Volume xxx, (1998) number yyy pp. 000–000

Rapid and Accurate Contact Determination between Spline Models using ShellTrees[†]

S. Krishnan[‡] M. Gopi M. Lin D. Manocha A. Pattekar

Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-3175

Abstract

In this paper, we present an efficient algorithm for contact determination between spline models. We make use of a new hierarchy, called ShellTree*, which comprises of* spherical shells*, and oriented bounding boxes. Each spherical shell corresponds to a portion of the volume between two concentric spheres. Given large spline models, our algorithm decomposes each surface into Bézier patches as part of pre-processing. At runtime it dynamically computes a tight fitting axis-aligned bounding box across each Bézier patch and efficiently checks all such boxes for overlap. Using off-line and on-line techniques for tree construction our algorithm computes ShellTrees for Bézier patches and performs fast overlap tests between them to detect collisions. The overall approach can trade off runtime performance for reduced memory requirements. We have implemented the algorithm and tested it on large models, each composed of hundred of patches. Its performance varies with the configurations of the objects. In most cases, it can accurately compute the contacts in about one millisecond.*

1. Introduction

Contact determination is an integral part of many graphics applications, including dynamic simulation, haptic rendering, virtual environments, computer animation and virtual prototyping. Given two models, the goal is to check for possible contacts between them as they move. These models may be represented using polygons, spline patches, implicitly defined functions or time-dependent functions.

Extensive literature on contact determination between polygonal models is available and several efficient algorithms have been proposed. In many graphics and CAD applications, the models are represented using splines surfaces, e.g. NURBS or Bézier patches. The simplest algorithms are based on tessellation of spline surfaces into polygons and then use collision detection algorithms for polygonal models. However, they can be slow and inaccurate in practice. There is considerable work on curve and surface intersections in geometric and solid modeling. Unfortunately, most of these algorithms only handle static models and focus on robust and accurate computation of all intersection components.

Main Contribution: In this paper, we present a rapid and accurate algorithm for contact determination between spline surfaces using a new hierarchy, *ShellTree*. A ShellTree is a hierarchical representation that consists of mainly higher-order bounding volumes, *spherical shells*, and *oriented bounding boxes* (OBBs) [12]. A spherical shell corresponds to a portion of the volume between two concentric spheres and encloses the underlying geometry. The principal advantage of using spherical shells comes from the fact that they provide local *cubic* convergence [21] to the underlying geometry. Therefore, there are fewer “false positives” in terms of overlap tests between the bounding volumes and the underlying primitives. This results in an improved overall performance for proximity queries between two objects in near-contact configurations or between two objects with high-curvature surfaces.

Given a large model composed of spline surfaces, the

[†] Supported in part by a Sloan fellowship, ARO Contract P-34982-MA, NSF grant CCR-9319957, NSF grant CCR-9625217, ONR Young Investigator Award and Intel.

[‡] Currently at AT & T Research Labs

© The Eurographics Association 1998. Published by Blackwell Publishers, 108 Cowley Road, Oxford OX4 1JF, UK and 238 Main Street, Cambridge, MA 02142, USA.

algorithm decomposes each surface into Bézier patches as part of pre-processing. The contact determination algorithm consists of two phases. In the first phase, it dynamically computes a tight fitting axis-aligned bounding box across each Bézier patch and makes use of sweep and prune technique [7] to check all bounding boxes for overlap. In the second phase, it considers pairs of Bézier patches whose bounding boxes overlap and verify their contact status using ShellTrees. Using off-line and on-line techniques for tree construction and lazy evaluation to reduce the memory requirements, our algorithm computes a hierarchical representation of each Bézier patch using ShellTrees. We also present efficient algorithms for overlap test between two spherical shells. We have implemented the algorithm and demonstrate its performance on a chain of tori undergoing rigid motion and on large models composed of hundreds of patches. In most cases, the contact determination algorithm takes anywhere between 0.5-1.5 milliseconds depending on the complexity of the simulation.

Organization: The rest of the paper is organized in the following manner. We briefly survey related work in Section 2 and present an algorithm to compute spherical shells for a Bézier patch in Section 3. Then, we present algorithms for fast overlap tests between spherical shells in Section 4 and use them to formulate a fast contact determination algorithm between models composed of hundreds or thousands of patches in Section 5. Section 6 gives a performance comparison between the spherical shells and other higher order bounding volumes. We describe the implementation and performance of our algorithm on different models in Section 7. Finally, we conclude in Section 8.

2. Related Work

The problem of contact determination has been extensively studied in computer graphics, computational geometry and robotics. Most of the earlier work had focussed on collision detection between convex polytopes. These include algorithms based on linear programming [37], Minkowski differences and convex optimization [6, 11] and tracking closest features based on external Voronoi regions [23]. For general polygonal models many algorithms based on hierarchical representations have been proposed [4, 5, 12, 16, 3, 17, 28, 29, 30]. It is possible to tessellate a spline model into polygons and apply one of these algorithms for polygonal models. However, the resulting algorithm can be slow and inaccurate. The accuracy of the overall approach would be limited by the polygonal approximation of the tessellation algorithm. For large models, its memory requirements will also be quite high.

Most environments consist of multiple objects. Performing $O(n^2)$ pairwise interference detection becomes a bottleneck for large n. Algorithms of complexity $O(nlog^2 n+m)$ have been presented for spheres in [15] and rectangular bounding boxes in [9], where m corresponds to the number of overlaps. More recently, Cohen et al. have presented algorithms and a system, I-COLLIDE, based on spatial and temporal coherence, for large environments composed of multiple moving objects [7].

The set of algorithms for curved or spline models can be divided into static models, dynamic objects undergoing rigid motion and models whose trajectories can be expressed as a closed form function of time. These is considerable literature in geometric and solid modeling on fast intersection computations between spline curves and surfaces. Algorithms for curve intersections are based on algebraic methods [26, 35], subdivision [22], interval arithmetic [18], Bézier clipping [33], fat arcs [36] and algebraic pruning [27]. To some degree, our spherical shell is a 3D extension of *fat arcs*, a bounding shape with cubic convergence for 2D curves [36]. All these algorithms accurately compute intersections between curves and can be used for contact determination. A number of approaches have been proposed for surface intersection. These are based on subdivision methods [22], algebraic methods [25, 19], robust detection of all the intersection components and tracing [1, 14, 34], interval arithmetic [38] and lattice evaluation methods [31]. The goal of these methods is to accurately compute the topology of the intersection curve and therefore, they can be slow for interactive applications.

As for dynamic objects undergoing rigid motion, algorithms based on algebraic formulation have been proposed by Lin and Manocha [24]. They utilize coherence between successive frames and work well for low degree models. Baraff [2] has presented algorithms using coherence for convex curved models. If the motions of the object can be expressed as a closed form function of time, many algorithms based on subdivision methods, interval arithmetic or bounds on derivatives and constrained minimization have been proposed [8, 13, 10]. However, they are fairly slow in practice and not appropriate for real-time applications. Despite the wealth of the literature in the area, solving contact determination between spline models rapidly and accurately remains a research challenge.

