotentialCollision}
    {Refining: total error is greater than ε}
    if ĥ(BV_a) > ĥ(BV_b) then
      return BVTest(LeftChild(a), b, δ, ε)
      ∪ BVTest(RightChild(a)), b, δ, ε)
    else
      return BVTest(a, LeftChild(b), δ, ε)
      ∪ BVTest(a, RightChild(b), δ, ε)
    end if
  end if
```

6.2.1. CHPM Front Computation

The BVTT front computed in the algorithm described above may contain multiple clusters representing the same portion of either *A* or *B*. This situation occurs when the traversal reaches BVTT nodes such as $(\boldsymbol{a_1},\boldsymbol{b_1})$ and $(\boldsymbol{a_1},\boldsymbol{b_2})$. It may be the case that $\hat{h}(BV_{\boldsymbol{a_1}})+\hat{h}(BV_{\boldsymbol{b_1}})>\varepsilon$ but $\hat{h}(BV_{\boldsymbol{a_1}})+\hat{h}(BV_{\boldsymbol{b_2}})\leq\varepsilon$. The traversal will split $\boldsymbol{a_1}$ into $\boldsymbol{a_2}$ and $\boldsymbol{a_3}$ in one branch of the BVTT but $\boldsymbol{a_1}$ will fall on the BVTT front in the other branch. We would like to have a single unique front

across each CHPM. In order to maintain this property the BVTT node $(\boldsymbol{a_1}, \boldsymbol{b_2})$ is forced to split into nodes $(\boldsymbol{a_2}, \boldsymbol{b_2})$ and $(\boldsymbol{a_3}, \boldsymbol{b_2})$.

6.2.2. Coherence-Based BVTT Front Computation

A further modification of the algorithm described above is made to take advantage of temporal coherence. Rather than recursively computing the BVTT front from the root for each timestep, we traverse the front from the previous timestep and make incremental updates. By collapsing the BVTT nodes into their parent node the level of refinement is reduced, and by splitting a BVTT node the level of refinement is increased. For massive models with deep LOD hierarchies, this approach leads to a substantial reduction of the time spent on BVTT computation.

Algorithm 3 Compute collision between two clusters
Input: $\boldsymbol{a}$, $\boldsymbol{b}$: clusters, δ: min. separation distance, ε: LOD error bound
Output: triangles of A and B in collision

```
ClusterCollide(a,b,δ,ε)
  RefinePMs(PM_a,PM_b,ε)
  T ← SubObjectCull(a,b,δ) {T is a set of triangle pairs}
  return ExactTest(T,δ)
```

6.3. Computing Dynamic LODs

We process each pair of clusters, $(\boldsymbol{a}, \boldsymbol{b})$, on the colliding front of the BVTT for exact collision detection. As shown in Alg. 3, the first step is to refine the PMs of the clusters. Each cluster pair must have a total deviation from the original meshes of not more than ε. In order to take advantage of temporal coherence, we refine the PMs based on their current state. If the sum of the errors is greater than ε, we apply vertex-splits to the PM with greater error until the error falls below ε. If the sum of errors is less than ε, we apply edge-collapses to the PM with lower error until applying one more edge-collapse would cause the total error to exceed ε. Once the PMs are refined, the total simplification error at each point of contact between the clusters will be less than ε. Since a single cluster may be in multiple cluster pairs of the BVTT front we ensure that the PMs are refined to meet the error bound in each BVTT front node.

6.4. GPU-based Culling

Performing all $O(n^2)$ pairwise tests between triangles of two clusters can be an expensive operation as the clusters may contain around 1K triangles. To further reduce the potentially colliding set of triangles, we employ GPU-based culling similar to [GRLM03, GLM04]. Triangles in the mesh selected from each cluster's PM are randomly partitioned into "sub-objects" of size k triangles. For each triangle of a sub-object we construct a BV dilated by $\delta/2$. Since these BVs must be constructed quickly at runtime, we use axis aligned bounding boxes.

