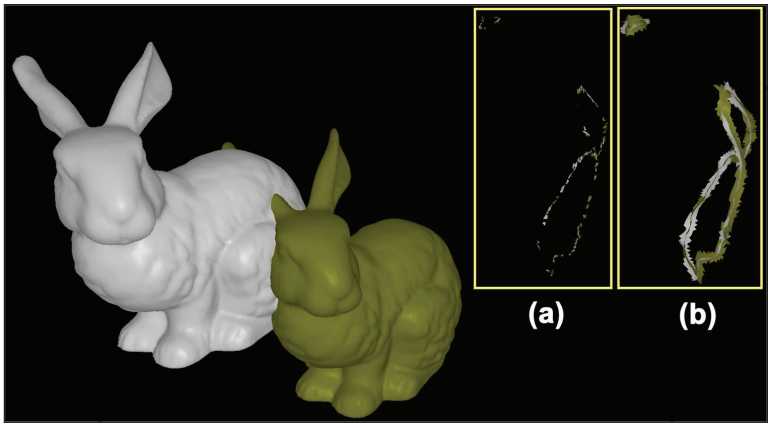
Fast and Reliable Collision Culling using Graphics Hardware

Department of Computer Science University of North Carolina at Chapel Hill November 2004

The Challenge

GPUs are well-optimized for 3-D vector and matrix operations, and complex computations on the frame-buffer pixel or image data. These operations are efficiently processed using multiple vertex and pixel processing units, each of which is programmable, allowing a user to execute a custom program. Moreover, the capabilities of GPUs to perform frame-buffer computations has been growing at a rate faster than Moore's law for CPUs. Different algorithms have exploited these capabilities to compute interferences or overlapping regions or to cull away portions of the models that are not in close proximity. These GPU-based collision detection algorithms are widely used for performing interactive simulations in gaming and virtual reality applications, robot motion-planning, line-of-sight queries etc. Further, many of these algorithms involve no preprocessing and therefore apply well to both rigid and deformable environments. In practice, GPU-based algorithms can offer better runtime performance as compared to object-space collision detection algorithms.



Precision: Our algorithm computes reliable collisions between the two bunnies, each with 68K triangles. The top right image (b) shows the output of our algorithm and the top left image shows the output of a GPU-based algorithm CULLIDE at 1400x1400 resolution. CULLIDE misses many collisions due to sampling errors

However, GPU-based collision detection algorithms suffer from limited precision. This problem is due to the limited viewport resolution (up to 11 bits on current GPUs), sampling errors and depth precision errors. Due to these errors, prior GPU-based algorithms may miss collisions and may result in an inaccurate simulation. In contrast, object-space collision detection algorithms are able to perform more accurate interference computations using IEEE 32 or 64-bit floating arithmetic on the CPUs.

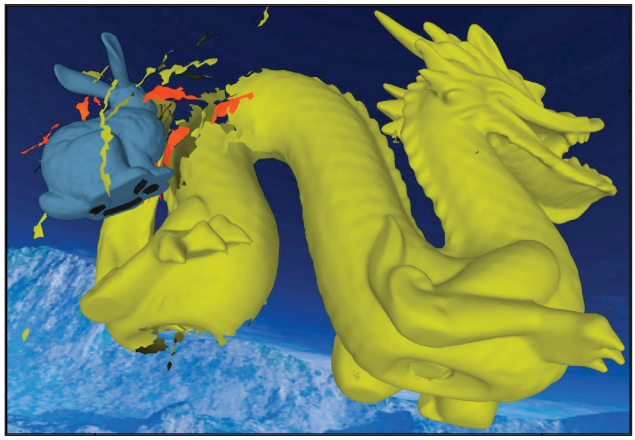
Highlights

- **More reliable computations over prior GPU-based algorithms**
- **More effective culling over prior CPU-based algorithms**
- **Broad applicability to non-manifold geometry, deformable models, and breaking objects**
- **Interactive performance with no preprocessing and low memory overhead**

We present a simple and efficient algorithm for fast and reliable collision culling between triangulated models in a large environment using GPUs. We perform visibility queries to eliminate a subset of primitives that are not in close proximity, thereby reducing the number of pairwise tests that are performed for exact proximity computation. We show that the *Minkowski sum* of each primitive with a sphere provides a conservative bound for performing reliable 2.5D overlap tests using GPUs. For each primitive in a collection of triangles, our algorithm computes a tight bounding offset representation. The bounding offset representation is a union of object oriented bounding boxes where each OBB encloses a single triangle. Our algorithm performs visibility queries using these UoBBs on GPUs to reject primitives that are not in close proximity. Our algorithm guarantees that no collisions will be missed due to limited framebuffer precision or quantization errors during rasterization.

