Efficient and Accurate Interference Detection for Polynomial Deformation and Soft Object Animation *

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

Abstract: We present efficient and accurate algorithms for collision and 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, loop intersection tests and singularity detection. 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.

*Supported in part by a Sloan foundation fellowship, NSF grant CCR-9319957, ARO Contract P-34982, ONR contract N00014-94-1-0738, ARPA contract DABT63-93-C-0048, NSF/ARPA Science and Technology Center for Computer Graphics & Scientific Visualization

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 [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, CP91], 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 algorithms for collision detection 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 $O(n\ logn)$ 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 Color Plates 2,3, 5a and 5b). Our system can accurately compute all the contacts between the models at *interactive* rates.

1.1 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 are close, 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 [Hof89]. 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 implicits, 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 takes *order of thousands of* *iterations* of interval arithmetic, which corresponds to a few tens of seconds on a high end workstation, to compute the contact between time dependent parametric models. They are *too slow* as compared to algorithms presented in this paper. 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].

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 and present efficient algorithms for detecting interference. We describe the algorithm implementation, performance and application in Section 4.

2 Background

In this section, we review some of the techniques used in the collision detection algorithm presented in Section 3. These include fast algorithms for collision detection between polyhedral models, large environments composed of multiple moving objects, and extension to hierarchical models.

2.1 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. In most realistic situations, an object has to be tested against only a small fraction of all objects in the environment (in its vicinity) for collision. We make use of the *Sweep and Prune* approach for sorting the bounding boxes [CLMP95]. In particular, the approach is based on projecting each bounding box onto the x, y and z axis and performing one-dimensional sort along each axis. If two bounding boxes collide in 3-D, then their orthogonal projections on the x, y, and z axes must overlap. Since the bounding boxes are axially-aligned, projecting them onto the coordinate axes results in intervals. We are interested in overlaps among these intervals, because a pair of bounding boxes can overlap *if and only if* their intervals overlap in all three dimensions. The resulting algorithm maintains three lists, one for each dimension. Each list contains the values of the endpoints of the intervals in each corresponding dimension. By sorting these lists, we can determine which intervals overlap. In general, such a sort 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. The resulting algorithm has been used to compute overlaps between bounding boxes of thousands of moving objects at interactive rates [CLMP95].

2.2 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 leaf node correspond to elements which can be tested for collision. We compute the smallest axis-aligned bounding volume for each leaf node. The bounding box at each intermediate node is computed by taking a minimum volume union of the bounding boxes of the children of the node. The resulting algorithm for overlap testing proceeds recursively. Initially it tests the bounding boxes at the root for overlap. Upon overlap, the children of these bounding boxes are sorted using the *Sweep and Prune* technique to determine if they too overlap. Sorting and determining children's overlap status is done recursively until we reach the leaf boxes which contain the primitives of both hierarchies. If the hierarchies are of the same height, then we test the boxes at the same level. If the hierarchies are of different heights, then the last level of the shallower hierarchy is tested against the descending levels of the other hierarchy. The interval lists of lower levels are maintained for the duration of overlap between the parents.

The level by level sorting of the bounding box hierarchies is tricky to manage both in terms of efficiency and memory usage for objects composed of thousands of primitives. Multiple lists have to be sorted at each level since not all children overlap. Moreover, this information needs to be managed efficiently. The implementation uses a bounding box management scheme that allocates the space for the data once, but can have multiple pointers to it. So instead of allocating this data value at multiple levels in the hierarchy, it efficiently swaps pointers during the sort. It also allocates only as much memory per pair of objects as the number of features of the objects that are interacting (as opposed to allocating memory for all possible feature pairs for the objects). More details are given in [PML95].

2.3 Linear Programming

Many algorithms for interference detection are reduced to checking whether the convex hulls of sets of points overlap. One possible solution is based on determining the convex hull of a set of points and checking the resulting polytope for overlap, however this takes $O(n \log n)$ time. Rather, we reduce this problem to linear programming.