We use GPU-based occlusion queries to cull the sub-objects between the two clusters. After rendering some geometric primitives, an occlusion query returns the number of pixels that pass the depth buffer test. We use these queries to perform a 2.5D overlap test between bounding volumes along the three orthogonal axes. First, the BVs for all the triangles of the first cluster are rendered under an orthographic projection. Then, the BVs for sub-objects from the second cluster are rendered with the depth test set to GL_GEQUAL. Sub-objects of the second cluster that have no pixels pass this reversed depth test are classified as non-intersecting with the BVs of all objects of the first cluster. These sub-objects may be culled from the set of possible collisions. The test is performed for projections along the x, y, and z axes. The same test is performed with the order of the clusters switched to cull sub-objects of the first cluster.

In order to ensure that errors are not introduced due to sampling in the frame buffer, we use a conservative algorithm to perform GPU-based culling [GLM04]. The BVs are expanded by taking their Minkowski sum with a sphere to ensure that they are rasterized into every pixel which they may partially cover.

6.5. Triangle Collision Test

We perform exact collision detection for triangles pairs that pass sub-object culling. Each triangle in the LOD representation of an object represents a set of triangles of the original model. In order to conservatively meet the error bound, an OBB is constructed for each triangle that contains the triangle plus the original mesh triangles that were simplified into it. To enclose the original geometric primitives, the OBB is initially a flat box aligned with the plane of the triangle containing its vertices. It is then dilated by the $\hat{h}$ value of its cluster. The OBBs are then further dilated by $\delta/2$ before being tested for intersection. Triangles whose enclosing OBBs are overlapping are reported as colliding.

6.6. Out-of-Core Computation

Our goal is to perform collision detection between models that cannot be stored in main memory. The CHPM representation also serves as a mechanism for out-of-core management. At runtime we keep the CHPM hierarchy for each object in the main memory, while the PMs for each cluster reside on the disk. A working set of PMs is kept in memory for collision detection. For each pair of colliding objects, we keep PMs for nodes on the BVTT front in main memory as well as their parents and children to handle LOD switches.

6.7. Unified Multiresolution Representation

One advantage of our approach is that the dynamic LOD representation used for collision detection can also be used for

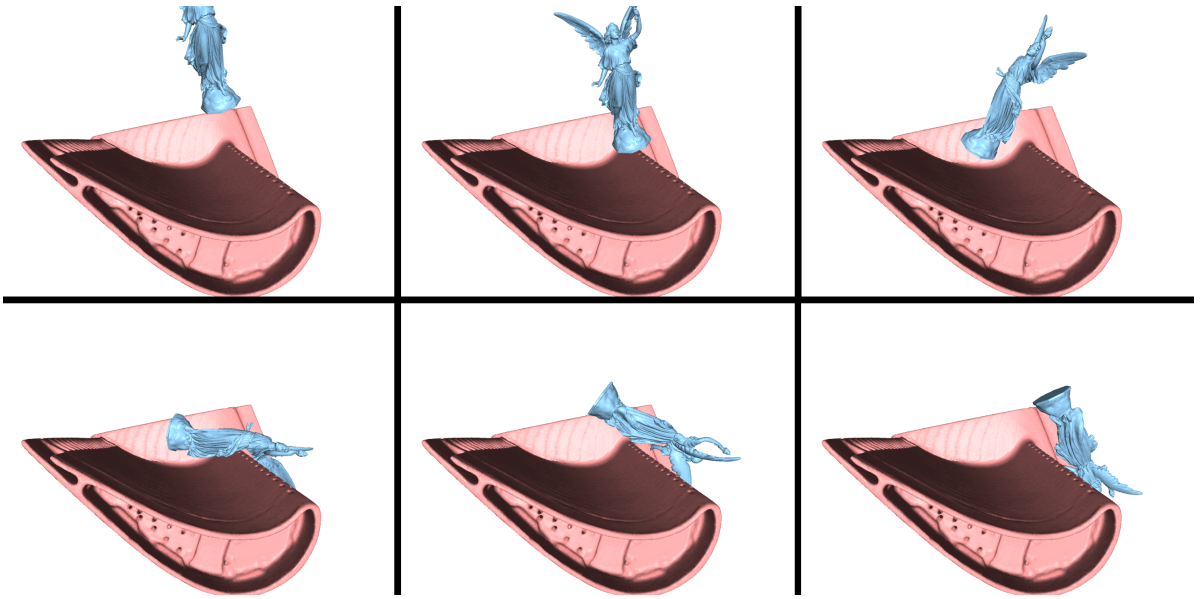