3. Spherical Shell Computation

In this section, we define a shell and present methods to compute shells for Bezier patches. Consider two concentric spheres. Let V_1 be the portion of space outside the smaller sphere but inside the larger sphere. Now, consider a single-sided cone whose apex coin-

© The Eurographics Association 1998

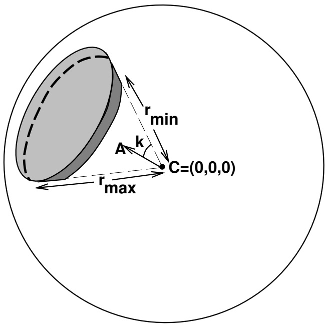

Figure 1: *Spherical shell*

cides with the common center of the two spheres. Let V_2 be the collection of points in the interior of the cone. Then, the volume in the intersection of volumes V_1 and V_2 is called a *spherical shell.* Without loss of generality, we assume that the common center of the two spheres and the apex of the cone lie at the origin. Then, mathematically we can define a shell as follows: Let $r_{\max}$ and $r_{\min}$ be the radii of the larger and smaller spheres respectively. Let $\mathbf{A} = (\mathbf{A_x}, \mathbf{A_y}, \mathbf{A_z})$ be a unit vector representing the axis of the cone. Let α be the half-angle of the cone. Then, the shell is defined as a set of all points (x, y, z) satisfying:

$$r_{\min}^2 \leq x^2 + y^2 + z^2 \leq r_{\max}^2$$
$$\frac{xA_x + yA_y + zA_z}{\sqrt{x^2+y^2+z^2}} \geq \cos\alpha \qquad (1)$$

Figure 1 illustrates a spherical shell.

Thus, a shell is completely determined by the following parameters:

- $C(x, y, z)$, the center,
- $r_{\min}$ and $r_{\max}$, the radii,
- $\mathbf{A}$, a unit vector representing the axis,
- k, the cosine of the half-angle.

Next, we describe an algorithm to compute a shell for a Bezier patch. Most of the algorithms presented in this section are the non-trivial three-dimensional extension of the fat arc construction proposed by Sederberg et. al. [36]. One of the important considerations in shell computation is its time complexity. At the heart of this issue is the trade-off between speed of collision detection and memory usage by the system. We would like to fit individual Bezier patches with spherical shells as quickly as possible so that we can save on memory usage without paying a significant penalty on time. This increases the bounds on the size of models that we can handle in our collision system.

3.1. Determining the center

The center of the shell is chosen as the center of a good spherical approximation to the given Bezier patch. In our case, we choose the four corner control points of the patch (they lie on the patch) and find the sphere that passes through them. It is quite obvious that this procedure does not produce the best spherical approximation to the patch. A much better approximation would be the least squares fit sphere for a denser sampling of points on the input patch. Known solutions to the least squares problems (like Singular Value Decomposition) are quadratic in the number of sampled points. This is quite expensive for building shells dynamically for detecting collisions.

The general equation of a sphere with center (c_x, c_y, c_z) and radius r is $(x-c_x)^2 + (y-c_y)^2 + (z-c_z)^2 - r^2 = 0$. Substituting $\frac{r^2-c_x^2-c_y^2-c_z^2}{2}$ by d and rearranging the above equation, we get

$$c_x x + c_y y + c_z z + d = \frac{x^2+y^2+z^2}{2}$$

In our case, c_x, c_y, c_z and d are unknowns. Each of 4 points (x_i, y_i, z_i) $(i = 1, 2, \ldots, 4)$ forms a row of a *linear system.* Solving this linear system gives us the vector (c_x, c_y, c_z, d). It is trivial to determine the center $\mathbf{C}$ and radius of the sphere from this vector.

We choose the center of this sphere to be the center of the shell. It is difficult to justify the use of this approach to determine the center and (later) the radii of the shells. In our experience, given the higher-order convergence of this bounding volume, this approach results in fairly satisfactory spherical shells.

3.2. Computing $r_{\min}$ and $r_{\max}$

The inner and outer radii of the spherical shell are determined by computing the range of distances $D(s, t)$ between the center $\mathbf{C}$ and the Bezier patch $\mathbf{F}(s, t)$.

A rational Bezier patch, $\mathbf{F}(s, t)$, of degree $m \times n$, defined in the domain $(s, t) \in [0, 1] \times [0, 1]$ and specified by a two dimensional array of control points $(\mathbf{v_{ij}}, w_{ij})$ is given by:

$$\mathbf{F}(s,t) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} \mathbf{v_{ij}} \mathcal{B}_i^m(s)\mathcal{B}_j^n(t)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij} \mathcal{B}_i^m(s)\mathcal{B}_j^n(t)}. \qquad (2)$$

$\mathcal{B}$ is the Bernstein basis function defined as

$$\mathcal{B}_i^m(s) = \begin{pmatrix} m \\ i \end{pmatrix} s^i (1-s)^{m-i}$$

© The Eurographics Association 1998

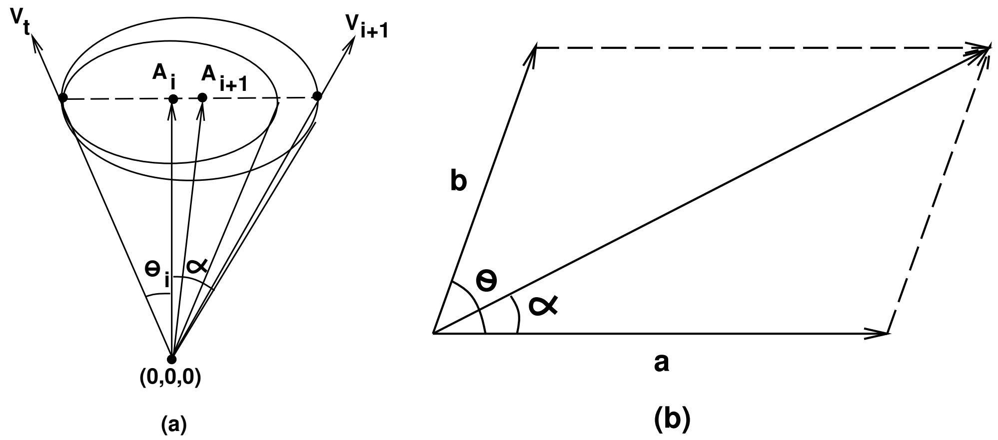

Figure 2: (a) Incremental computation of shell angle and axis vector (b) Illustration of vector addition by parallelogram rule

The distance function $D(s,t)$ is given by

$$D(s,t) = \sqrt{D^2(s,t)} = \sqrt{(\mathbf{F}(s,t) - \mathbf{C}) \cdot (\mathbf{F}(s,t) - \mathbf{C})}$$

The function $(\mathbf{F}(s,t) - \mathbf{C})$ can be written in the Bernstein basis of the form

$$\frac{\sum_{i=0}^{m} \sum_{j=0}^{n} (\mathbf{v_{ij}} - w_{ij}\mathbf{C}) \mathcal{B}_i^m(s) \mathcal{B}_j^n(t)}{\sum_{i=0}^{m} \sum_{j=0}^{n} w_{ij} \mathcal{B}_i^m(s) \mathcal{B}_j^n(t)}$$