Given two sets of points $\mathbf{S}_1$ and $\mathbf{S}_2$, let $\mathbf{C}_1$ and $\mathbf{C}_2$ be their convex hulls respectively. A necessary and sufficient condition for the non-intersection of $\mathbf{C}_1$ and $\mathbf{C}_2$ is the existence of a separating plane between $\mathbf{S}_1$ and $\mathbf{S}_2$. By definition, the convex hull of a set of points includes all extremal points of the set with respect to any plane. This separating-plane test between convex hulls is equivalent to a separating-plane test between the sets of points. For the given point sets in three-dimensional space, this separating plane test can be expressed as a linear-programming problem. The constraints of the linear programming are of the form:

$$\forall_{i,j} \begin{cases} i \in \mathbf{S}_1 \Rightarrow ax_i + by_i + cz_i + d > 0 \\ j \in \mathbf{S}_2 \Rightarrow ax_j + by_j + cz_j + d < 0 \end{cases}$$

The minimizing function can be set to a constant. We only care about the existence of a, b, c and d such that these constraints are satisfied. In computational geometry literature, algorithms of linear complexity to solve the linear programming problem are well-known (given a fixed dimension). In our application, we make use of Seidel's randomized linear time algorithm [Sei90]. It works very well in practice.

2.4 Bézier and B-spline surfaces

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 $\Re^3$ of the form:

$$\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t)).$$

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

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

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 $B_{i,m}(s) = \begin{pmatrix} m \\ i \end{pmatrix} s^i(1-s)^{m-i}$ 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-splines 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 as the *convexity* property of the control points.

2.5 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. 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.

3 Collision Detection

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 assume that the polygons correspond to a special instance (a linear surface patch). Furthermore, we make *no assumption* on the motion of the objects. The resulting algorithm for interference detection needs to check for:

- Self-collision
- Overlap between adjacent patches of an object
- Interference test between non-adjacent patches in close vicinity of each other

We will make use of algorithms based on the convexity properties and subdivision to check for these intersections. However, the resulting algorithms can be slow when two objects are fairly close to each other or actually collide. To circumvent those problems, we initially store a hierarchical representation for each patch.

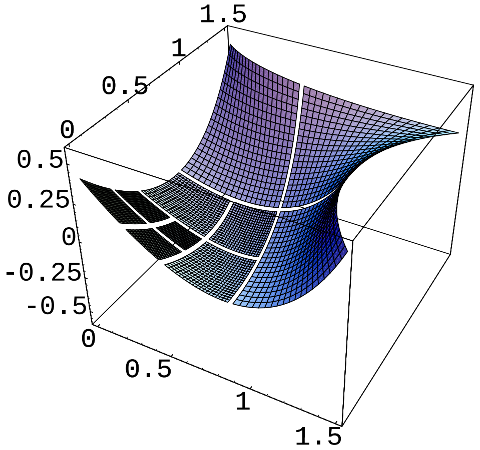


Figure 1: Hierarchical subdivision of spline patch

3.1 Model Preprocessing

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 Δ. The resulting tree has a depth of three or four and each node can have multiple children. The main motivation of this hierarchical decomposition arises from the fact that we do not need to perform too many subdivisions to check for overlap at run-time. Furthermore, we compute an axis-aligned bounding box corresponding to each node in the tree. The resulting algorithm makes use of the hierarchical sweep-and-prune algorithm to check for potential overlaps.

3.2 Online Algorithm

Given the control point representation of the object boundary, the overall algorithm at the 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 in the following sections.

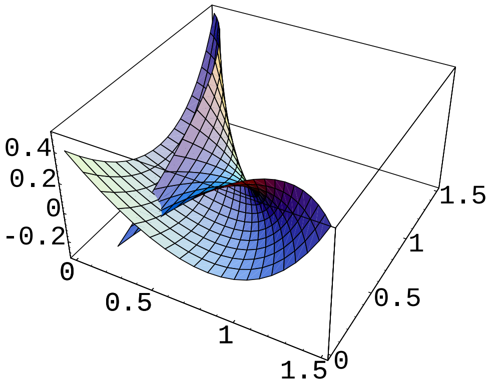


Figure 2: A self-intersecting spline patch, $\mathbf{F}(s,t)$

3.2.1 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.

