Efficient and Accurate Interference Detection for Polynomial Deformation

Merlin Hughes  Christopher DiMattia  Ming C. Lin†  Dinesh Manocha
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
USA
{hughesm,dimattia,lin,manocha}@cs.unc.edu
http://www.cs.unc.edu/~geom/deform/index.html

Abstract
We present efficient and accurate algorithms for interference detection among objects undergoing polynomial deformation. The scope of our algorithms include physically-based models undergoing dynamic simulation subject to non-penetration constraints, variational models, deformable models used in soft object animation, geometric models including polygonal meshes, parametric surfaces such as Bézier patches and B-splines, and solid models defined by such surfaces. Our algorithms use axis-aligned bounding boxes and convex hulls of the objects to identify the object pairs in close vicinity. They use subdivision, convex hull properties and linear programming to perform surface intersection tests and loop intersection tests. Frame-to-frame coherence is utilized to achieve incremental computations. The resulting algorithms have been implemented and work well in practice. In particular, we are able to compute all contacts accurately and at interactive speeds for flexible bodies undergoing second-order polynomial deformations.

1 Introduction
Over the last few years a number of techniques have been developed for three-dimensional computer animation. These techniques produce animations directly from models, shape deformations and sets of equations specifying the dynamic behaviors of structures or machines. One major technique, soft object animation [WW92a], intimately links the model and the animation of the data representing the model. Soft object animation is a set of techniques for animating a model with a deformation that varies with time. Previous work on deformable object animation uses physically-based methods [TPBF87, TF88a, WW90, BW92], variational techniques [WW92b, TQ94], and local and global deformations applied directly to the geometric models. Such deformations include those applied to the vertices of the polygonal models, free-form deformations (FFD) [TS86], extended free-form deformations (EFFD), animated free-form deformations (AFFD) [Coq90], direct manipulation of free-form deformations [HHK92], local and global deformations [Baa84] and hierarchical B-spline deformations [FB88].

A fundamental problem in these formulations is efficient and accurate collision detection between the boundaries of the objects. This includes interference checking between the boundaries of two objects and self-collision detection. In most of these applications, the motion of the objects cannot be expressed as a closed function of time. In most cases collision detection has been handled in very simple ways, subject to geometrical optimization which exploits the simplicity of the situation. In more general frameworks involving complex deformations and wrinkling situations, collision detection has been considered very challenging and time consuming [VT94, VCT95]. The problem of collision detection is also an integral part of physically-based modeling, robot motion planning, as well as computer simulated environments [Bar90, BB88, Hah88, HBZ90, MW88, Pen90, PW90, Stu87].

We present efficient and accurate collision detection algorithms for objects undergoing polynomial deformation. The underlying geometric models are composed of polygonal meshes, parametric surfaces and solids defined by such meshes, Bézier patches and B-spline surfaces. No assumptions are made on the motion of the objects or on the polynomial deformation. As an object undergoes polynomial deformation, hierarchical B-spline deformation or free-form deformation, we compute the control points of the resulting boundary and check for self-collision or interference between the surfaces. Our algorithms proceed in a hierarchical manner. At the top of the hierarchy, they
compute axis-aligned bounding boxes and check for intersections in an incremental manner. After the bounding boxes overlap, the convex hull property of the control points, i.e. the entire curve or surface is contained in the convex hull of the control points, is used to quickly perform rejection tests. However, the computation of convex hulls at each time step can be expensive and requires $O(n \log n)$ time [PS85]. Therefore, we check for overlaps between convex hulls by reducing the problem to one of linear programming and make use of fast randomized algorithms of linear complexity [Sei90]. Results of these tests are then combined with subdivision methods, hierarchical sweep-and-prune and loop detection to check for interference between the boundary of two surfaces and singularity detection for self-collision detection. We utilize frame-to-frame coherence by making use of the geometric state at the previous time instance to perform incremental computations. The algorithms have been implemented and we demonstrate their performance on polygonal models undergoing second-order deformations based on the formulation in [WW90] (as shown in Plates 1 and 2). Our system can accurately compute all the contacts between the models at interactive rates.

The rest of the paper is organized in the following manner. We briefly survey the state of the art in interference checking and collision detection algorithms in Section 2. In Section 3, we pose the problems of collision detection between objects undergoing polynomial deformations as checking overlap between Bézier and B-spline surfaces. In section 4 we present efficient algorithms for detecting interference between such surfaces. In section 5 we describe the algorithm implementation, with performance and applications in Section 6.