It is a well known fact that the product of two Bernstein polynomials can also be represented as one. We now derive this representation for $D^2(s,t)$.

$$D^2(s,t) = \frac{\sum_{i=0}^{m} \sum_{j=0}^{n} (\mathbf{v_{ij}} - w_{ij}\mathbf{C}) \mathcal{B}_i^m(s) \mathcal{B}_j^n(t)}{\sum_{i=0}^{m} \sum_{j=0}^{n} w_{ij} \mathcal{B}_i^m(s) \mathcal{B}_j^n(t)} \cdot \frac{\sum_{k=0}^{m} \sum_{l=0}^{n} (\mathbf{v_{kl}} - w_{kl}\mathbf{C}) \mathcal{B}_k^m(s) \mathcal{B}_l^n(t)}{\sum_{k=0}^{m} \sum_{l=0}^{n} w_{kl} \mathcal{B}_k^m(s) \mathcal{B}_l^n(t)}$$

$$= \sum_{p=0}^{2m} \sum_{q=0}^{2n} U_{pq} \mathcal{B}_p^{2m}(s) \mathcal{B}_q^{2n}(t)$$

Equating the individual powers of this expression, we get

$$U_{pq} = \sum_{i=\max(0,p-m)}^{\min(m,p)} \sum_{j=\max(0,q-n)}^{\min(n,q)} u_{ij}$$

$$u_{ij} = \frac{(\mathbf{v_{ij}} - w_{ij}\mathbf{C}) \cdot (\mathbf{v_{p-i,q-j}} - w_{p-i,q-j}\mathbf{C})}{w_{ij} w_{p-i,q-j}}$$

If the Bezier patch is of degree $m \times n$, the distance squared function is a Bernstein polynomial of degree $2m \times 2n$. Because of the convex hull property of Bernstein polynomials, bounds on $D^2(s,t)$ are within the minimum and maximum of the U_{pq}'s. The minimum and maximum values of U_{pq} provide the r_{min} and r_{max} of the shell.

3.3. Computing the axis vector and angle

In this section, we present an algorithm to compute the axis and angle of the shell that bounds the entire Bezier patch. The algorithm presented is incremental in nature and is repeated as many times as the number of control points of the patch. This algorithm can be used generally to find a bounding cone for a set of vectors. In our case, these vectors are emanating from the center of the shell and points to each control point of the patch. For simplicity, we assume that the center is at the origin. The bounding cone computation is very similar to the one presented by Sederberg et. al. [32]. However, the result published there has a minor error in it. We have taken the liberty of correcting it here. Figure 2(a) is also taken from [32]. The bounding cone obtained from this algorithm is not the optimal one, but it is near-optimal and linear in time complexity.

Let us assume that after i iterations of the algorithm, our unit axis vector is $\mathbf{A_i}$ with shell angle θ_i. Let $\hat{\mathbf{v}}$ represent the normalized form of a vector $\mathbf{v}$. Given a set of n vectors $\mathbf{V_1}, \mathbf{V_2}, \ldots, \mathbf{V_n}$, we initialize $\mathbf{A_1}$ to $\hat{\mathbf{V_1}}$ and θ_1 to 0. Each subsequent vector $\mathbf{V_{i+1}}$ is checked to see if it lies within the cone. The vector lies within the cone if $\cos\theta_i \leq \hat{\mathbf{V_{i+1}}} \cdot \mathbf{A_i}$. If it lies within the cone, $\mathbf{A_{i+1}} = \mathbf{A_i}$ and $\theta_{i+1} = \theta_i$. Otherwise, a new cone is formed which is the smallest cone containing the old cone and the new vector. Let the angle between the vectors $\hat{\mathbf{V_{i+1}}}$ and $\mathbf{A_i}$ be α as shown in Figure 2(a). Consider a vector $\mathbf{V_t}$ that lies in the same plane as $\mathbf{A_i}$ and $\hat{\mathbf{V_{i+1}}}$ and makes an angle θ_i

© The Eurographics Association 1998

with $\mathbf{A_i}$. Clearly, this vector lies on the previous cone. We want to determine this vector in terms of $\mathbf{A_i}$ and $\hat{\mathbf{V_{i+1}}}$. From the Figure 2(a), it is clear that $\mathbf{V_t}$ can be expressed as some linear combination of $\mathbf{A_i}$ and $-\hat{\mathbf{V_{i+1}}}$ (*i.e.,* $\mathbf{V_t} = q\mathbf{A_i} - \hat{\mathbf{V_{i+1}}}$). To determine q, we use the parallelogram law for the sum of two vectors. It is shown in Figure 2(b).

Consider two vectors $\mathbf{a}$ and $\mathbf{b}$ at an angle θ. Let the vector $\mathbf{a}+\mathbf{b}$ form an angle α with $\mathbf{a}$. From the triangle law, we know that

$$\frac{\| \mathbf{a} \|}{\| \mathbf{b} \|} = \frac{\sin(\theta - \alpha)}{\sin \alpha}$$

In our case, q, which is the ratio of the lengths of $\mathbf{A_i}$ and $-\hat{\mathbf{V_{i+1}}}$, is given by

$$q = \frac{\sin(\theta_i + \alpha)}{\sin \theta_i}$$

Given $\mathbf{V_t}$, we compute the new axis of the cone

$$\mathbf{A_{i+1}} = \frac{\hat{\mathbf{V}}_\mathbf{t} + \hat{\mathbf{V_{i+1}}}}{\| \hat{\mathbf{V}}_\mathbf{t} + \hat{\mathbf{V_{i+1}}} \|}$$

and the angle is given by $\cos\theta_{i+1} = \mathbf{A_{i+1}} \cdot \hat{\mathbf{V_{i+1}}}$.

$\mathbf{A_n}$ and θ_n give the axis and the angle of the cone bounding all the original vectors. For Bezier patches, n is the number of control points of the patch.

The complexity of fitting a shell to a Bezier patch as described here is determined by the complexity of each step - determining center, minimum radius, maximum radius, shell axis and shell angle. Determining the center involves the solution of a 4×4 linear system. It is a constant time operation. The minimum and maximum radius determination step is roughly quadratic in the number of control points. The complexity of obtaining the shell angle and axis is linear in the number of control points. Overall, our algorithm is quadratic in the number of control points. The typical degree of the patches involved in most model descriptions are bicubic (16 control points). Our experiments suggest that the average operation count to build a shell for a Bezier patch of degree 2×3 is about 150-180.

4. Overlap Tests between Shells

In this section, we present the overlap test between two shells. We have included the overlap test in another manuscript that has been sent for publication [21]. We include it here for the sake of completeness. Before we start, some notation has to be introduced. Without loss of generality, the two shells S_1 and S_2 are determined by the following parameters.

- S_1
  - **center:** $C_1 = (0,0,0)$
  - **axis vector:** (a,b,c)
  - **cosine of half angle:** k_1
  - **radii:** $[r_{1n}, r_{1x}]$
- S_2
  - **center:** $C_2 = (0,0,d)$
  - **axis vector:** (l,m,n)
  - **cosine of half angle:** k_2
  - **radii:** $[r_{2n}, r_{2x}]$

The overlap test proceeds in three main stages.