Figure 5: Collision Example. *This image sequence shows discrete positions from our dynamic simulation application. The* 28*M-triangle Lucy model falls on and bounces off the* 1.7*M-triangle turbine-blade model and the response is computed using impulse-based simulation. In this simulation the collision detection took an average of* 18*ms per timestep. The error bound,* ε*, was set to be 0.04% of the width of the Lucy.*

interactive rendering [YSGM04]. This can be especially important for handling massive models. The memory requirements of storing separate representations for collision detection and rendering may be prohibitively high. LOD selection for collision detection and graphical rendering can be unified by appropriate error metrics. When computing the BVTT we stop the traversal only when metrics for both collision tests and visual rendering have been satisfied. Similarly, the PMs are refined so that the LOD error is less than the error bounds for both collision detection and visual rendering.

7. Implementation and Performance

In this section we describe our implementation and highlight its performance on complex models.

7.1. Implementation

We have implemented our out-of-core simplification and runtime system on a dual 2.4GHz Pentium-IV PC, with 1GB of RAM and a GeForce FX 5950 Ultra GPU with 128MB of video memory. Our system runs on Windows XP and uses the operating system's virtual memory through memory mapped files for out-of-core access to the data.

We achieve high throughput for rendering and sub-objects culling from graphics cards by storing the mesh data on the GPU, thereby reducing the data transferred to the GPU each frame. We use the GL_ARB_vertex_buffer_object OpenGL extension that performs GPU memory management for both the vertex and the face arrays. Each timestep we only to need to update the BVs and mesh data of clusters whose PMs have changed refinement level since the previous timestep. Furthermore, we use GL_NV_occlusion_query extension to perform collision culling.

Model	Lucy	PP	Turbine	Dragon
Triangles (M)	28	12.8	1.7	0.8
Num Clusters (K)	14	6.4	3.4	1.7
Size of CHPM (MB)	1341	849	88	48

Table 2: *Benchmark Models* Model complexity and number of cluster are shown.

7.2. Benchmark Models

Our algorithm has been applied to two different applications with massive models. They are :

Dynamic simulation: A Lucy model falling onto the CAD model of a turbine blade.

Navigation: A user navigating in a coal-fired power plant model with a flying dragon model.

The Lucy model composed of more than 28 million polygons (Fig. 5), the power plant consisting of more than 12 million polygons and 1200 objects (Fig. 6), the CAD turbine model consisting of a single 1.7-million polygon object (Fig. 5), and the dragon model consisting of 800 thousand polygons (Fig. 6). The details of these models are shown in Table 2.

