Transformation Semantics: An Efficient Approach for Collision Detection

José Gilvan Rodrigues Maia, Creto Augusto Vidal, Joaquim Bento Cavalcante-Neto
Departamento de Computação – Universidade Federal do Ceará (UFC)
60455-760 – Fortaleza – CE – Brazil
{gilvan, cvidal, joaquimb}@lia.ufc.br

Abstract

Collision detection is an important problem in many kinds of applications. This work presents an efficient approach for exact collision detection between complex, deformable models. The approach consists of a method for fast extraction of semantics from transformation matrices which places models in a scene, together with a general strategy for fast intersection tests. Non-uniform scaling is also supported efficiently. Our experiments demonstrate that our strategies are well suitable for real-time applications.

1. Introduction

Collision detection is a fundamental task for many kinds of applications, such as robotics, industrial prototyping, physics simulation, games and virtual reality (VR), which requires precise detection of interactions between objects in a simulated environment [1]. In Figueiredo et al. [2], the authors point out that such applications need realistic modeling of interaction between the elements of the environment in order to obtain convincing results. Simulation of simple contacts, impacts and deformations demands robust geometric algorithms for detecting collisions, computing distances and determining the contact surface, if any [3].

Intersection algorithms must be robust in order to correctly compute results in the many possible situations, so they do not report any false positives or negatives. Developing such algorithms is not only theoretically difficult, but the limited floating-point representation implies various practical aspects when creating strategies for collision detection. On the other hand, interactive applications such as games and VR also require efficient tests for maintaining real-time performance [3].

There are a number of software libraries for collision detection between polygonal models [1, 4, 5, 6, 7], usually consisting of a triangle mesh. Such libraries employ the most diverse strategies for efficiently computing intersections or distances. Some of these libraries provide various kinds of tests and support scenes composed by multiple models in motion [4, 5, 6, 7], while other libraries only work with an object pair per time [1]. These collision detection libraries do not provide any direct support for representations such as terrains, triangle fans and triangle strips, which are widely used for rendering. This not only introduces overheads when using such representations, but also creates difficult collision queries when such representations are needed.

Support for scaling models is an important feature for modeling virtual environments, allowing flexible instancing of models in a scene [8]. Most of these collision detection libraries also lack support for nonuniform scaling models, and the few libraries supporting this feature require explicit scale factors to be provided *a priori* by users.

In this work, we describe an efficient approach for real-time collision detection involving rigid and deformable geometric models, aimed at developing general-purpose interactive applications. Our strategy is based on extraction of semantics from transformation matrices and their use for computing intersections in different coordinate systems. We show how the choice of coordinate systems affects performance and complexity of intersection algorithms. We also show how to rapidly compute a global Axis-Aligned Bounding-Box (AABB) from a local AABB and a transformation matrix. These strategies are implemented as a library that is very easy to integrate not only with high-level systems but with those representations typically used by low-level rendering interfaces such as OpenGL and Direct3D.

The rest of the paper is organized as follows. In Section 2, it is given an overview of the related work on collision detection. The principal theoretical

elements for the proposed strategies are described in Section 3. In Section 4, we present a discussion about the most important implementation aspects of the proposed approach. The experiments performed using our strategies and a discussion on their results are described in Section 5. Finally, we present an analysis of this work and its limitations in Section 6.

2. Related work

There are many techniques for intersection tests in the literature. These techniques sometimes are used to develop representative collision detection libraries. In the following we present these techniques and the libraries, as well as the strategies they employ and their limitations.

2.1. Intersection tests

Intersection tests can be classified, according with movement of models, in three categories: static, continuous and pseudo-continuous [3]. Changes in positions and orientations of objects are not considered when performing static tests, while in continuous tests these positions and orientations can vary with time. Computations needed for static tests are typically much simpler than those required by continuous tests. Because of this, pseudo-continuous tests are usually used as an approximation for continuous tests by performing successive static tests [9]. Collision tests can also support (or not) deformable or breakable models. The capability of detecting self-intersections is also required in order to respect physical constraints when models are deformed.