- Check for trivial rejection scenarios. For example, we can surely reject the shells if the distance between their centers is greater than the sum of their outer radii.
- Then we test if one of the extremal radii (r_{1n} or r_{1x}) is involved in the intersection with the other shell (S_2). This procedure contains a number of early acceptance exits. For example, it is easy to check if the extremal radius of both shells are intersecting.
- Finally, we check if the conical portions of the shells are the only intersecting portions between the shells.

In our implementation, the tests for trivial acceptance and rejection scenarios significantly contribute to the speed of the overlap test.

We start by answering a simpler test. Let us define the set of points in the shell volume that are at a constant distance r from the center as the *shell restricted to* r. A simpler test is to see if the shell S_1 restricted to r_1 intersects shell S_2 restricted to r_2. Given the above parameters, the two spheres intersect along the plane $z = D'$, where $D' = \frac{d^2+r_1^2-r_2^2}{2d}$. Any intersection of the two restricted shells can only occur on this plane. Let us now write the algebraic constraints for the two restricted shells. The first of the two equations is the sphere equation and the second is the constraint for it to lie within the cone. This is obtained by taking a simple dot product. For S_1,

$$\begin{aligned} x^2 + y^2 + z^2 &= r_1^2 \\ ax + by + cz &\geq r_1 k_1 \end{aligned} \tag{3}$$

and for S_2,

$$\begin{aligned} x^2 + y^2 + (z-d)^2 &= r_2^2 \\ lx + my + n(z-d) &\geq r_2 k_2 \end{aligned} \tag{4}$$

Subtracting the two sphere equations, we get $z = D'$. The equation of the intersecting circle becomes $x^2 + y^2 = r_1^2 - D'^2$. Therefore, the two restricted shells intersect if there is a point (x, y, D') that satisfies

© The Eurographics Association 1998

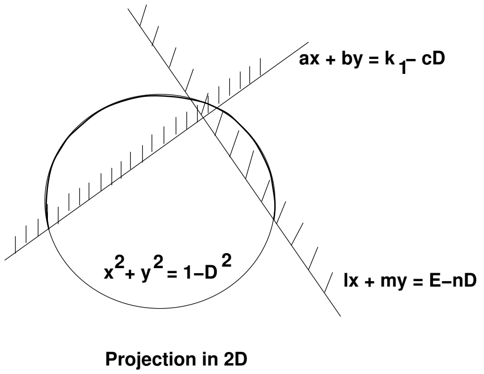

Figure 3: Reduction of overlap test to the plane

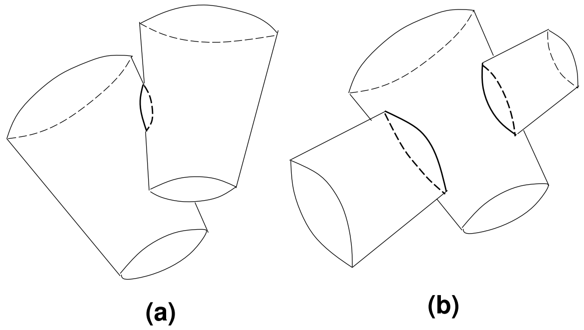

Figure 4: Orientation of shells that do not involve extreme radii

$$\begin{aligned} x^2 + y^2 &= r_1^2 - D'^2 \\ ax + by &\geq r_1 k_1 - cD' \\ lx + my &\geq r_2 k_2 + nd - nD' \end{aligned} \quad (5)$$

Scaling all the equations in (5) by a factor $\frac{1}{r_1}$ and replacing D'/r_1 by D and $(r_2k_2 + nd)/r_1$ by E, we get

$$\begin{aligned} x^2 + y^2 &= 1 - D^2 \\ ax + by &\geq k_1 - cD \\ lx + my &\geq E - nD \end{aligned} \quad (6)$$

Essentially, the test boils down to checking if there exists some point on the circle which lies in the positive half-space of both lines (see Figure 3). To check for satisfaction of (6), let us parameterize x and y in the circle by

$$x = \sqrt{1 - D^2}\frac{1 - t^2}{1 + t^2}, \quad y = \sqrt{1 - D^2}\frac{2t}{1 + t^2}$$

Substituting this in the other two inequalities and simplifying, we get

$$\left(\frac{k_1 - cD}{\sqrt{1 - D^2}} + a\right)t^2 - 2bt + \left(\frac{k_1 - cD}{\sqrt{1 - D^2}} - a\right) \leq 0 \quad (7)$$

and

$$\left(\frac{E - nD}{\sqrt{1 - D^2}} + l\right)t^2 - 2mt + \left(\frac{E - nD}{\sqrt{1 - D^2}} - l\right) \leq 0 \quad (8)$$

This test can be performed easily by computing the roots of each of these equations (if they exist in the real domain) and checking if the appropriate intervals where the signs of the equations are non-positive have an intersection. However, solving the quadratic equations involves a couple of square roots. Square root is a relatively expensive operation as compared to other arithmetic operations like additions and multiplications. If we can avoid square roots in our computation, it would speed up our overlap test significantly. It is fairly simple to still carry out the test without computing their actual roots. We shall now provide the details of this test.

Given two quadratic equations $(a_2x^2 + a_1x + a_0, b_2x^2 + b_1x + b_0)$, we want to check if there exists some value of x for which both the quadratic equations have non-positive values.

For the rest of this discussion, we assume that the quadratic equations have real roots and their leading coefficients (a_2 and b_2) are positive. This makes the description much simpler. Our test is easily extended to handle cases when either or both of the equations have only complex roots. Let us assume that the roots of the equations are (x_1, x_2) and (y_1, y_2) respectively. If the roots of the two quadratics interleave each other (for example, $x_1 < y_1 < x_2 < y_2$), then they must have some regions where both equations are non-positive. This means that $(a_2y_1^2 + a_1y_1 + a_0)(a_2y_2^2 + a_1y_2 + a_0) < 0$. Expanding the above inequality and rearranging the terms, we get

$$\begin{aligned} a_2^2(y_1y_2)^2 + a_2a_1(y_1y_2)(y_1 + y_2) + a_2a_0((y_1 + y_2)^2 - 2y_1y_2) \\ + a_1^2(y_1y_2) + a_1a_0(y_1 + y_2) + a_0^2 < 0. \quad (9) \end{aligned}$$

We know that $y_1 + y_2 = \frac{-b_1}{b_2}$ and $y_1y_2 = \frac{b_0}{b_2}$. This gives the inequality

$$\begin{aligned} a_2^2b_0^2 - a_2a_1b_1b_0 + a_2a_0b_1^2 - 2a_2a_0b_0b_2 \\ + a_1^2b_0b_2 - a_1a_0b_1b_2 + a_0^2b_2^2 < 0. \quad (10) \end{aligned}$$

© The Eurographics Association 1998

If this inequality is satisfied, the test is done. However, it is possible that the roots of one equation lie completely inside the interval determined by the roots of the other equation. We have assumed that the leading coefficients of the two quadratics are positive. That means that the only turning point of both equations are minima. Let the turning point of the equations be denoted $t_1 = \frac{-a_1}{2a_2}$ and $t_2 = \frac{-b_1}{2b_2}$. Then the equations satisfy the original condition if either one of these turning points lie within the roots of the other equation. That is, if one of

$$(t_1 - y_1)(y_2 - t_1) = -t_1^2 + t_1\frac{-b_1}{b_2} - \frac{b_0}{b_2} > 0, or$$

$$(t_2 - x_1)(x_2 - t_2) = -t_2^2 + t_2\frac{-a_1}{a_2} - \frac{a_0}{a_2} > 0$$

is satisfied, we declare a satisfactory test.