Our approach to checking for self-intersection reduces to testing whether the pseudo-normal patch includes the origin. A necessary condition for the 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, $\mathbf{F}(s,t)$, its pseudo-normal patch is defined as:

$$\mathbf{N}(s,t) = \mathbf{F}_s(s,t) \times \mathbf{F}_t(s,t),$$

where $\mathbf{F}_s$ and $\mathbf{F}_t$ 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 $\mathbf{F}(s,t)$ (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 is the fact that its Gauss map does not contain the origin.

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 $\mathbf{N}(s,t)$ and the origin, it subdivides $\mathbf{N}(s,t)$ 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 Δ. At this stage, an algebraic check is made to ensure that the subdivision is converging on a cusp and not a self intersection.

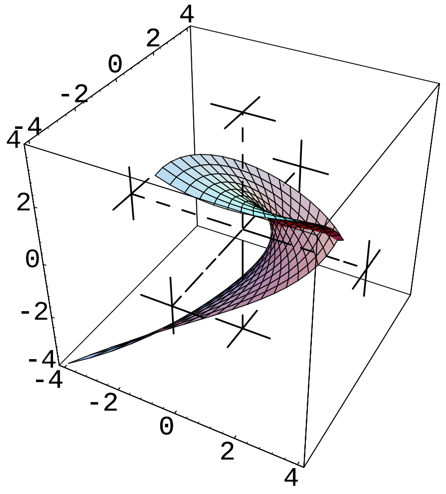


Figure 3: Pseudo-normal patch of $\mathbf{F}(s,t)$

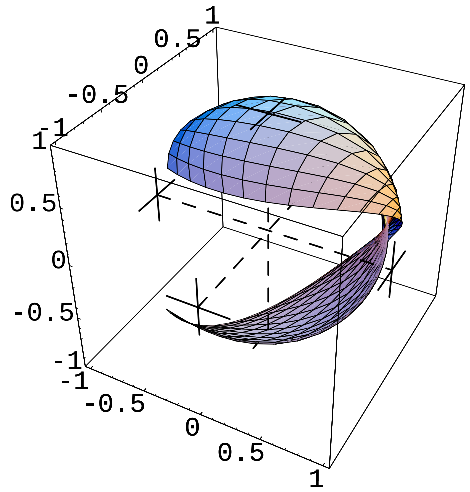


Figure 4: Gauss map of $\mathbf{F}(s,t)$

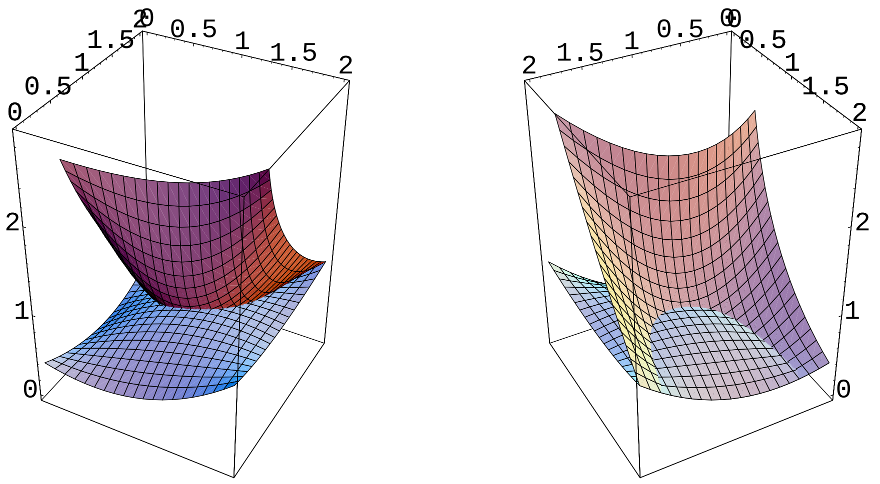


Figure 5: Two views of adjacent Bézier patches: $\mathbf{F}_1(s,t)$ and $\mathbf{F}_2(s,t)$

3.2.2 Intersection between Adjacent Patches

The adjacent patches share a common boundary curve. A simple subdivision algorithm based on convexity property and linear programming does not work well, as the two patches are overlapping along the boundary. Rather we utilize the fact that overlap of the pseudo-normal patches is a necessary condition to detect self intersection between two adjacent patches of a surface, or else the surface self-intersection will be detected elsewhere.