In scenes containing multiple objects, collision detection is usually performed by two distinct phases: broad phase and narrow phase [2]. Distant, non-overlapping pairs are rapidly discarded by using simple tests during broad phase. Pairs remaining from this early exit are individually processed during the narrow phase, whereas the most complex intersection tests take place in order to determine the contact surface, if any.

Different intersection tests usually give rise to different contact information. The time of impact is required for most continuous tests, for example. Distinct applications may entail different data when an intersection is found. Maia et al. [3] categorize intersection tests, according with the information they report, as: picking or ray casting tests; distance computing, interpenetration vectors or closest feature tests; volume versus polygonal model tests; and model versus model tests.

2.2. Strategies for real-time collision detection

Various algorithms have been proposed in literature for detecting collisions. A classification of such algorithms is an important tool when highlighting their advantages and their drawbacks for real-time applications. Figueiredo et al. [2] present a taxonomy for collision detection algorithms, which categorizes such algorithms in two groups based on their data structures: Spatial Subdivision and Bounding Volume Hierarchies (BVHs).

Spatial subdivision structures such as octrees, grids, k-d trees, R-trees and BSP-trees can be adapted to represent scenes composed by polygonal models, providing spatial coherence that is exploited for faster culling and intersection tests. However, according with [2], such representations are not well suited for broad phase in dynamic scenes and for narrow phase involving deformable models, because deformations on models demand refitting the subdivision, which may be an expensive task.

BVHs are multi-resolution representation for polygonal models that are useful for rapidly rejecting non-intersecting pairs (either objects or primitives) [1]. Although BVHs cannot detect an object inside another because they are usually based on boundary representations, they are very useful for collision detection [7]. The main drawbacks of BVHs are the elevated memory requirements and the time for refitting hierarchy to the geometry of deformable models. Researchers have proposed various strategies for overcoming such limitations and improving efficiency of BVH-based intersection tests [4, 6, 10].

Besides these two groups of collision detection algorithms, there are other three important groups, whose strategies are based on Distance Fields, Spatial Hashing and Graphics Hardware, respectively.

Discrete distance fields [11] can be used to represent volumetric objects and to efficiently perform collision detection tests. However, this strategy is inadequate for real-time applications in dynamic scenes composed by complex models [7, 11]. Hashing functions can speed up intersection computations in scenes containing small objects, and do not require any special pre-processing [12, 13]. This approach is clearly efficient for highly dynamic scenes and, moreover, provides natural support for deformable and breakable models. Such approach is more suitable for scenes with a fine level of mesh discretization.

Other kind of algorithms exploits the capabilities of graphics hardware for dramatically accelerating intersection tests [14, 15]. Such algorithms depend on specific hardware, which is a requirement that cannot be always fulfilled in low-cost systems, for example.

Algorithms and heuristics were developed specially for use in specific stages of collision detection. Cohen et al. [5] developed a sort-based Sweep and Prune algorithm for broad phase, which is linear over the number of objects. Gottschalk [16] introduced the Separating Axes Theorem (SAT) for intersection tests. Tests based on linear systems and optimization problems for polihedra were replaced by iterative searches for a separating axis.

Bergen [4] points out that, when a hierarchy of AABBs or OBBs (Oriented Bounding Boxes) is employed for collision culling between two models, about 60% of the separating axes found corresponds to a face normal. Based on this, Bergen proposed an approximate test (SAT-lite) that considers only face normals as separating axis when performing collision detecting against AABB trees. The iterative method of Gilbert-Johson-Keerthi [17] for distance computation, referenced in the literature as GJK, has influenced many researchers and their algorithms [18, 19]. The principal advantages of GJK are: simplicity, performance and easy generalization for diverse primitives [19].

2.3. Collision detection libraries

A variety of collision detection libraries were developed for use in general interactive applications, providing support for intersection tests against complex polygonal models. The following libraries are noticeable because of their popularity and performance: SOLID, I-Collide, Opcode, RAPID and SWIFT++ [4, 5, 6, 1, 7]. Most of these libraries exploit temporal coherence (TC) between successive frames for further speed up of intersection tests.