Even though the coefficients of our quadratic equations ((7) and (8)) have square roots, the expressions we actually compute in the tests do not involve square roots.

The discussion so far has concentrated on the case when both shells are restricted (r_1 and r_2 are fixed). However, in our case, r_1 and r_2 range from $[r_{1n}, r_{1x}]$ and $[r_{2n}, r_{2x}]$ respectively. This introduces two variables to the system of constraints and makes the test a lot more difficult. We can reduce the number of variables in our system to one if we observe that when two shells overlap, at least one of the extreme radii (r_{1n}, r_{1x}, r_{2n} or r_{2x}) of the shells is also involved except for the cases shown in Figure 4. We will come back to these two cases later. So our test for overlap between two shells reduces to four overlap tests between a shell restricted to one of its extreme radii and another shell. The tests described above carries through to this case as well. However, the signed expression we are evaluating changes to a (quartic) polynomial in a given range. We use Sturm sequences [39] to evaluate the signs of these polynomials without actually performing root computation. We omit many of the details here. But a detailed version of the test can be found in [20].

We shall now discuss the cases when one of the extreme radii are not involved in the intersection. We will give the details of how to perform the test using the case shown in Figure 4(a). The other case is handled similarly. Figure 4(a) shows two cones that just penetrate each other and intersect in a loop. From the surface intersection literature, it is well known that for two surfaces to intersect in a loop, there must be (at least) one collinear normal between them [32]. For the special case of cones, we know that every normal vector when extended will intersect the axis vector. So if

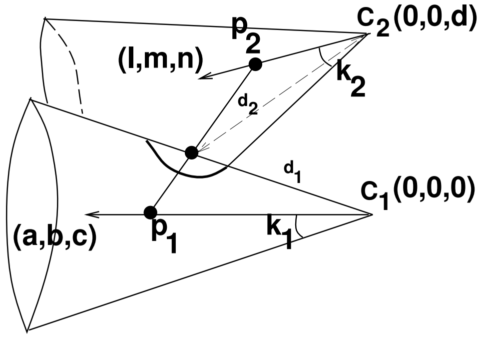

Figure 5: Cone/Cone intersection

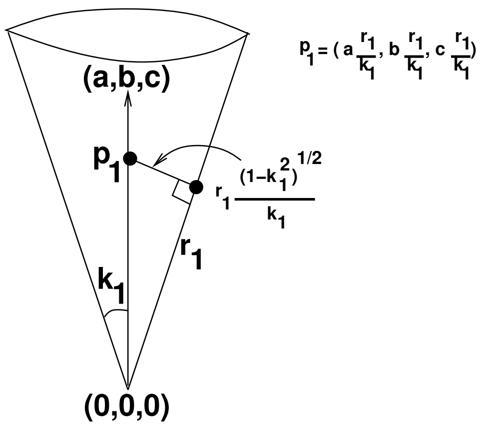

Figure 6: Normal to a cone parameterized by distance to the center

there exists a collinear normal, it will pass through the axis of each of the cones. In Figure 5, p_1p_2 is a line segment passing through the axis of the two cones which are normal to the respective cones at the point where they intersect them. Let these points be q_1 and q_2 respectively. Let the distance between C_1 (C_2) and q_1 (q_2) be d_1 (d_2).

Let us now compute the coordinates of the points p_1 and p_2. It is easy to see that $p_1 = (a\frac{d_1}{k_1}, b\frac{d_1}{k_1}, c\frac{d_1}{k_1})$ (see Figure 6) and $p_2 = (l\frac{d_2}{k_2}, m\frac{d_2}{k_2}, n\frac{d_2}{k_2} + d)$. If we denote the distance between p_1 and p_2 as D, then the unit vector along the common collinear normal (denoted by $\vec{n}$) is

$$\frac{1}{D}(l\frac{d_2}{k_2} - a\frac{d_1}{k_1}, m\frac{d_2}{k_2} - b\frac{d_1}{k_1}, n\frac{d_2}{k_2} - c\frac{d_1}{k_1} + d)$$

$\vec{n}$ is a collinear normal if it satisfies the following two constraints.

© The Eurographics Association 1998

$$\vec{n} \bullet (a,b,c) = -(1-k_1^2)^{1/2} (\text{angle between them is } (90+\theta))$$
$$\vec{n} \bullet (l,m,n) = (1-k_2^2)^{1/2} (\text{angle between them is } (90-\theta))$$

$\bullet$ is the dot product. Rearranging the equations so that D appears on the r.h.s, we get

$$\frac{1}{k_1(1-k_1^2)^{1/2}} d_1 - \frac{al+bm+cn}{k_2(1-k_1^2)^{1/2}} d_2 - \frac{cd}{(1-k_1^2)^{1/2}} = D \quad (11)$$

and

$$\frac{1}{k_2(1-k_2^2)^{1/2}} d_2 - \frac{al+bm+cn}{k_1(1-k_2^2)^{1/2}} d_1 + \frac{nd}{(1-k_2^2)^{1/2}} = D \quad (12)$$

Subtracting equation (12) and (11), replacing $al + bm + cn$ with K and rearranging, we get

$$d_1 = \frac{k_1((1-k_1^2)^{1/2} + K(1-k_2^2)^{1/2})}{k_2((1-k_2^2)^{1/2} + K(1-k_1^2)^{1/2})} d_2 + k_1 \frac{n(1-k_1^2)^{1/2} + c(1-k_2^2)^{1/2}}{(1-k_2^2)^{1/2} + K(1-k_1^2)^{1/2}} d \quad (13)$$

Equation (13) can be written as $d_1 = \alpha d_2 + \beta$. Substituting this relation back into equation (11) and squaring both sides (to prevent square roots), we get a quadratic equation in d_2 ($Ad_2^2 + Bd_2 + C = 0$). It is easy to check if there are valid solutions to this equation within our allowed range of d_2. If not, there is no overlap of the loop type.

Let us now assume that there are valid solutions of the (d_1, d_2) pair that satisfy the above equations. This means that there are points where collinear normals exist. To check if this corresponds to an intersection, all we have to check is if

$$\| p_1q_1 \|_2 + \| q_2p_2 \|_2 \geq \| p_1p_2 \|_2$$

From Figure 6 this relation is

$$\frac{(1-k_1^2)^{1/2}}{k_1} d_1 + \frac{(1-k_2^2)^{1/2}}{k_2} d_2 \ \geq \ D$$

Squaring both sides and substituting the relation between d_1 and d_2 from equation (13), we get a condition of the form

$$Gd_2^2 + Hd_2 + I \geq 0$$

It is easy to check if this condition is satisfied by the solutions already obtained. This entire test can be achieved without any square roots using Sturm sequences.