2 Previous Work

The problem of collision detection has been extensively studied in the literature. The simplest algorithms for collision detection are based upon bounding volumes and spatial decomposition techniques. These bounding volumes work very well in performing “rejection tests” whenever two objects are far apart. However, once the two objects come close to each other, spatial decomposition techniques based on subdivision are used to solve the interference problem. Recursive subdivision is robust but computationally expensive, and it often requires substantially more memory. As a result, the overall algorithms are slow. There has been a considerable amount of literature on algorithms for animation and simulation environments with models composed of convex polytopes [MW88, GJK88]. More recently, in applications involving dynamic simulations and physical motion, geometric coherence has been utilized to devise algorithms based on local features [Bar90, LC91]. This has significantly improved the performance of collision detection algorithms in dynamic environments. The idea of coherence has been generalized to large environments composed of hundreds and thousands of objects by Cohen et. al. [CLMP95] and the resulting I-COLLIDE system works very well for objects undergoing rigid motion.

As for curved objects, Herzen et al [HBZ90] have described a general algorithm for time dependent parametric surfaces. It treats time as an extra dimension and also assumes bounds on derivatives. The algorithm uses subdivision techniques in the higher dimensional space and can therefore be slow. A similar method using interval arithmetic has been presented for collision detection between implicitly defined models in [Duf92]. Duff has extended it to dynamic environments as well. However, for commonly used spline surfaces, computing and representing the implicit representations is expensive [Ho89]. Pentland and Williams, [PW90], proposed using implicit functions to represent shape and the property of the “inside-outside” functions for collision detection. Besides its restriction to implicit surfaces, this algorithm has a drawback in terms of robustness as it uses point samples. A detailed explanation of these problems is given in [Duf92]. Algorithms based on interval arithmetic are useful, as they can handle a number of environments including objects undergoing deformable motion. However, their practical utility is not clear at the moment. They are currently restricted to objects whose motion can be expressed as a closed form function of time. As we mentioned above, this is typically not the case in most applications. Furthermore, they are slow in practice. In particular, algorithms presented in [HBZ90, Duf92] converge very slowly to the solution and can take order of minutes to find precise contact. This has been improved by [ea93] by combining interval methods with Newton iteration to improve the convergence. However, the resulting algorithm requires on the order of thousands of iterations of interval arithmetic to compute the contact between time dependent parametric models, a few tens of seconds on a high end workstation. Earlier, Lin and Manocha [LM93] presented efficient algorithms for curved models composed of spline surfaces and algebraic surfaces undergoing rigid motion. Other approaches for deformable motion are based on polygonal approximation of the model and checking the resulting polygons for collision [BW92]. However, such algorithms may not be accurate. It is possible that the actual surfaces collide with each other, but their polygonal approximations do not. In the animation literature, a number of algorithms for collision detection have been proposed for specific environments like cloth animation, dressing of synthetic actors, hair animation etc. [CYTT92, LND92, YT93, KAT93]. More recently, Volino et. al. [VT94, VCT95] have presented efficient algorithms for self-collision detection on smoothly discretized surfaces undergoing complex...
deformations like wrinkling as well as adaptive spatial subdivision algorithms for animated rigid bodies [BT95].

3 Background

In this section, we review some of the techniques used in our collision detection algorithm presented in Section 4. These include fast algorithms for collision detection between polyhedral models, large environments composed of multiple moving objects, and the extension to hierarchical models.

3.1 Sweep and Prune

Given a large environment with \( N \) moving objects, we use a bounding box based scheme to reduce the bottleneck of testing all possible \( O(N^2) \) pairs of objects for collisions. We make use of the Sweep and Prune approach for sorting the bounding boxes [CLMP95]. This approach is based on projecting each bounding box onto the \( x, y \) and \( z \) axis and performing one-dimensional sort along each axis on the resulting intervals. A pair of bounding boxes overlap if and only if their intervals overlap in all three dimensions. In general, the sort in this algorithm would take \( O(N \log N) \) time, where \( N \) is the number of objects. We can reduce this time bound by retaining the sorted lists from the previous frame, changing only the interval endpoints. In environments where the objects make relatively small movements between frames, the lists will remain nearly sorted, so we can re-sort using insertion sort in expected \( O(N) \) time. The overall running time of the algorithm is \( O(N + M) \), where \( M \) is the number of bounding boxes actually overlapping. In practice \( M \) is typically a small number. Sweep and Prune algorithms have been used to compute overlaps between bounding boxes of thousands of moving objects at interactive rates [CLMP95].