These libraries represent polygonal models through a triangle mesh or a simple polygon mesh, and do not consider that graphics engines commonly used by applications for rendering can represent models in terms of triangle fans or strips. These representations require less memory and are more efficient for rendering [3, 20]. Thus, the use of these collision detection libraries yields triangle mesh conversion overheads.

RAPID [1] introduced OBB trees and SAT for computing intersections between meshes, and does not provide direct support for deformable models. SOLID [4] is based on an extended GJK algorithm [17], and provides support for efficient tests against volumes and deformable models represented by AABB trees. I-Collide [5] implements Sweep and Prune broad phase and is built on top of RAPID. Switf++ [7] represents models trough a hierarchy of convex hulls, and provides support for broad phase, distance computing and mesh versus mesh intersections.

From these libraries, only SOLID provides support for deformable and non-uniformly scaled models. However, the scale factors must be provided a priori by the user application. Deformable models are supported by OPCODE, which does not support any scaling at all. In this library, scaling must be applied to the model's vertices first and then the corresponding AABB tree must be refitted. Clearly this gives raise to an overhead.

The use of deformable models in the other libraries implies complete rebuild of volume hierarchies, which is unfeasible in a real-time application or simply introduces a bottleneck. With the exception of OPCODE, all libraries supporting broad phase do not provide a direct way of testing a given pair of objects and introduce unnecessary burden for simple tasks.

3. Exploiting transformation semantics for detecting collisions

In this section, we first present our method for transformation inversion and semantics extraction. Then, we show how we can use transformation semantics for obtaining efficient intersecting tests without knowing a priori scale factors and for computing global AABBs containing the model's local AABB in world space. AABB trees are the representation of choice for polygonal models, because the hierarchy refitting overhead is moderate [4]. Moreover, our global AABB calculation method makes it easy to use AABB trees for sweep and prune algorithm.

3.1. Efficient Extraction of Transformation Semantics

Consider a transformation matrix **M**, resulting from composition of matrices **S** (scale), **R** (rotation) and **T** (translation), in this order. **M** transforms a locally defined model into world coordinates. Various graphics libraries use exactly this form to represent matrices transforming geometric models, because such representation is clear to comprehend and also allows for optimized routines for updating scene graphs [3, 20]. Wu [21] uses an equivalent representation for **M** for describing a simple and efficient inversion method. However, his method only considers isotropic scales. Our representation for **M**, depicted in (1), is more general and considers non-uniform scales.

We demonstrate that $\mathbf{M}^{-1}$, described in equation (2) can be obtained without computing square roots. The

3x3 part of $\mathbf{M}^{-1}$ is obtained by first transposing each column and then dividing each row by the square of its norm. This can be done because $\mathbf{r}_i$ form an orthonormal basis. The translational part of $\mathbf{M}^{-1}$ is then obtained by multiplying $-\mathbf{t}$ by the recently computed 3x3 part of $\mathbf{M}^{-1}$. Adapting this method for extracting of **R**, **S** and **T** from **M** is straightforward and requires three square roots to be computed. Note that simply inverting **M** does not require any square root to be computed. Although computing square roots is a bit slow, it is a relatively cheap operation in recent hardware.

$$\mathbf{M} = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

$$\mathbf{M} = \begin{bmatrix} s_x\mathbf{r}_1 & s_y\mathbf{r}_2 & s_z\mathbf{r}_3 & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}^{-1} = (\mathbf{TRS})^{-1} = \begin{bmatrix} \mathbf{r}_1^{\mathrm{T}}/\mathbf{s}_x & -\mathbf{r}_1 \cdot \mathbf{t}/\mathbf{s}_x \\ \mathbf{r}_2^{\mathrm{T}}/\mathbf{s}_y & -\mathbf{r}_2 \cdot \mathbf{t}/\mathbf{s}_y \\ \mathbf{r}_3^{\mathrm{T}}/\mathbf{s}_z & -\mathbf{r}_3 \cdot \mathbf{t}/\mathbf{s}_z \\ \mathbf{0}^{\mathrm{T}} & \mathbf{1} \end{bmatrix} \quad (2)$$