The overlap test may involve one square root computation (to compute distance between the centers of the shells) and up to 700 arithmetic operations, in the worst case. However, in a number of cases, the algorithm performs early rejects or early accepts. We ran a simulation of generating randomly positioned and oriented shells with limits on the distance between their centers (to prevent too many simple tests). In this simulation, our overlap test takes $300 - 400$ arithmetic operations on an average.

5. Contact Determination Using ShellTrees

In this section, we present an efficient and accurate contact determination algorithm between spline models using ShellTrees. A *ShellTree* is a hierarchical representation that consists of spherical shells and oriented bounding boxes (OBBs). By using off-line and on-line techniques for tree construction and lazy evaluation to reduce the memory requirements, our algorithm computes a hierarchical representation of each Bézier patch using ShellTrees. Next, we give an overview of the contact determination algorithm.

5.1. Algorithm Overview

Given a large model composed of spline surfaces, the algorithm decomposes each surface into Bézier patches as part of offline pre-computation. The algorithm computes a three-dimensional tight fitting axis-aligned bounding boxes (AABB) and a ShellTree for each Bézier patch. To reduce the memory requirement, it only constructs a static, fixed-height ShellTree for each patches using mostly spherical shells. Each spherical shell is computed using the methods outlined in Section 3. The height of the static ShellTree is specified by the user, as it depends on the applications and the contact configuration.

The online computation of our contact determination algorithm consists of two phases. In the first phase, it dynamically updates the AABBs across each Bézier patch. Then, it sorts these bounding volumes in 3-space to determine which pairs of patches are overlapping. Exploiting the temporal and spatial coherence, it utilizes sweep and prune technique[7] to find all potential pairs of overlapping patches in nearly linear time. In the second phase, it considers pairs of Bézier patches whose bounding boxes overlap and verifies their contact status using the static ShellTrees. First, the algorithm goes down from the parent nodes of the trees to check for possible intersections between shells using the overlap tests explained in section 4. As it traverses down the trees, if overlaps continue

© The Eurographics Association 1998

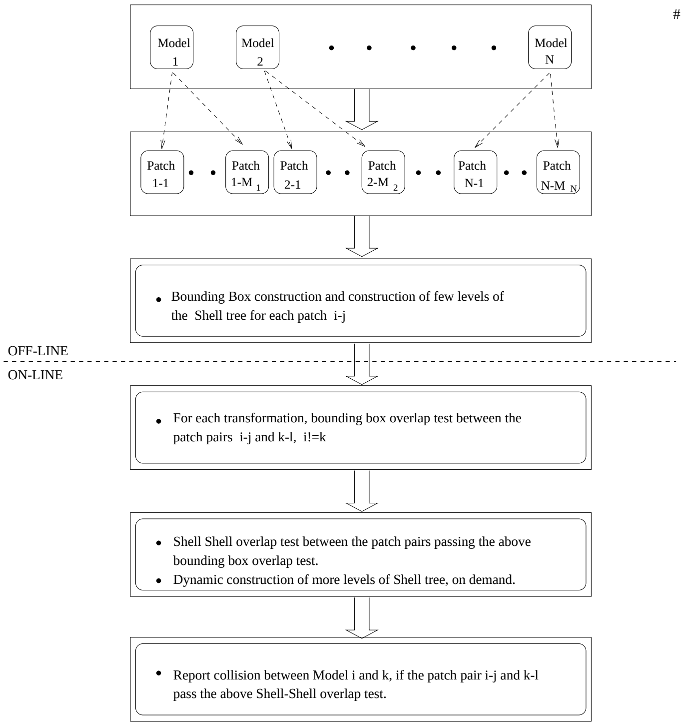

Figure 7: Overall Algorithm for Contact Determination

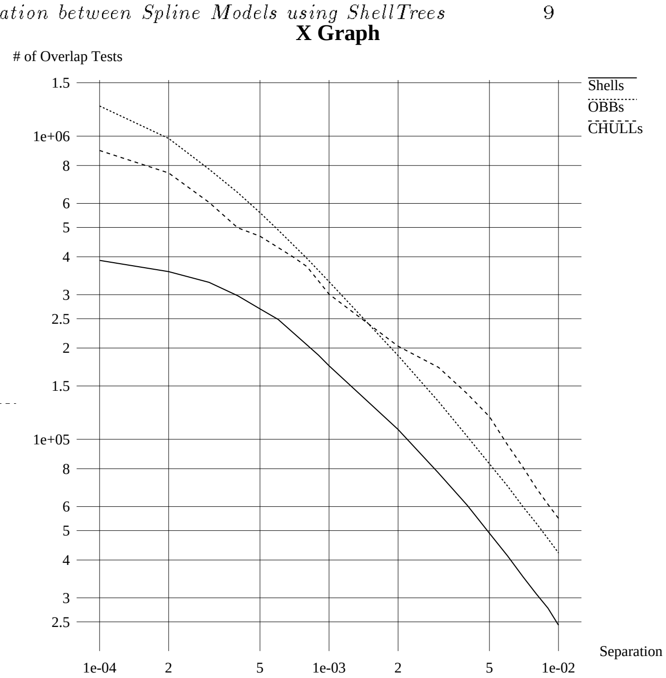

Figure 8: OBBs, Convex hulls and Shells for parallel close proximity

to occur between two nodes, more levels of ShellTrees are constructed as needed to report potential contacts. Otherwise, the algorithm terminates and no collision is reported.

A flow diagram that illustrates the entire process, that consists of both offline and online computations, is given in Fig. 7.

6. Comparison with Other Approaches

In this section, we discuss the convergence of our approach with some of the previous approaches used for collision detection. Since, most of the other methods are not readily applicable to spline models, we decided to use triangulated representations of geometry to perform our comparison. The model we have chosen to study the convergence is that of *parallel close proximity*. To illustrate parallel close proximity, consider, two concentric spheres that are ϵ apart from each other. It is easy to see that as ϵ decreases, the number of overlap tests required to state that they do not intersect increases. The rate of increase is directly related to the convergence of the bounding volume used. In [21], we had argued theoretically that the spherical shell bounding volume exhibits *cubic* convergence. Other bounding volumes like convex hulls and Oriented Bounding Boxes (OBBs) have been shown to converge quadratically.

We performed some experiments to empirically verify these claims. In our simulations, we used two identical models, each of which consisted of two patches from the side of the Utah teapot. The patches have slowly varying curvature and are convex everywhere. Each of these models consisted of 50,176 triangles. In each frame of the simulation, we displaced one model slightly relative to the other and performed a collision query. The simulation consisted of several such frames, each with a different displacement between the models. For each collision query, we counted the number of overlap tests required. Collision queries were performed using ShellTrees, OBB trees and Convex hull trees. The results obtained have been summarized in Table 1.