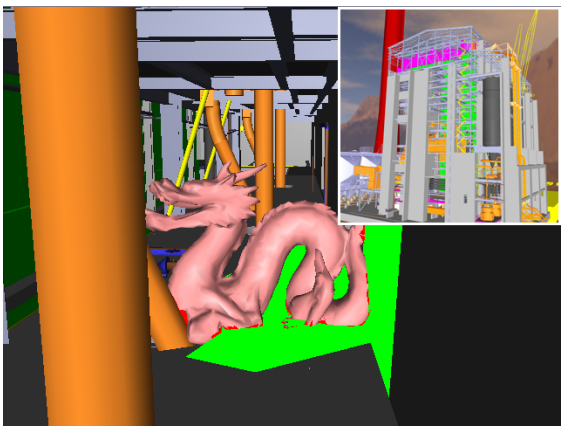

Figure 6: Collision Example. *We tested our conservative collision detection algorithm on a path for the* 0.8*M triangle dragon model along a path through the* 12*M-triangle power-plant model. The average collision query time is* 55*ms and the total memory footprint is* 200*MB. The error bound is set to* 0.04*% of the width of the dragon model. In this path the models have deep penetration and this screenshot colliding triangles are show in red and green. In the upper right the entire power plant model is shown to illustrate its complexity.*

7.3. Performance

Dynamic simulation We have implemented an impulse based rigid body simulation[MC95]. We are able to perform collision detection between the Lucy and blade model at an interactive rate (12-30 frames per second). An image sequence from this simulation is show in Fig. 5. The average collision query time was 18ms. Moreover, we are able to simultaneously perform interactive rendering and collision detection by using a 250MB memory footprint. Most of the query time is spent on the sub-object culling (55%) and very little is spent on PM and cluster refinement (1%).

Navigation For our navigation benchmark we moved a 0.8M triangle dragon model along a path in the 12M triangle power plant model and detected collisions with the objects in the power plant model. Fig. 6 shows a screenshot from the path. The average query time is 55ms and the memory footprint is 200MB.

7.4. Memory requirement

Our CHPM as a dual hierarchy requires 122MB per million vertices on average. Quantization for geometry and compression on PMs can further improve the memory requirement This is low compared to around 560MB per million vertices needed to represent an OBBtree [GLM96]. Furthermore, our out-of-core representation keeps only the cluster hierarchy and the PMs of a subset of the clusters in the main memory.

8. Analysis and Limitation

In this section, we briefly discuss factors that affect the performance of our algorithm and its limitations.

8.1. Performance Analysis

The performance of our algorithm depends on many factors including motion coherence, relative contact configuration, model tessellation, and the error bound, ε. In general, our algorithm achieves the highest speed-up in regions of contact between highly-tessellated and almost flat surfaces. In such regions, the algorithm computes a drastic simplification with a low Hausdorff deviation. Furthermore, the OBBs fit flat mesh regions more tightly than those regions with high curvature.

Our algorithm also exploits temporal and spatial coherence between successive frames. The coarse-grained cluster level refinement performs incremental computations to refine the front. The out-of-core management relies on coherence between timesteps to fetch and prefetch PMs from the disk. We also exploit coherence to reuse bounding box data loaded into the GPU memory, which is needed to obtain high throughput from the GPUs for occlusion queries.

8.2. Comparison with CLODs

CLODs proposed by Otaduy and Lin [OL03] are precomputed dual hierarchies of static LODs used for multiresolution collision detection. The precomputed LODs and their bounding volume hierarchies are used to accelerate collision computations at runtime. As a result, the runtime overhead of CLODs is relatively small as compared to our approach. However, switching LODs between static LODs in the CLOD-algorithm can result in a large discontinuity in the simulation. On the other hand, our algorithm provides smooth fine-grained local control of simplification error within each cluster. This operation is very efficient and reduces the "popping" or discontinuities between successive collision queries. The underlying formulation of CLODs assumes that the input model is a closed, manifold solid and is not applicable to polygon soups. On the other hand, our algorithm is applicable to all models, including polygon soups.

8.3. Limitations

Our algorithms works well for our current set of applications. However, it has some limitations. It relies on temporal coherence for out-of-core management, front computation, and GPU memory management. In situations where many objects come into close proximity within a single timestep, memory stalls may occur as PMs are fetched from the disk. Also, if there is little motion coherence between successive instances then fetching for out-of-core may not keep up with the simulation. Moreover, our algorithm can be very conservative in some cases. Our surface deviation error bounds

may not be very tight for certain meshes. Moreover, our algorithm can be overly conservative and may return too many "false positives." An example is two objects (e.g. two concentric spheres) in parallel close proximity with a separation distance $\delta > d < \delta + \varepsilon$.