Given two adjacent Bézier patches $\mathbf{F}_1(s,t)$ and $\mathbf{F}_2(s,t)$ (as shown in Fig. 5). Let $\mathbf{N}_1(s,t)$ (shown in Fig. 6) and $\mathbf{N}_2(s,t)$ (shown in Fig. 7) be the corresponding pseudo-normal patches, respectively. A sufficient condition for non-overlap between the adjacent patches (at other points besides the common boundary curve) corresponds to the fact that their pseudo-normal patches do not overlap. The pseudo-normal patches of $\mathbf{F}_1(s,t)$ and $\mathbf{F}_2(s,t)$ are shown overlapping in Fig. 8. This condition is tested by computing the control points of each of the pseudo-normal patches and testing them for overlap using linear programming. In the case where they overlap, the patches are subdivided and the algorithm is applied recursively.

3.2.3 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 patch at the root based on the polynomial deformation and uses them to compute the smallest axis-aligned bounding box volume enclosing a given patch (as shown in Color Plate $4a$, $4b$ and $4c$). The bounding boxes are also used for finding pair 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 2.5.

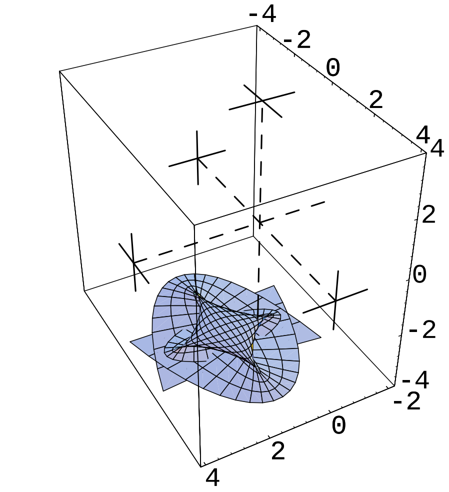


Figure 6: Pseudo-Normal patch of $\mathbf{F}_1(s,t)$

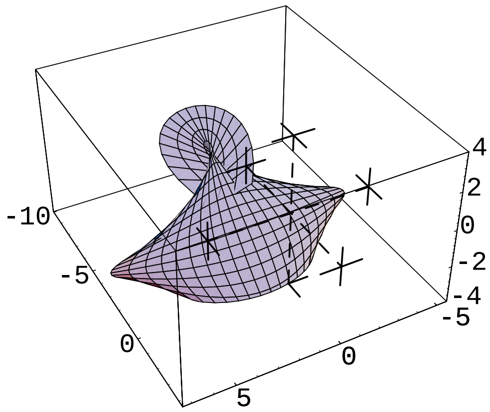


Figure 7: Pseudo-Normal patch of $\mathbf{F}_2(s,t)$

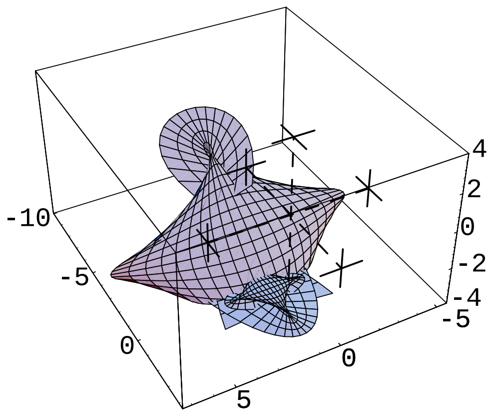


Figure 8: Overlapping pseudo-normal patches of $\mathbf{F}_1(s,t)$ and $\mathbf{F}_2(s,t)$

3.3 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 plane. The constraints of linear programming, as defined in Section 2.4, correspond to the boundary of a convex region in $\Re^4$ (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, $\mathbf{P}$, of the form $ax + by + cz + d$. At the next frame, we check whether $\mathbf{P}$ 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 $\mathbf{P}$ (if it exists).