Our inversion method is very efficient, requiring 27 multiplications, 3 divisions and 12 additions against Wu's method, which demands 21 multiplications, 1 division and 8 additions. Thus we avoid the brute-force approach that yields 280 multiplications, 1 division and 101 additions. Wu's method is proven to be just about 17% and 31% faster in 10 million executions in Pentium IV and Pentium II processors, respectively. Since square roots are avoided, it is possible to achieve not only better performance, but also robustness in intersection algorithms because half mantissa bits is not lost due to square roots.

3.2. Computation of Global AABBs for Broad Phase

Sweep and Prune algorithm requires global (world space) AABBs containing each model for collision culling. A typical 3D scene usually provides a local AABB for each model, which is transformed into world space by a matrix **M**. The naïve approach for obtaining the global AABB consists of transforming the eight vertices of the local AABB by **M**, and then using these vectors to compute the global AABB. We present a geometric method that is significantly simpler and faster than the naïve approach. Observe that the components of the local AABB's edge vectors are, in world space, complementary with respect to the dimensions of the global AABB that tightly fits the transformed, local AABB. This is illustrated by Figure 1. First, we transform the minimum point of the AABB by **M**, obtaining **p**. The directions of the global edges are obtained efficiently by lookup over the first three columns from **M**. Then these directions are multiplied by the width, height and depth of the local AABB. The sign of each component of each edge vector in world space is checked. If that component is negative, then it moves **p** in the direction of the minimum point of the global AABB. Otherwise, we are moving **p** towards the maximum point.

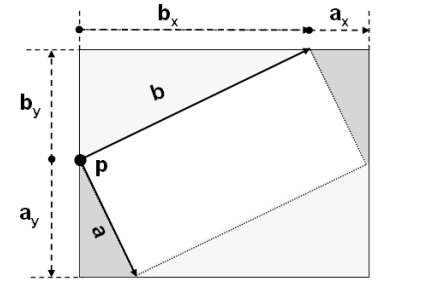

Figure 1. An oriented box and its edge directions, a and b. The maximum and minimum points of the global AABB are p + a_x + b_x + b_y (*positive* components) and p + a_y (*negative* components), respectively.

3.3. Efficient Collision Detection between Deformable Models

Although refitting instead of rebuilding a BVH is cheaper when the respective model changes its shape, this procedure has a considerable cost [4] and can become a bottleneck in scenes containing some complex deformable models or various simpler deformable models. We propose a slight modification in the broad phase procedure that overcomes this problem. The scene provides a local, up-to-date AABB containing each deformable model. An AABB tree is refitted only when the global AABB containing the corresponding model, if deformable, is touched during broad phase. Thus, our procedure updates only those models that must be processed in the narrow phase, and considerably improves the efficiency of intersection tests.

3.4. Primitive Intersection Tests using Transformation Semantics

The intersection algorithms between primitives have to be adapted in order to provide support for scaling. Our approach for adapting such algorithms is

based on the diagram illustrated in Figure 2. The directed graph in this figure establishes a visual reference for concatenation of transformations from A and B coordinate systems, so edges represents the required transformations between coordinate systems, illustrated as vertices. Our intersection algorithms choose one from seven possible coordinate systems to carry out intersection tests. Transformation semantics provide a way to transform primitives to the desired coordinate system. Transformation matrices from A and B to B", for example, is performed by the matrices $\mathbf{T^{-1}_B T_A R_A S_A}$ e $\mathbf{R_B S_B}$, respectively