9. Conclusions and Future Work

We have presented a new algorithm for out-of-core collision detection using the CHPM representation. There are many benefits to this approach:

- We are able to accelerate the computation using LODs while ensuring all contact regions are detected.
- Our algorithm efficiently handles models with tens of millions of triangles using out-of-core computations.
- The CHPM representation and supporting algorithms can handle models of arbitrary topology and polygon soups.
- We use a unified representation for collision detection and interactive rendering of massive models and use a finite-memory footprint.

There are several areas for future work. First, we would like to develop tighter error bounds to reduce the number of false positives, and thereby decrease the number of bounding volume tests. Second, we would like to incorporate other error metrics, such as those based on the object velocity and size, as well as visual perception. Our current framework can be easily extended to the other metrics by replacing the constant ε with a function $\varepsilon()$. Thirdly, we would like to apply our LOD-based collision detection framework to several applications including: motion planning, navigation, and dynamic simulation. Lastly, we would like to extend our algorithms to perform other proximity queries such as computing separation distance, penetration depth, and contact normals.

Acknowledgments

Our work was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards ACI 9876914 and ACR-0118743, ONR Contracts N00014-01-1-0067 and N00014-01-1-0496, DARPA Contract N61339-04-C-0043 and Intel.

We thank Kitware for the CAD Turbine model. The Dragon and Lucy models are courtesy of the Stanford Computer Graphics Laboratory. The power plant environment is courtesy of an anonymous donor. We would like to thank Miguel Otaduy, Stephane Radeon, and the other members of the Gamma research group at UNC for their useful discussion and support.

References

[BKSS90] BECKMANN N., KRIEGEL H., SCHNEIDER R., SEEGER B.: The r*-tree: An efficient and robust access method for points and rectangles. *Proc. SIGMOD Conf. on Management of Data* (1990), 322–331. 3

[CVM*96] COHEN J., VARSHNEY A., MANOCHA D., TURK G., WEBER H., AGARWAL P., BROOKS F., WRIGHT W.: Simplification envelopes. In *Proc. of ACM Siggraph'96* (1996), pp. 119–128. 2

[EMB01] ERIKSON C., MANOCHA D., BAXTER B.: Hlods for fast display of large static and dynmaic environments. *Proc. of ACM Symposium on Interactive 3D Graphics* (2001). 2, 4

[ESB02] EL-SANA J., BACHMAT E.: Optimized view-dependent rendering for large polygonal dataset. *IEEE Visualization* (2002), 77–84. 3

[ESV99] EL-SANA J., VARSHNEY A.: Generalized view-dependent simplification. *Computer Graphics Forum* (1999), C83–C94. 3

[FNB03] FRANQUESA-NIUBO M., BRUNET P.: Collision prediction using mktrees. *Proc. CEIG* (2003), 217–232. 3

[GH97] GARLAND M., HECKBERT P.: Surface simplification using quadric error bounds. *Proc. of ACM SIGGRAPH* (1997), 209–216. 2

[GLM96] GOTTSCHALK S., LIN M., MANOCHA D.: OBBTree: A hierarchical structure for rapid interference detection. *Proc. of ACM Siggraph'96* (1996), 171–180. 3, 5, 10

[GLM04] GOVINDARAJU N., LIN M., MANOCHA D.: *Fast and reliable collision detection using graphics hardware*. Tech. rep., University of North Carolina, Department of Computer Science, 2004. 3, 8