4 Implementation and Performance

We have implemented a subset of the algorithms described above. The initial results are very promising. 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 proposed by Witkin and Welch [WW90] and used by Baraff and Witkin [BW92] for dynamic simulation of non-penetrating flexible bodies.

4.1 Quadratic Deformations

The choice of the quadratic deformation model was motivated by its simplicity and powerful capabilities [WW90, BW92]. It is reasonably fast in terms of computing the deformation and a representation of the boundary in terms of Bézier patches (as shown in Color Plates 1 and 3). The model is that of space undergoing a quadratic transformation in each dimension. It is represented as:

$$\begin{aligned} x' &\leftarrow f(1, x, y, z, xy, yz, zx, x^2, y^2, z^2) \\ y' &\leftarrow f(1, x, y, z, xy, yz, zx, x^2, y^2, z^2) \\ z' &\leftarrow f(1, x, y, z, xy, yz, zx, x^2, y^2, z^2), \end{aligned}$$

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:

$$\begin{aligned} x &= f(1, s, t) \\ y &= f(1, s, t) \\ z &= f(1, s, t). \end{aligned}$$

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 deformation function turns it into a subset of a biquadratic tensor-product Bézier patch of the form:

$$\begin{aligned} x &= f(1, s, t, st, s^2, t^2) \\ y &= f(1, s, t, st, s^2, t^2) \\ z &= f(1, s, t, st, s^2, t^2). \end{aligned}$$

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 all Bézier patches also have biquadratic pseudonormal patches:

$$\begin{aligned} \tfrac{\delta f}{\delta s} &= f(1, s, t) \\ \tfrac{\delta f}{\delta t} &= f(1, s, t) \\ \tfrac{\delta f}{\delta s} \times \tfrac{\delta f}{\delta t} &= f(1, s, t, st, s^2, t^2). \end{aligned}$$

For a general biquadratic Bézier , the pseudonormal patch will be bicubic.

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.

4.2 Collision Detection

Our collision detection scheme makes use of hierarchical sweep and prune to efficiently determine which patches are in close proximity. The subsequent surface-surface tests are implemented using subdivision and linear programming. Self intersection is tested using pseudonormal patches, subdivision algorithms and linear programming.

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 locations 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.

4.2.1 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. It is also necessary to perform curve-surface intersection tests to make use of any information gained from overlapping pseudonormal patches. 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. Our initial test is performed by the hierarchical sweep-and-prune algorithm. Subsequently, when patch bounding boxes are intersecting we use a randomized linear programming feasibility test to evaluate whether a separating plane can be constructed between the control points of the patches. If this test indicates that no such plane can be constructed, we proceed to subdivide the patches and perform the tests recursively. Otherwise we compute one such separating plane and use it as a starting point for the next frame.

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.

4.2.2 Self-Intersection Testing

Testing for self intersection of the bodies is *relatively more difficult* in practice by the extreme proximity of their constituent planes. 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. Instead, we test for non-overlap of the pseudonormal patches as an exclusion test for intersection.

Our intersection test between adjacent faces tests for overlapping pseudonormal patches by applying linear programming to test the feasibility of constructing a separating plane for the patches, through the origin. If such a plane does not exist, 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.

The final phase of self-intersection testing is to test for self intersection of each patch. This requires a linear programming test to construct 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 pseudormal 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.

4.2.3 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 Bézier patch $\mathbf{F}$ 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 derivate of the function:

$$a\mathbf{Fx} + b\mathbf{Fy} + c\mathbf{Fz} + d = 0$$

becomes zero:

$$\begin{aligned} a\mathbf{Fx}_s + b\mathbf{Fy}_s + c\mathbf{Fz}_s + d = 0 \\ a\mathbf{Fx}_t + b\mathbf{Fy}_t + c\mathbf{Fz}_t + d = 0 \end{aligned}$$

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.

