Fast continuous collision detection among deformable Models using graphics processors

CS-525 Presentation

Presented by

Harish
Abstract:

We present an interactive algorithm to perform continuous collision detection between general deformable models using graphics processors (GPUs). We model the motion of each object in the environment as a continuous path and check for collisions along the paths. Our algorithm precomputes the chromatic decomposition for each object and uses visibility queries on GPUs to quickly compute potentially colliding sets of primitives. We introduce a primitive classification technique to perform efficient continuous self-collision. We have implemented our algorithm on a 3.0GHz Pentium IV PC with a NVIDIA 7800 GPU, and we highlight its performance on complex simulations composed of several thousands of triangles. In practice, our algorithm is able to detect all contacts, including self-collisions, at image-space precision in tens of milli-seconds.

Introduction:

The problem of collision detection (CD) arises in modeling, simulation and interaction for many different areas, including virtual reality (VR), cloth simulation, haptic rendering, animation, rapid prototyping, CAD/CAM, robotics, and entertainment. In this paper, we primarily focus on CD over a given time period between deformable objects that includes inter-object collisions between disjoint objects and intra-object collisions (or selfcollisions) within each deformable model. Most of the earlier work in CD has been restricted to CD at discrete time instances and these algorithms may not check for possible overlaps between successive time steps. As a result, it is possible to miss a collision and result in visual artifacts, inconsistent state, or incorrect simulation that can significantly affect the sense of immersion and lead to break in presence (BIP) in a virtual environment (VE). Such problems can be
especially challenging in environments composed of thin or fast moving objects, like cloth on virtual avatars.

In order to overcome the limitations of discrete CD Algorithms, different techniques have been proposed that model the motion between successive time instances as a continuous path and check for collisions along these paths. These are classified as continuous collision detection (CCD) algorithms. However, current CCD algorithms are only able to handle rigid objects, articulated models, or simple deformable meshes (consisting of only a few hundreds of polygons) at interactive rates.

**Related work:**

The problem of CD has been extensively studied in the literature. A good overview of different algorithms is available in some recent surveys. In this section, we give a brief survey of earlier work related to CCD, GPU based methods and CD between deformable models.

1. Continuous collision detection
2. Collision detection between deformable models
3. GPU-based collision detection algorithms

**Overview:**

In this section, we give an overview of our CCD algorithm. We first formulate the problem of CCD among general deformable models. We present a novel classification algorithm to perform fast continuous collision culling between general deformable models.

1. Problem formulation
2. Continuous collision culling
3. Algorithm
4. Analysis on mesh decomposition
Fig. 1. Benchmark I: The cloth is modeled using 7K triangles and our mesh decomposition results in 11 colors (as shown in (b)). The chromatic decomposition algorithm results in 23 colors and the comparison is highlighted in Fig. (c). Using AABBs, the PCS is reduced to 2000–2500 triangles and is highlighted in red in (d). Using the stage II of our algorithm, we further reduce the size of the PCS to 40 primitives and is shown in (e). The sequence of images (f), (g) show the various instances of a skirt due to the motion of the avatar. As the simulation progresses many complex folds and wrinkles arise. Our algorithm is able to detect all the collisions within 100 ms on a Pentium IV PC with a GeForce 7800GPU.

Fig. 2. Benchmark II: In this simulation, a cloth modeled using 6K triangles falls and drapes around a sphere. Our mesh decomposition algorithm partitions the mesh into 10 independent sets as shown in (b). We show a comparison with the chromatic decomposition algorithm in (c). Using the AABBs in the stage I of our algorithm, the size of PCS is reduced to 1600 triangles. The stage II of our algorithm further reduces the size of the PCS to 30 triangles. The PCS obtained after stages I and II are highlighted in red in (d) and (e), respectively. We illustrate the various instances of the simulation in (f), (g), respectively. The average collision detection time is 80 ms.
**Interactive continuous collision detection:**

In this section, we present our multi-stages CD algorithm. We first describe the pre-processing phase followed by the runtime algorithm.

1. Pre-process
2. Bounding representations
3. Run-time algorithm

---

![Figure 3](image-url)

**Fig. 3.** Benchmark III: In this simulation, the cloth is modeled using 12:5K triangles. Our mesh decomposition algorithm partitions the mesh into 10 independent sets as shown in (b). Our AABB-based collision culling algorithm in stage I reduces the size of PCS to 6000 triangles. The stage II of our algorithm further reduces the size of the PCS to 1000 triangles. The PCS obtained after stages I and II are highlighted in red in (c) and (d), respectively. We illustrate the various instances of the simulation in (e), (f), (g) and (h), respectively. The average collision detection time is 300 ms.
Fig. 4. Culling efficiency and performance on Benchmarks I, II, and III: we highlight the amount of culling efficiency obtained in each of the stages in (a), (c), and (e). The graphs indicate over an order of magnitude improvement in culling efficiency using stage II over stage I. (b), (d) and (f) highlight the time taken to update the AABB hierarchy, the collision time in stage I and the total collision time. The average collision time is around 100 ms for Benchmark I, 90 ms for Benchmark II and 300 ms for Benchmark III.
Implementation and results:

The authors have implemented this algorithm on a Pentium IV PC with 2GB memory and NVIDIA GeForce 7800 GTX GPU. They have used the OpenGL API under Windows XP and GL_NV_occlusion_query for performing the full visibility queries asynchronously. We improve the rendering performance by storing the mesh vertices using the GL_vertex_buffer_object extension on the GPU. They were able to achieve a throughput of 20 million triangles per second. They used an image resolution of 1000 _ 1000 for collision computations, and used three axis-aligned views to perform collision culling. They have applied this algorithm to detect collisions on general meshes used for cloth simulation.

Analysis:

In this section, we analyze the performance of our algorithm. This includes the culling efficiency and different factors that govern the overall performance.

1. Performance
2. Culling efficiency
3. Comparison
4. Limitations

Conclusions and future work:

The authors have presented an interactive continuous collision detection algorithm for general deformable meshes in virtual environments. They have decomposed the problem into adjacent and non-adjacent collision detection. The algorithm takes advantages of primitive connectivity information to perform efficient collision culling between bounding swept volumes to ensure no collision is missed between time steps. The collision culling is performed using 2.5D queries on the GPUs. It is able
to check for collisions, including self-collisions, in complex simulations consisting of many thousands of triangles. There are many avenues for future work. The authors would like to use this approach for other application including avatar motion and surgical simulation. Furthermore, they would like to perform other proximity queries including separation distance and penetration depth computation.

**Acknowledgments:**

This research is supported in part by ARO Contract DAAD 19-02-1-0390, and W911NF-04-1-0088, NSF Awards 0400134, 0118743, DARPA and RDECOM Contract N61339-04-C-0043 and Intel Corporation. We thank the UNC GAMMA group for many useful discussions and support. We are also grateful to the reviewers for their feedback.