Figure 8 shows a log-log plot of the number of collision queries required for the three types of hierarchies as a function of the displacement between models. It is clear that the curve for OBBs and convex hulls show similar behavior. The differing slopes of these

© The Eurographics Association 1998

S.No.	Displacement	Shells	OBBs	Convex Hulls
1	0.01	24,395	42,207	54,903
2	0.008	30,925	52,889	69,085
3	0.006	41,257	70,091	95,823
4	0.004	60,607	102,023	141,383
5	0.002	107,739	188,887	202,891
6	0.001	174,785	331,017	301,103
7	0.0008	205,027	395,527	371,547
8	0.0006	248,607	490,057	429,149
9	0.0004	297,677	651,353	498,625
10	0.0002	357,333	982,415	755,973
11	0.0001	389,175	1,256,949	896,581

Table 1: *Overlap tests required for different displacements*

curves with respect to the shells imply that shells require asymptotically fewer overlap tests than OBBs, as a function of the displacement between the models in our experiment. As a result, for large models composed of thousands of polygons in close proximity positions, hierarchies based on spherical shells provide faster interference detection algorithms.

7. Implementation and Performance

An implementation of the algorithms detailed in this paper, is done in C++. This section describes the data structures, and the conceptual design of our system.

Each model is assumed to be a collection of untrimmed rational Bézier patches. The input to the system is n such models – $1 \ldots n$. The patches of the model i are numbered $(i, 1) \ldots (i, n_i)$. Each patch in the system is considered as an object.

7.1. Data structure

A ShellNode class consists of the control points of the patch P, the shell corresponding the patch and pointers to the left and right children. The patch P is subdivided into two, and the ShellNodes for the resultant patches P_l and P_r form the left and right children for the ShellNode of P. We call this hierarchical structure of ShellNodes as a ShellTree.

7.2. Pre-Processing

In the pre-processing stage, for each patch, a fixed-height ShellTree of ShellNodes is built statically. The height of the ShellTree is chosen by the user. This "static" tree will remain in the system till the end of a simulation. In our experimental studies, we varied the "static" height parameter to study the trade-off between speed of collision detection and memory usage in our system. We report these results later. Axis aligned bounding boxes for each patch is also constructed, using the control-point net. The *min* and *max* values of the bounding boxes for each of the three coordinate axes are sorted in linked lists.

The major computation costs in the pre-processing stage come from adding models to the system, construction of static fixed-height ShellTree and bounding box for each patch.

7.3. Runtime Computation

During the runtime when a model undergoes a rigid transformation, all its patches are transformed by the same transformation matrix and its bounding boxes are updated accordingly. Simultaneously the sorted linked list is also updated. With simple axis-aligned bounding box overlap tests, the list of all patch pairs (i, k)-(j, l), $i \neq j$, is created. Thus it is ensured that no unnecessary intersection test is performed between the patches of the same model.

This list is then given to the subroutine that performs Shell-Shell overlap test. The transformation matrices and the statically constructed ShellTree for each patch in the pair are used to call the shell-shell overlap test function. Given the transformation matrices, a shell in the original tree is transformed by updating its center and axis vector. This is an advantage in dynamic environments because the cost of updating the bounding volume is small. The overlap routine traverses both the trees and checks for overlaps between them recursively. If the algorithm is not able to answer the overlap test satisfactorily with the given tree, more levels of the subdivision tree are constructed dynamically and the overlap test is continued. The subdivision stops when the subdivided patch is less than a

© The Eurographics Association 1998

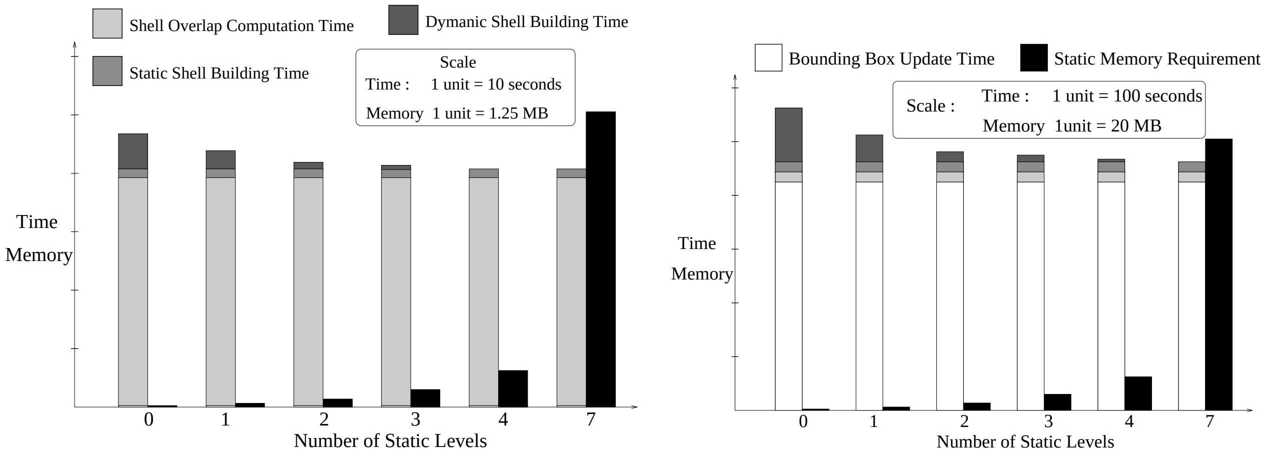

Figure 9: (a) Performance on Interlaced Torii (b) Performance on N-Body Collision

Number of Static Levels	0	1	2	3	4	7
Torii						
Static Shell Building Time(secs)	1.27	1.41	1.29	1.30	1.32	1.66
Dynamic Shell Building Time(secs)	6.1	3.2	1.5	0.72	0.33	0
NBody						
Static Shell Building Time(secs)	17.7	17.7	17.9	18.1	18.5	24.5
Dynamic Shell Building Time(secs)	105	53	26	12	6	0

Table 2: *Performance statistics of the collision system*

fraction of the original patch in its size. At this stage, it is triangulated into two triangles and they form the leaf nodes of the tree. When the tree traversal during the shell-shell overlap test reaches the leaf of the tree, triangle-shell or triangle-triangle overlap test is done. The dynamically constructed tree is deleted at the end of its usage, to reduce the runtime memory requirement.

If accurate overlap is detected in the above process, then the model pair whose patches overlap is entered in a list. During the collision reporting stage, this list is traversed and a report is created for the user.

The major computation costs during the runtime come from bounding box overlap test, dynamic tree construction, and the shell-shell overlap test.

7.4. Performance

We tested the performance of our system with two experiments. The first own known as the *interlaced tori* experiment consisted of a chain of five tori with nine tensor product bezier patches each. Each torus moves in a direction opposite to its adjacent tori. On collision with any one of the other torus, its direction of motion is reversed. We ran this experiment for 200 frames. The total number of collisions reported was 55. The average shell overlap test was found to be 38 microseconds. The total time spent for the transformation of patches and bounding box update was 0.07seconds. The total number of shell overlap tests was 1,044,707. The bar chart 9 shows the break up of the run-time costs.

The second experiment was an *N-body* experiment, in which ten goblets were set to motion in a cubical volume. Each goblet had 72 tensor product bezier patches, thus totaling 720 patches. The goblets were allowed to bounce off the walls and collide against each other. Simple collision response was modeled. This experiment was run for 200 frames and there was a total of 240 collisions reported. In this experiment the average overlap time, was calculated to be 31 microseconds. The total time spent for the whole simulation on

© The Eurographics Association 1998

transformation of patches and bounding box update took a constant time of 102 seconds. The total number of shell overlap tests was approximately 616,000. These two are independent of the height of the static tree chosen initially. The break up of run time costs is shown by the figure 9. The statistics of the above two experiments are given in the Table 2.