[GRLM03] GOVINDARAJU N., REDON S., LIN M., MANOCHA D.: CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2003), 25–32. 3, 8

[Hop96] HOPPE H.: Progressive meshes. In *Proc. of ACM SIGGRAPH* (1996), pp. 99–108. 2, 3, 4

[Hop97] HOPPE H.: View dependent refinement of progressive meshes. In *ACM SIGGRAPH Conference Proceedings* (1997), pp. 189–198. 2

[HTG03] HEIDELBERGER B., TESCHNER M., GROSS M.: Real-time volumetic intersections of deforming objects. *Proc. of Vision, Modeling and Visualization* (2003). 3

[Hub93] HUBBARD P. M.: Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality* (October 1993). 3, 5

[JTT01] JIMENEZ P., THOMAS F., TORRAS C.: 3d collision detection: A survey. *Computers and Graphics 25*, 2 (2001), 269–285. 3

[KHM*98] KLOSOWSKI J., HELD M., MITCHELL J., SOWIZRAL H., ZIKAN K.: Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Trans. on Visualization and Computer Graphics 4*, 1 (1998), 21–37. 3

[KOLM02] KIM Y., OTADUY M., LIN M., MANOCHA D.: Fast penetration depth computation using rasterization hardware and hierarchical refinement. *Proc. of Workshop on Algorithmic Foundations of Robotics* (2002). 3

[KP03] KNOTT D., PAI D.: Cinder: Collision and interference detection in real-time using graphics hardware. *Proc. of Graphics Interface* (2003), 73–80. 3

[LDF97] L. DE FLORIANI P. MAGILLO E. P.: Building and traversing a surface at variable resolution. In *IEEE Visualization* (1997). 3

[LE97] LUEBKE D., ERIKSON C.: View-dependent simplification of arbitrary polygon environments. In *Proc. of ACM SIGGRAPH* (1997). 3

[LGLM00] LARSEN E., GOTTSCHALK S., LIN M., MANOCHA D.: Distance queries with rectangular swept sphere volumes. *Proc. of IEEE Int. Conference on Robotics and Automation* (2000). 4, 7

[LM03] LIN M., MANOCHA D.: Collision and proximity queries. In *Handbook of Discrete and Computational Geometry* (2003). 3

[LRC*02] LUEBKE D., REDDY M., COHEN J., VARSHNEY A., WATSON B., HUEBNER R.: *Level of Detail for 3D Graphics*. Morgan-Kaufmann, 2002. 2

[MC95] MIRTICH B., CANNY J.: Impulse-based simulation of rigid bodies. In *Proc. of ACM Interactive 3D Graphics* (Monterey, CA, 1995). 10

[OD01] O'SULLIVAN C., DINGLIANA C.: Collisions and perception. *ACM Trans. on Graphics 20*, 3 (2001), pp. 151–168. 1, 3, 5

[OL03] OTADUY M. A., LIN M. C.: CLODs: Dual hierarchies for multiresolution collision detection. *Eurographics Symposium on Geometry Processing* (2003), 94–101. 1, 3, 5

[Paj01] PAJAROLA R.: Fastmesh: Efficient view-dependent mesh. In *Proc. of Pacific Graphics* (2001), pp. 22–30. 3

[TCL99] TAN T.-S., CHONG K.-F., LOW K.-L.: Computing bounding volume hierarchies using model simplification. In *ACM Symposium on Interactive 3D Graphics* (1999), pp. 63–70. 1

[WLML99] WILSON A., LARSEN E., MANOCHA D., LIN M. C.: Partitioning and handling massive models for interactive collision detection. *Computer Graphics Forum (Proc. of Eurographics) 18*, 3 (1999), 319–329. 3

[XESV97] XIA J., EL-SANA J., VARSHNEY A.: Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (June 1997), 171–183. 2, 3

[YSGM04] YOON S.-E., SALOMON B., GAYLE R., MANOCHA D.: *Quick-VDR: Interactive View-Dependent Rendering of Massive Models*. Tech. Rep. TR04-011, Univeristy of North Carolina-Chapel Hill, 2004. 2, 3, 4, 5, 9