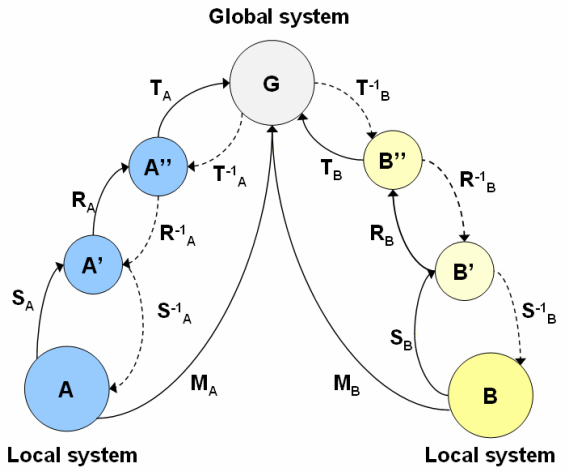

Figure 2. Diagram illustrating the transitions between the seven coordinate systems available for intersection tests.

All the basic intersection tests needed for narrow phase between AABB trees are simple to obtain by using this paradigm. A' and B' systems illustrated in Figure 2 are usually most convenient for tests using SAT, because it requires only the pre-computation of rotations, translations and scales from a system to another. This simplifies intersection tests between OBBs because they relative orientation is the same for the entire AABB hierarchy [4]. The same cached information can be reused for testing an AABB in A system against a triangle in B using SAT, which requires testing 13 potential separating axes. Testing a triangle pair for intersection is similar. However, we prefer computing intersections in A system, because the transformation of all triangles to A' or B' systems is avoided. A cached 4x3 matrix is then used to map triangles from B to A.

Another improvement is obtained when the volume and number of primitives in each model is considered when traversing AABB trees during intersection tests. The adoption of the local system of the most complex model, for example, reduces the total number of transformed triangles before the collision test takes place. Testing a triangle pair using SAT requires checking 11 possible separating axes in the general case and 8 axes when the given triangles are coplanar [9]. In this case, SAT looks at triangles as they were degenerated prisms. This algorithm can be accelerated in the coplanar case by projecting the triangles over the axis-aligned plane that maximizes their area, which simplifies computations.

Intersection tests between an AABB tree and a volume in local system are simplified if accomplished in A', because distortions due to scales can lead to more complex intersection algorithms. A sphere, for example, gets deformed so intersection tests are done against an ellipsoid. Ray casting is simpler, because the endpoints of a line segment can be transformed to the model's local space before the intersection tests take place, and parametric values reported by local tests are still valid in world space. However, special care must be taken when rays are used, because the intersection algorithm must consider scale distortions in the direction.

4. Implementation

The strategies described in prior sections were developed in C/C++ as a modification to OPCODE 1.3 [6]. Tough this library employs various optimization strategies it does not provide any support for scaled models in the variety of intersection tests this library implements. Moreover, the library was modified in order to support not only triangle meshes, but also triangle strips, triangle fans and terrains represented as regular point grids. Original OPCODE does not support 64-bit processors and it is very inefficient for representing non-indexed geometry because memory is wasted with trivial index tables.

Thus two new versions of the library were developed, introducing support for non-uniformly scaled models (version 1.3.1) and for other kinds of geometry (version 1.3.2), indexed or not. This second version is not only more flexible than the original library but saves memory when non-indexed geometry is used. We also added optional intersection tests, like, for example, SAT-based triangle-triangle overlap. OPCODE also allowed us to test the effectiveness of our approach through the variety of intersection tests it provides (picking, volume versus model and model versus model).

5. Results

Our custom versions of OPCODE were used in a variety of tests carried out in a Pentium IV 3GHz with 1GB of RAM memory. The proposed method for computation of global AABBs is about 6 times faster than the naïve algorithm in 10 million executions. This method can be used not only for broad phase collision detection, but also for efficient culling in rendering algorithms. Moreover, our implementation of the broad phase algorithm became about 5.4 times faster in a scene containing 64 deformable models. Of course, time spent for refitting AABB trees was discarded in this test.