Algorithm

Our algorithm CULLIDE uses the imagespace occlusion queries available on current GPUs and computes a potentially colliding set (PCS) of objects. Given n objects that are potentially colliding P_1, ..., P_n, we present a linear time two-pass rendering algorithm to test if an object P_i is fully visible against the remaining objects, along a view direction. Occlusion queries are used to test if an object is fully visible or not. To test if an object P is fully visible against a set of objects S, we first render S into the frame buffer. Next, we set the depth function to GL GEQUAL and disable depth writes. The object P is rendered using an occlusion query. If the pixel pass count returned by occlusion query is zero, then the object P is fully visible and therefore, does not collide with S. Using this formulation, we prune objects P_i that do not overlap with other objects in the environment. The algorithm begins with empty frame buffer and



Dynamically Generated Objects: A breaking object simulation in which a bunny composed of 35K triangles falls on a dragon composed of 112K triangles and decomposes the dragon. Our algorithm takes 30 – 60 ms to compute all the collisions during the simulation.

proceeds in two passes as follows:
In the first pass, we rasterize the primitives in the order $P_1, ..., P_n$ testing if they are fully visible. In this pass, if a primitive P_i is fully visible, then it does not intersect any of the objects $P_1, ..., P_i$-1. In the second pass, we perform the same operations but render the primitives in the order $P_n, .., P_1$. In this pass, if a primitive P_i is fully visible, then it does not intersect any of the objects $P_n, .., P_i$+1. At the end of two passes, if a primitive is fully visible in both the passes, the primitive does not interfere with the remaining primitives and is removed from the PCS. The view directions are chosen along the world-space axes and collision culling is performed using orthographic projections.

In order to resolve sampling errors, we compute the Minkowski sum of a sphere whose radius is a function of the viewport resolution and frame-buffer precision and each primitive P, and use it for performing reliable collision culling.

Results

We utilize the GPU for fast and reliable pruning of primitive pairs and perform exact interference tests on the CPU. We have implemented this collision culling algorithm on a Pentium IV PC with NVIDIA GeForce FX 5950 card. Our results are demonstrated on two complex simulations as shown in the supplementary images and video at http://gamma.cs.unc.edu/FAR

Dynamically generated objects: A breaking object simulation in which a bunny composed of 35K polygons falls on a dragon composed of 112K polygons and fractures the dragon. As the simulation progresses, hundreds of broken pieces are introduced and our algorithm takes 30 - 60 ms to compute all collisions.

Tree with falling leaves: In this scene, leaves fall from the tree and undergo non-rigid motion. They collide with other leaves and branches. The environment consists of more than 40K triangles and 150 leaves. Our algorithm, FAR, can compute all the collisions in about 35 msec per time step.

Non-Rigid Motion: A non-rigid simulation in which leaves collide with branches of the tree. The environment has more than 40K triangles and 150 leaves. Average collision query time is 35 ms.

Team Members

Ming C. Lin, professor
Dinesh Manocha, professor
Naga Govindaraju, research assisant professor

Research Sponsors

U.S. Army Research Office
Defense Advanced Research Projects Agency
Intel Corporation
National Science Foundation
Office of Naval Research

Selected Publications

Govindaraju, N., S. Redon, M. Lin, and D. Manocha. CULLIDE: Interactive collision detection between complex models in large models using graphics hardware. *ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 2003.

Govindaraju, N., M. Lin, and D. Manocha. Fast and reliable collision culling using graphics processors. *ACM Symposium on Virtual Reality Software and Technology*, 2004.

Govindaraju, N., M. Lin, and D. Manocha. Quick-CULLIDE: Fast Inter- and Intra-Object Collision Culling Using Graphics Hardware. To appear in the *Proc. of IEEE Virtual Reality Conference*, 2005.

Key Words

Geometric algorithms, graphics hardware

For More Information

http://gamma.cs.unc.edu/FAR

http://gamma.cs.unc.edu/FAR