From the above two experiments, we found that, on an average, a shell-shell overlap test takes between 30 to 40 microseconds and, shell construction for one ShellNode takes 138 microseconds. All the timings presented here were measured on an SGI-Onyx with an R10000 processor, 194 MHz clock and 2 GB main memory.

The pattern of motion of the objects in the scene decides the time spent on bounding box updates. In the interlaced tori example, the motion of tori were very smooth, slow, and in only one dimension. On the other hand, in the n-body experiment, the motion of the goblets were very fast, and in all three dimensions. The use of frame-to-frame coherence in the bounding box update algorithm explains the difference in the corresponding times.

7.5. Memory Requirements

It is clear from the run time costs that, with the more levels of static shell building, the collision response time would decrease. On the other hand, the memory requirements of the system increases as these static trees stay with the system till the end. On an average, one node of the ShellTree requires about 116 bytes, on an average. For a h-level ShellTree, there are $(2^{h+1} - 1)$ ShellNodes. For our experiments, we restricted the number of levels of ShellTree to represent a patch to be 7. For a ShellTree of height 7, the static memory required is $(2^{7+1} - 1) * 116 = 29.52$KB. For the interlaced tori simulation, with 45 patches in the scene, the total memory required for all the ShellTrees would be, $45 * 29.52\text{KB} = 1.3MB$. The choice of number of static levels to be built in the pre-processing stage, depends on various factors like the size of the simulation in terms of total number of patches, the user's system memory availability and the time deadlines for the collision detection system. This choice is left to the user.

8. Conclusion

In this paper, we have introduced a new bounding volume hierarchy for detecting collisions called ShellTrees. We present efficient algorithms to build spherical shells for general spline models and for performing fast overlap tests between these shells. We compare its performance to other hierarchies and have shown that for many close proximity situations, the hierarchies based on spherical shells provide faster interference detection algorithms.

Our algorithm computes ShellTrees for Bézier patches and performs fast overlap tests between them to detect collisions. The overall approach can trade off runtime performance for reduced memory requirements. We have implemented the algorithm and tested them on large models composed of hundred of patches. In most cases, it can accurately compute the contacts in a few milliseconds.

References

1. C.L. Bajaj, C.M. Hoffmann, J.E.H. Hopcroft, and R.E. Lynch. Tracing surface intersections. *Computer Aided Geometric Design*, 5:285–307, 1988.

2. D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *ACM Computer Graphics*, 24(4):19–28, 1990.

3. G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. Boxtree: A hierarchical representation of surfaces in 3d. In *Proc. of Eurographics'96*, 1996.

4. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *Proc. SIGMOD Conf. on Management of Data*, pages 322–331, 1990.

5. S. Cameron. Approximation hierarchies and s-bounds. In *Proceedings. Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 129–137, Austin, TX, 1991.

6. S. Cameron. Enhancing gjk: Computing minimum and penetration distance between convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, 1997.

7. J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, pages 189–196, 1995.

8. Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *ACM Computer Graphics*, 26(2):131–139, 1992.

9. H. Edelsbrunner. A new approach to rectangle intersections, part i. *Int. J. of Comput. Math*, 13:209–219, 1983.

10. J. Snyder et. al. Interval methods for multi-point collisions between time dependent curved surfaces. In *Proceedings of ACM Siggraph*, pages 321–334, 1993.

11. E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between objects in three-dimensional space. *IEEE J. Robotics and Automation*, vol RA-4:193–203, 1988.

© The Eurographics Association 1998

12. S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*, pages 171–180, 1996.

13. B. V. Herzen, A. H. Barr, and H. R. Zatz. Geometric collisions for time-dependent parametric surfaces. *Computer Graphics*, 24(4):39–48, 1990.

14. M.E. Hohmeyer. A surface intersection algorithm based on loop detection. *International Journal of Computational Geometry and Applications*, 1(4):473–490, 1991. Special issue on Solid Modeling.

15. J.E. Hopcroft, J.T. Schwartz, and M. Sharir. Efficient detection of intersections among spheres. *The International Journal of Robotics Research*, 2(4):77–80, 1983.

16. P. M. Hubbard. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993.

17. J. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. In *Siggraph'96 Visual Proceedings*, 1996.

18. P.A. Koparkar and S. P. Mudur. A new class of algorithms for the processing of parametric curves. *Computer-Aided Design*, 15(1):41–45, 1983.

19. S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on the lower dimensional formulation. *ACM Transactions on Graphics*, 16(1):74–106, 1997.

20. S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shells: A higher-order bounding volume for fast proximity queries. Technical report, Department of Computer Science, University of North Carolina, 1997.

21. S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shell: A higher order bounding volume for fast proximity queries. In *Submitted for Review to the Third International Workshop on Algorithmic Foundations of Robotics*, 1998.

22. J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):150–159, 1980.

23. M.C. Lin and John F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.

24. M.C. Lin and Dinesh Manocha. Fast interference detection between geometric models. *The Visual Computer*, 11(10):542–561, 1995.

25. D. Manocha and J.F. Canny. A new approach for surface intersection. *International Journal of Computational Geometry and Applications*, 1(4):491–516, 1991. Special issue on Solid Modeling.

26. D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves I: simple intersections. *ACM Transactions on Graphics*, 13(1):73–100, 1994.

27. D. Manocha and S. Krishnan. Algebraic pruning: A fast technique for curve and surface intersections. *Computer Aided Geometric Design*, 20:1–23, 1997.

28. B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yield polyhedral modeling results. In *Proc. of ACM Siggraph*, pages 115–124, 1990.

29. M. Ponamgi, D. Manocha, and M. Lin. Incremental algorithms for collision detection between general solid models. In *Proc. of ACM/Siggraph Symposium on Solid Modeling*, pages 293–304, 1995.

30. S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of International Conference on Robotics and Automation*, pages 3324–3329, 1994.

31. J.R. Rossignac and A.A.G. Requicha. Piecewise-circular curves for geometric modeling. *IBM Journal of Research and Development*, 31(3):296–313, 1987.

32. T.W. Sederberg and R.J. Meyers. Loop detection in surface patch intersections. *Computer Aided Geometric Design*, 5:161–171, 1988.

33. T.W. Sederberg and T. Nishita. Curve intersection using bézier clipping. *Computer-Aided Design*, 22:538–549, 1990.

34. T.W. Sederberg and T. Nishita. Geometric hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 8:97–114, 1991.

35. T.W. Sederberg and S.R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, 1986.

36. T.W. Sederberg, S. White, and A. Zundel. Fat arcs: A bounding region with cubic convergence. *Computer Aided Geometric Design*, 6:205–218, 1989.

37. R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

38. J. Snyder. Interval arithmetic for computer graphics. In *Proceedings of ACM Siggraph*, pages 121–130, 1992.

39. J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, 1993.

© The Eurographics Association 1998