The lazy update of AABB trees has proven to be an efficient heuristic for minimizing the bottleneck of refitting BVHs to deformable geometry during runtime. In a sequence of 10K frames in the same scene being animated, only 15 of the 64 models have to be refitted per frame. In other words, our strategy is about 4.26 times faster than the naïve approach for this particular scene. Clearly, collision detection in scenes with a reduced number of models in close proximity will be further accelerated because only those AABB trees corresponding to models involved in narrow phase need to be refitted.

Intersection tests between a pair of polygonal models are faster when the objects are in close proximity, because extraction of transformation semantics requires the computation of three square roots. However, such tests only occur after the broad phase algorithm discarded distant, non-colliding pairs, so the overlap tests between boxes and triangles from BVHs consumes most processing time in this case.

We used an approach similar to that used by Bergen [4] and performed exhaustive pairwise tests between two relatively complex models: bunny (5200 triangles) and cow (5000 triangles). These models were placed inside a cube in random positions and orientations before collision detection takes place. Scaling was not used in this experiment because we wanted to see how our modified OPCODE (1.3.2) compares with the original library (1.3).

Tough computation of three square roots is required by our implementation, it slightly outperforms the original library in the tests when models are in close proximity. As we pointed out, when models are distant, the original library is faster because its box-box SAT test is much simpler than computing three square roots. Our tests also show that triangle-triangle SAT tests discard pairs that are in slight contact, and reports a false negative. We repeated this experiment using a modified SAT that forces the computation of all potential separating axes and uses 'fat' prisms. We concluded that, even in the worst case, SAT is faster than Möller's algorithm originally used by OPCODE. As a result, our implementation is about 7% faster than the original library (see Table 1).

Table 1. Time measured for 50K and 500K pairwise collision tests of two models randomly placed inside a cube.

#tests	50K		500K	
version	**1.3**	**1.3.2**	**1.3**	**1.3.2**
total (ms)	2.221K	2.128K	20.91K	2.004K
#collisions	833	832	8188	8134
#pairs	393.8K	393.6K	3.681M	3.549M
collision time	2.132 K	1.998K	1.994K	1.872K
ms/ collision	2.56	2.40	2.43	2.31
rejection time	88.15	130.41	974.5	13246
ms/ rejection	$1.81e^{-3}$	$2.65e^{-3}$	$1.98e^{-3}$	$2.69\ e^{-3}$

We also developed an interactive application for visualization of the contact surface between models manipulated by the user. Performance tests were carried out in this application using two significantly complex models: Stanford bunny (69K triangles) and whitestar (91.3K triangles). Scales were used in this case, allowing for interaction between the models, because the spaceship model was too small (Figure 3). It takes an average time of 62.5 ms for testing intersection between these models in deep contact situations. The corresponding contact surface contains about 9.81K triangle pairs, resulting in only 6.37ns for testing each pair.

As illustrated in Figure 3, we also carried out a performance analysis of testing intersections between volumes (AABB, OBB, Sphere and Capsule) and geometric models. The performance reported by our tests is about the same of those obtained by the original library, which is quite fast for volume tests.

A ray casting algorithm was implemented in order to exhaustively execute tests of a ray against a model. Intersection tests first were carried out in A' coordinate system. An image with 1024x768 pixels takes 29.3 seconds in average to be rendered from a scene containing 149K triangles. Then we modified the picking routines so intersection tests are carried out in the model's coordinate system, avoiding scaling triangles and boxes during intersection tests. The average rendering time for the same configuration dropped to 16.63 seconds, resulting in a speedup factor of 56.7% for 30 consecutive images. Figure 4 shows the scene being rendered in real-time and the

corresponding image produced by ray casting. Moreover, intersection tests became more robust, particularly when a triangle is hit near edges, because precision is not lost due to the computation of square roots.