3.2 Hierarchical Sweep and Prune

The sweep and prune technique has been extended to hierarchical models in [PML95]. Assume that we are given two hierarchical representations and the primitives at each hierarchical model in [PML95]. Assume that we are given two hierarchical representations and the primitives at each hierarchical model in [PML95]. Assume that we are given two hierarchical representations and the primitives at each hierarchical model in [PML95]. Assume that we are given two hierarchical representations and the primitives at each hierarchical model in [PML95]. Assume that we are given two hierarchical representations and the primitives at each hierarchical model in [PML95].

The algorithms described in this paper are applicable to polygonal surfaces or spline surfaces undergoing polynomial deformation. The resulting surfaces after the deformation are represented in terms of Bézier or B-spline surface patches. In particular, each patch is a piecewise rational surface in \( \mathbb{R}^3 \) of the form:

\[
F(s, t) = (X(s, t), Y(s, t), Z(s, t), W(s, t)).
\]

Geometrically, each patch is represented in terms of control points and a linear combination of the basis functions:

\[
F(s, t) = (\sum_{i=0}^{m} \sum_{j=0}^{n} V_{ij} B_{i,m}(s) B_{j,n}(t)),
\]

where \( V_{ij} = (x_{ij}, y_{ij}, z_{ij}, w_{ij}) \) are the control point coordinates. The basis functions correspond to Bernstein polynomials of the form \( B_{i,m}(s) = \binom{m}{i} s^i (1-s)^{m-i} \) for Bézier patches. The basis functions for B-spline surfaces are defined using knot sequences [Far93]. In this paper, we will assume that the basis functions for B-spline surfaces have at least \( C^1 \) continuity. Furthermore, the entire surface is contained
in the convex hull of the control points, \( V_{ij} \) [Far93]. This is referred to as the convexity property of the control points.

### 3.5 Intersection of Bézier and B-spline Surfaces

There are several conditions, some necessary and some sufficient, that can be used for interference detection between Bézier and B-spline patches. The most obvious necessary condition is overlap between the bounding boxes of the patches. This simple test can be performed by a sweep-and-prune algorithm as described earlier. Determination of the bounding box of a spline patch is remarkably simple; the patch is completely enclosed by the convex hull of its control points. The bounding box of its control points thus provides a bounding box suitable for this simple exclusion test, i.e. overlap between bounding boxes of the patches.

An improved intersection algorithm between a pair of patches is based on the convexity property of the control polytopes. If the convex hulls of the control points do not overlap, the patches do not intersect. Otherwise each patch is subdivided into two or more patches and the intersection algorithm is applied recursively to each pair. The overlap between convex hulls is tested using linear programming. Other algorithms for interference detection test for intersection between the boundary curves of one patch with the other patch (and vice-versa). This test is combined with checks for closed loops between two patches [Hoh91]. All of these tests are implemented using subdivision based algorithms. The resulting algorithms are relatively simple to implement and serve as simple rejection tests for patches far away from each other. However, consider two patches which are only a small distance away from each other. The subdivision algorithm would be applied a number of times to each patch. Not only is the resulting approach slow, it results in data proliferation. Other algorithms for surface intersection are based on tracing methods [MC91, KM94]. However, these algorithms require some pre-processing and are not fast enough for deformable models, where the control points are changing.

### 4 Our Algorithm

In this section, we present the efficient and accurate collision detection algorithm for objects undergoing polynomial deformation. At each stage of the algorithm a new control point representation is computed for each patch constituting the boundary based on the deformation. We make no assumption on the motion of the objects.

#### 4.1 Overview

An algorithm for interference detection of polynomial deforming objects needs to check for self-collision of a patch, overlap between adjacent patches and intersection among patches in close vicinity of each other. When patches are found to be potentially intersecting, subdivision is used and tests are repeated. To improve efficiency, we initially store a hierarchical representation of each patch.