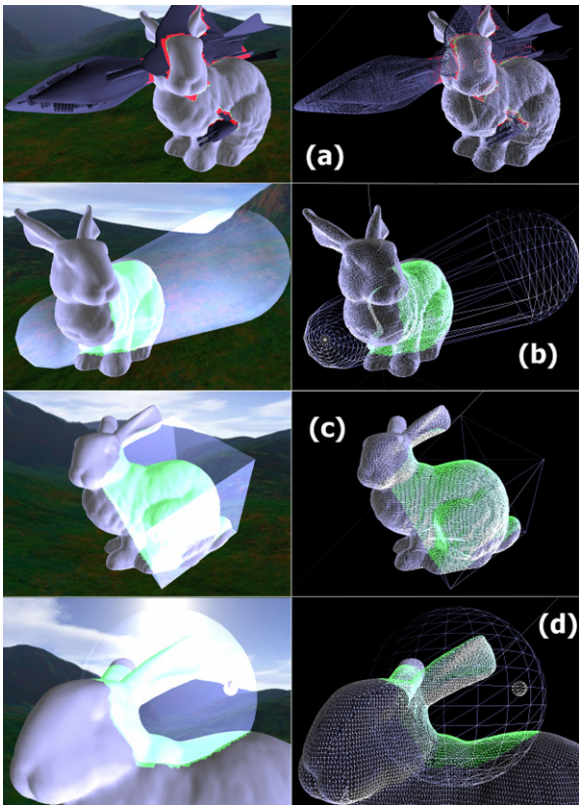

Figure 3. A variety of intersection tests using complex models. The contact surface between two complex models is highlighted (a). Intersection surfaces: against a capsule (b), an oriented box (c) and a sphere (d).

6. Conclusions

Collision detection is an essential task in various applications that demands precise, robust and efficient algorithms. In this work, we presented an approach for exact collision detection between deformable models, which supports non-uniform scaling and accelerates intersection tests based on transformation semantics.

We also presented fast methods for inverting and extracting semantics from transformation matrices used by typical graphics applications. An efficient, geometric algorithm for computation of global AABBs for broad phase collision was also proposed. Moreover, the lazy update of AABB trees during broad phase significantly reduces the number of AABB trees that must be refitted to deformable models. A number of experiments demonstrate that the proposed approach is suitable for real applications. Furthermore, the modified collision library we developed is efficient, flexible and easy to use.

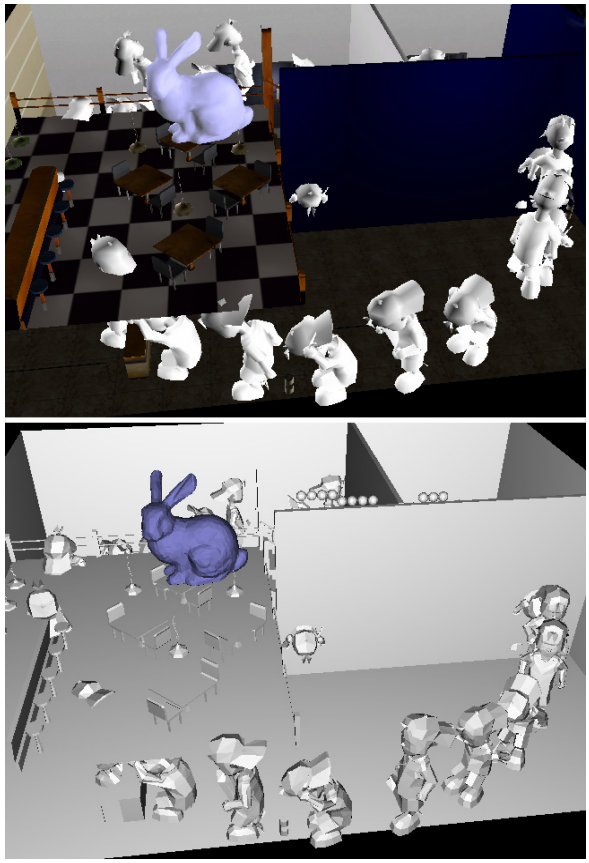

Figure 4. A scene displayed in real-time (top). The same scene is rendered via ray casting (bottom), with 1024x768 pixels in 16.63s.

7. Acknowledgements

This work was partially supported by FUNCAP/Brazil.

8. References

[1] Gottschalk, S., Lin, M.C. & Manocha, D. (1996A), "OBBTree: A Hierarchical Structure for Rapid Interference Detection", *Proc. of SIGGRAPH '96*, pp. 171–180, 1996.

[2] Figueiredo, M., Marcelino, L. & Fernando, T. (2002), "A Survey on Collision Detection Techniques for Virtual Environments", *Proceedings of the 5th SVR*, Fortaleza, October 2002, pp. 295-317

[3] Maia, J. G. R., Cavalcante-Neto, J. B. & Vidal, C. A. (2003) "CRAbGE: Um MotorGráfico Customizável, Expansível e Portável Para Aplicações de Realidade Virtual", *Proceedings of the 6th SVR*, Ribeirão Preto, SP, Brazil, pp. 03-14, October 2003.

[4] Bergen, G. Van Den (1997), "Efficient Collision Detection of Complex Deformable Models using AABB Trees", *Journal of Graphics Tools*, 2, 4, pp. 1-13.

[5] Cohen, J., Lin. M. C., Manocha, D., & Ponamgi, M. (1995), "I-Collide: An interactive and exact collision detection system for large-scale environments", *Proceedings of ACM I3D*, pp. 189-196.

[6] Terdiman, P. (2002) "OPCODE User Manual", http://www.codercorner.com/Opcode.htm.

[7] Ehmann, S. A. & Lin, M. C. (2001), "Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition", *Proceedings of the ACM Eurographics*.

[8] Vidal, C. A., Gomes, G. A. M. (2004), "Uma Ferramenta de Autoria de Ambientes Virtuais Adaptável a Diferentes Motores Gráficos", *Proceedings of the 7th SVR*, São Paulo, pp. 212-224.

[9] Eberly, D. H. (2005), "Geometric Tools", http://www.geometrictools.com. Visited in October 2005.

[10] Schmidl, H., Walker N. & Lin, M. C. (2004), "AB: Fast Update of OBB Trees for Collision Detection Between Articulated Bodies", *Journal of Graphics Tools*, 9(2):1-9, 2004

[11] Hirota, G., Fisher, S. & Lin, M. C. (2000), "Simulation of non–penetrating elastic bodies using distance fields", *Technical Report TR00-018*, University of North Carolina, 2000.

[12] Smith, R. L. (2005), "Open Dynamics Engine", http://www.ode.org. Visited in November 2005.

[13] Teschner, M., Heidelberger, B., Müller, M., Pomeranets, D. & Gross, M. (2003), "Optimized Spatial Hashing for Collision Detection of Deformable Objects", *Proceedings of VMV'03*, pp. 47-54, 2003.

[14] Heidelberger, B., Teschner, M. & Gross, M. (2003), "Real–Time Volumetric Intersections of Deforming Objects", *Proceedings of VMV'03*, pp. 461-468, 2003.

[15] Govindaraju, N. K., Lin, M. C. and Manocha, D. (2005), "Quick-CULLIDE: Efficient Inter- and Intra-Object Collision Culling using Graphics Hardware", *Proc. of IEEE VR 2005*, ACM, Bonn, March, pp. 1-10.

[16] Gottschalk, S. (1996B), "Separating Axis Theorem", *Technical Report TR96-024*, Dept. of Computer Science, UNC Chapel Hill, 1996.

[17] Gilbert, E. G., Johnson, D. W. & Keerthi, S. S. (1988), "A fast procedure for computing the distance between complex objects in three-dimensional space", *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

[18] Rabbitz, R. (1994), "Fast Collision Detection of Moving Convex Polyhedra", *Graphics Gems IV*, pp. 83-109.

[19] Bergen, G. Van Den (1999), "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects", *Journal of Graphics Tools*, 2(4): pp. 1-13.

[20] OGRE3D (2005), "Object-oriented Graphics Rendering Engine", http://www.ogre3d.org. Visited in December 2005.

[21] Wu, K. (1994), "Fast inversion of Length-and Angle-Perserving Matrices", *Graphics Gems IV*, 1994, pp. 199-206.