Our algorithm has a preprocessing stage where each surface patch (or polygon) is subdivided into smaller patches and represented hierarchically (as shown in Fig. 1). Each leaf node corresponds to a spline patch whose surface area is less than an input parameter \( \Delta \). The resulting tree has a depth of three or four and each node can have multiple children. The main motivation of this hierarchical decomposition is in reducing the number of subdivisions needed at run-time; it is more efficient to deform a subdivided patch than to subdivide a deformed patch. Hierarchical sweep-and-prune can be efficiently employed to check for potential overlaps of this hierarchy.

![Figure 1: Hierarchical subdivision of spline patch](image)

Given the control point representation of the object boundary, the overall algorithm at run-time proceeds in the following manner.

- Check each patch corresponding to the boundary of the original object for self-collision.
- Check adjacent patches of an object for intersections.
- Use the hierarchical sweep-and-prune algorithm to find all pairs of patches in close vicinity of each other. The resulting algorithm proceeds top down and computes the control point representation of each node using the polynomial deformation in a lazy manner and computes new axis-aligned bounding boxes accordingly.
- Use the exact intersection algorithm to check the patch pairs in close vicinity of each other.

We elaborate each step of the algorithm in detail below.

#### 4.2 Self-Collision Detection

A spline patch can be self-intersecting, as shown in Fig. 2. Given the control points corresponding to the patch, our
goal is to check for self-intersection. In general, no good, efficient and robust algorithms are known for detecting self-intersections.

![Figure 2: A self-intersecting spline patch, \( F(s, t) \)](image)

Our approach to checking for self-intersection reduces to testing whether the pseudo-normal patch includes the origin, and testing for curve-surface self-intersections.

A necessary condition for a closed-loop self-intersection is the existence of two points on the surface where the surface normals are opposite in direction. We check for this necessary condition using the pseudo-normal patch and Gauss map. Given a \( m \times n \) tensor product Bézier patch, \( F(s, t) \), its pseudo-normal patch is defined as:

\[
N(s, t) = F_s(s, t) \times F_t(s, t),
\]

where \( F_s \) and \( F_t \) are the partial derivative vectors. The pseudo-normal patch is a Bézier patch as well and its parametric degree is \((2m-1) \times (2n-1)\) for a polynomial patch and \(3m \times 3n\) for a rational patch. The pseudo-normal patch of the spline surface in Fig. 2 is shown in Fig. 3. The projection of the pseudo-normal patch onto a unit sphere is the Gauss map of \( F(s, t) \) (as shown in Fig. 4). Its algebraic representation typically involves a square-root expression and cannot be exactly represented as a Bézier patch. A sufficient condition that a given patch is not self-colliding in a closed-loop is the fact that its Gauss map does not contain the origin; this means that there are no two points whose normals are opposite in direction.

The resulting algorithm for checking for self-collision computes the control points of its pseudo-normal patch. It uses linear programming to check whether the convex hull of the control points contains the origin. If there is no separating plane between the control points of \( N(s, t) \) and the origin, it subdivides \( N(s, t) \) and recursively checks whether each subdivided patch contains the origin. This step is repeated until each subdivided patch is almost linear or its area is less than \( \Delta \). At this stage, an algebraic check is made to ensure that the subdivision is converging on a self-intersection.

Testing for curve-surface self-intersection is somewhat non-trivial; the test can be performed algebraically; for example, to test for self intersection of the surface with the curve, \( t = 1 \), we generate three equations of the form:

\[
\frac{F_s(s, t) - F_u(u)}{s-u} = 0
\]

and compute all the solutions in the domain, \((s, t, u) \in [0, 1] \times [0, 1] \times [0, 1]\). The solutions are computed using resultants and eigenvalues [Man94b, Man94a]. This test is relatively slow. To speed up the computation, we test for intersection between the edges and a slightly reduced patch (obtained by trimming the domain), using the convexity properties and subdivision.

![Figure 3: Pseudo-normal patch of \( F(s, t) \)](image)

![Figure 4: Gauss map of \( F(s, t) \)](image)
Figure 5: Front view of adjacent Bézier patches: $F_1(s,t)$ and $F_2(s,t)$

4.3 Intersection between Adjacent Patches

A simple subdivision algorithm based on convexity property and linear programming does not work well for adjacent patches, as the two patches are overlapping along the common boundary. Rather we utilize the fact that either the intersection will be a closed loop intersection, or an edge-surface intersection [Hoh91].

Overlap of the pseudo-normal patches is a necessary condition for a closed loop intersection between two adjacent patches of a surface. Given two adjacent Bézier patches $F_1(s,t)$ and $F_2(s,t)$ (as shown in Fig. 5). Let $N_1(s,t)$ (shown in Fig. 7) and $N_2(s,t)$ (shown in Fig. 8) be the corresponding pseudo-normal patches, respectively. A sufficient condition for non-closed loop overlap between the adjacent patches corresponds to the fact that their pseudo-normal patches do not overlap. The pseudo-normal patches of $F_1(s,t)$ and $F_2(s,t)$ are shown overlapping in Fig. 9. This condition is tested by computing the control points of each of the pseudo-normal patches and testing them for overlap using linear programming to test the feasibility of constructing a separating plane through the origin.

We can use the convexity property of curves and surfaces to perform an exclusion test for curve-surface intersection. We use linear programming to test for the feasibility of constructing a separating plane between the control points of the curve and surface. This plane must pass through a common point if the curve and surface share a corner of the model.

In the case where one of these rejection tests fail, the patches are subdivided and the algorithm is applied recursively.

Figure 6: Back view of adjacent Bézier patches: $F_1(s,t)$ and $F_2(s,t)$

Figure 7: Pseudo-Normal patch of $F_1(s,t)$

4.4 Interference detection between non-adjacent patches

The resulting algorithm uses hierarchical sweep-and-prune to avoid checking all the $O(N^2)$ patch pairs for collision. Initially, it computes the control points of the deformed patch at the highest level of the hierarchy and uses these to compute the smallest axis-aligned bounding box volume enclosing the patch (as shown in Plate 2). The bounding boxes are used for finding pairs of patches in close vicinity of each other. The algorithm is applied recursively to all the children of the nodes, whose bounding boxes are overlapping. This involves computation of new control points, updating bounding boxes and checking for overlap. All pairs of leaf nodes in close vicinity of each other are tested for overlap based on the exact interference test presented in section 3.5.
4.5 Frame-to-frame Coherence

In most applications, the objects undergo small deformations between successive instances. As a result, the “geometric state” of the objects at the last frame is a good approximation to the state at the current instance. The hierarchical sweep-and-prune approach utilizes this coherence in terms of sorting 3D bounding boxes. The resulting algorithm for finding all object pairs in close vicinity of each other takes about $O(N)$ time in practice.

The major bottleneck in the intersection algorithms listed above is linear programming and surface subdivision. As the objects undergo polynomial deformation, the new control points are computed for each patch. In [Bar90], Baraff has presented an algorithm to detect collisions between convex polytopes undergoing rigid motion using linear programming and coherence. In particular if two convex polytopes do not overlap, there exists a separating plane overlapping with the face or edge of one of the polytopes. The particular face or edge is used as a “witness” and used in computing the separating plane at the next instance [Bar90]. Unfortunately, it is difficult to extend this approach to curved objects undergoing deformation. We do not compute the convex hull of the control points and therefore, no edge or face information is available to us. Furthermore the number of edges and faces are not fixed, as the objects undergo deformation.

Rather we use a simple scheme to exploit coherence. Whenever the convex hulls of two objects are non-overlapping, there exists a family of separating planes. The constraints of linear programming, as defined in Section 2.4, correspond to the boundary of a convex region in $\mathbb{R}^4$ (defined by the unknowns $a, b, c$ and $d$). The existence of a separating plane implies that this region is not a null set. We extend the randomized algorithm in [Sei90] to compute a point in this convex region. It corresponds to a separating plane, $P$, of the form $ax + by + cz + d$. At the next frame, we check whether $P$ is a separating plane between the modified control points. If not, we reformulate the constraints of linear programming using the new control points, check for a feasible solution and compute a new separating plane $P$ (if it exists).

5 Implementation

We have implemented a subset of the algorithms described above. The initial results are very promising, and more experiments are planned. The randomized linear programming algorithm is relatively simple to implement and works very well in practice. The routines for sweep-and-prune were found to be no less efficient for small numbers of objects than the $N^2$ algorithm. In particular, we tested the performance of the algorithm on polygonal models undergoing quadratic deformation, as described by Witkin and Welch [WW90], and used by Baraff and Witkin [BW92] for dynamic simulation of non-penetrating flexible bodies.

5.1 Quadratic Deformations

The choice of the quadratic deformation model was motivated by its simplicity and powerful capabilities [TPBF87, TF88a, WW90, BW92]. It is reasonably fast in terms of computing the deformation and a representation of the boundary as Bézier patches. The model is that of space undergoing a quadratic transformation in each dimension.

For dimension $i$, it is represented as:

$$ x_i' = f_i(1, x, y, z, x y, x z, y z, x^2, y^2, z^2) $$

thereby implying that each function contains no other coefficients of the variables $x, y$ and $z$. A parametric plane in the object space is defined as:

$$ x_i = f_i(1, s, t) $$

We assume that the domain of the patch is a subset of $(s, t) \in [0, 1] \times [0, 1]$. The application of a quadratic deor-
It can be represented as a trimmed biquadratic Bézier patch. The domain of the patch is a polygon in the $[0, 1] \times [0, 1]$ domain. We compute the control points of this patch.

Given polyhedral models undergoing quadratic deformation, we compute the boundary in terms of trimmed biquadratic polynomial Bézier patches. We make use of many of the intersection detection methods described earlier. One advantage of this deformation model is that those Bézier patches which result from a quadratic deformation of a plane only have biquadratic pseudonormal patches; general biquadratic Bézier patches have bicubic pseudonormal patches.

$$f(s, t) = f(1, s, t),$$
$$\frac{\partial f}{\partial s} = f(1, s, t),$$
$$\frac{\partial f}{\partial t} = f(1, s, t).$$

Our current implementation supports objects composed of arbitrary convex polygons, however triangles and parallelograms are most efficient. Non-convex polygons are triangulated using the algorithm presented in [NM95]. Furthermore, these objects can be rendered without 'cracks' appearing between adjacent faces, and are easily subdivided with no redundancy in the parameter space. In future, we intend to test our algorithms with more complex deformation functions.

5.2 Collision Detection

The hierarchy of bounding boxes we use is based entirely upon the model structure. The parent bounding boxes are those of the objects themselves. The next level is composed of bounding boxes of each face and further levels are created as faces are subdivided based on the area criterion. In this deformable model, the corner control points of the patch are defined by the coordinates of the world-space quadratic function at the four corners of the parameter space. All the other control points are additionally determined by the local partial derivatives.

We need to determine the control points in order to render each patch. In doing so, we compute an axis-aligned bounding box for the patch by recording the minima and maxima of the control points along each axis. This calculation provides bounding box information for each spline patch. The bounding boxes of the objects are computed from the bounding boxes of the patches. The implementation of the sweep-and-prune algorithm maintains sorted interval lists using an insertion sort with modified swapping. Whenever the bounding boxes of two bodies begin intersecting, we construct a new interval list for the pair composed of bounding boxes of the component patches. This operation can be performed rapidly because we already maintain interval lists within each body to perform efficient self-intersection testing.

5.3 Surface-Surface Intersection Testing

The cost of performing a curve-surface intersection test using subdivision methods is more than half of the cost of performing the test entirely using surface subdivision. Given that typical patches have four edges, in practice it is more efficient to perform surface-surface intersection tests entirely using subdivision and using the convexity property than with closed loop and curve-surface intersection tests.

We subdivide the parameter space of each patch into four equal parts, determining the control points and bounding boxes of each subpatches. This is followed by bounding box and separating plane tests and testing between each relevant pair of subpatches. We terminate the subdivision when subpatches are sufficiently planar. At that stage, we approximate them with planes and check the resulting planes for overlap.

5.4 Inter-Patch Self-Intersection Testing

Testing for self intersection of the bodies is significantly more difficult in practice by the extreme proximity of their constituent patches. We can apply our normal surface-surface intersection test between non-adjacent patches. It is not suitable for testing between adjacent patches however because the subdivision will traverse all the way down at the adjoining edge.

We apply the closed loop and curve-surface intersection rejection tests; if these fail, we subdivide the faces parallel to the adjoining edge, apply surface-surface intersection testing between non-adjacent subpatch pairs, and perform the adjacent-face test between the two adjacent subpatches.

5.5 Intra-Patch Self-Intersection Testing

The final phase of self-intersection testing is to test for self intersection of each patch, and this is the most time consuming part. Either the self intersection will be a closed loop self-intersection or a curve-surface self-intersection.

The loop intersection can be rejected with a linear programming test, by constructing a plane through the origin which holds the entire pseudonormal patch on one side. In fact, a sufficient condition for this test to succeed is that any of the adjacent-patch pseudonormal tests could construct a separating plane so this test actually comes for free.

If no separating plane can be constructed, we must subdivide and test the plane against itself. The adjacent-face test could be used here, but it is in fact more efficient to use another procedure here. We subdivide the patch into four subpatches along the longest edge, corresponding to the first third, the first two thirds, the second two thirds and the final third. We perform patch self-intersection tests on the two larger subpatches, and patch-patch intersection tests between the two smaller subpatches. This accounts for all possible self-intersections, and handles the close proximity efficiently.
5.6 Collision Detection with Planes

A commonly desired intersection test is that with the infinite plane, for testing against the floor and boundaries of an environment. Any Bezier patch \( F \) parameterized by \( s \) and \( t \), can be considered independently in \( x, y, \) and \( z \). The patch will achieve a maximum or minimum with respect to the plane:

\[
ax + by + cz + d = 0
\]

at an extreme point of its trimming curve, or where the derivative of the function:

\[
aFx + bFy + cFz + d = 0
\]

becomes zero:

\[
aFx,s + bFy,s + cFz,s = 0
\]

\[
aFx,t + bFy,t + cFz,t = 0
\]

Determining these extrema becomes the problem of evaluating the two parameter-space curves which correspond to the zero of these partial derivatives, and determining the intersections between these curves and each other and the trimming curve of the parameter space. The deformation function must then be evaluated at the corners of the trimming curve, and all of the partial-derivative intersections.

For patches produced by the quadratic deformation, these zero partial-derivative curves are linear. To test a patch with four sides against the floor, the deformation function must be evaluated at the four corners of the parameter space, the four intersections of the zero derivative curves with the perimeter and the one intersection of the curves with each other. This is much more efficient than testing using subdivision. Bounding boxes can obviously still be used as an exclusion test.

6 Performance

We ran our simulations on an SGI Onyx with Reality Engine 2 graphics capability. Figure 10 demonstrates performance in an environment of approximate density 1/64 composed of interacting, deforming cubes, i.e. the undeformed objects occupied one sixty-fourth of the local space. This timing does not include self-interference testing; the models’ deformation functions are constrained to prevent self-intersection, by requiring that the deformation functions be monotonically increasing within the domain of the model.

6.1 Environment

Our simulation environment consists of multiple dynamically deforming bodies interacting in a shared environment. Each body is composed of a mesh of polygons undergoing dynamic quadratic deformation. In world space, the polygons are transformed to trimmed biquadratic Bezier patches which are adaptively tessellated for display.

6.2 Results

The final plates demonstrate some of our results.

Plate 1 shows one of the more complex environments we have modeled; deforming polygonal fish interact in an environment of deforming blades of grass. The quadratic deformation can be applied easily to achieve realistic motion of the objects. The second scene contains over 120 interacting objects in a relatively dense environment. Performance is still dominated by rendering and update of the deformation functions.

Plate 2 is composed of three subsequent frames of an animation of collision between two complex deforming objects. In the initial frame, the object bounding boxes are not overlapping. In the second frame, the objects have moved closer and two levels of the bounding box hierarchy begin to overlap (indicated by brighter bounding boxes), requiring patch-patch intersection testing. Finally the objects intersect.

References


Plate 1 shows one of the more complex environments we have modeled; deforming polygonal fish interact in an environment of deforming blades of grass. The quadratic deformation can be applied easily to achieve realistic motion of the objects. The second scene contains over 120 interacting objects in a relatively dense environment. Performance is still dominated by rendering and update of the deformation functions.

Plate 2 is composed of three subsequent frames of an animation of collision between two complex deforming objects. In the initial frame, the object bounding boxes are not overlapping. In the second frame, the objects have moved closer and two levels of the bounding box hierarchy begin to overlap (indicated by brighter bounding boxes), requiring patch-patch intersection testing. Finally the objects intersect.

References


Figure 10: Performance of the Algorithm


Plate 1. Animation with Colliding Deformable Bodies.


Plate 2a.
Before Collision, No Overlap.

Plate 2b.
Bounding Box Overlap.

Plate 2c.
Collision Detected.