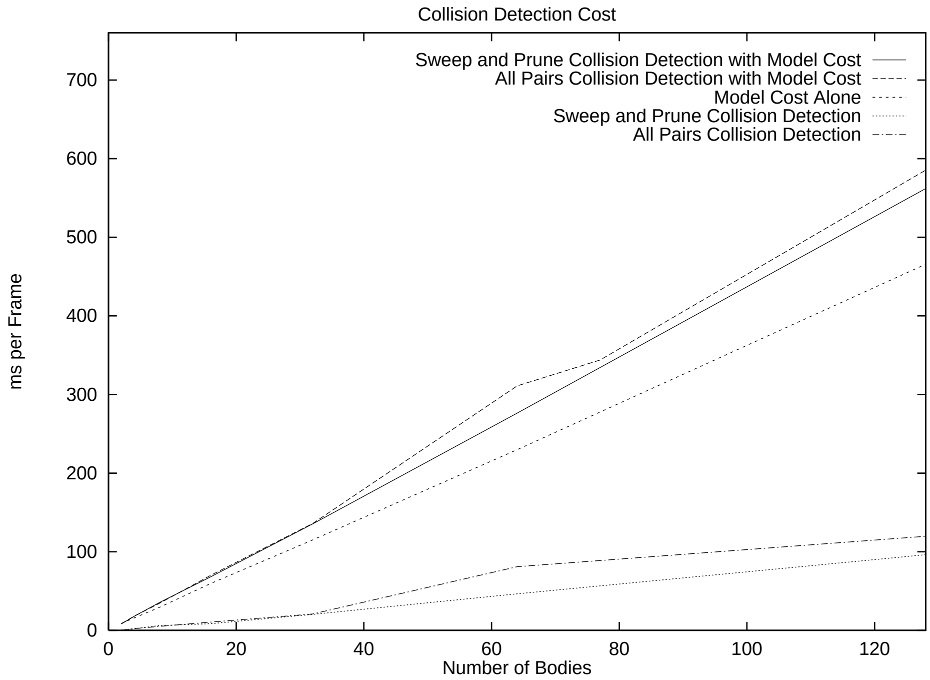


Figure 9: Performance of the Algorithm

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.

Color Plate 2 illustrates the zero-derivative curves of a deformed cuboid. The blue curves are zero-derivative curves with respect to the floor; the red and green curves are with respect to the other two axis planes. The white bars correspond to the extrema of all patches with respect to the floor; these are all either at a corner or along a blue zero-derivative curve.

4.3 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 Bézier patches which are adaptively tessellated for display.

4.4 Performance

We ran our simulations on an SGI Onyx with Reality Engine 2 graphics capability. Figure 9 demonstrates performance in an environment of constant density (1/64) composed of interacting, deforming cubes.

4.5 Results

The color plates demonstrate some of our results. Color Plate 1 depicts a quadratically deformed cube which simply demonstrates our model. The surface normals are indicated across the surface.

Color Plate 2 demonstrates the evaluation of extrema with respect to a plane. The extrema of a deformed cube with respect to the environment walls are tracked efficiently.

Color Plate 3 shows a simple environment consisting of deforming cuboids interacting in a simple environment composed of walls and some static polytopes. Static polytopes can be considered simply as objects whose faces need not be subdivided for intersection detection.

Color Plate 4 is comprised of three subsequent frames of an animation of collision between two complex deforming objects. In the initial frame, parent bounding boxes are not overlapping. Parent and child bounding boxes are displayed. In the second frame, the objects have moved closer and both parent and child bounding boxes are overlapping (indicated by brighter bounding boxes), requiring patch-patch intersection testing. Finally the objects intersect.

Color Plate 5 shows some 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.

References

[Baa84] A. Baar. Global and local deformations of solid primitives. *ACM Computer Graphics*, 18(3):21–30, 1984.

[Bar90] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *ACM Computer Graphics*, 24(4):19–28, 1990.

[BB88] R. Barzel and A. Barr. A modeling system based on dynamic constraints. *ACM Computer Graphics*, 22(4):31–39, 1988.

[BT95] S. Bandi and D. Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. *Computer Graphics Forum*, 14(3):259–270, 1995.

[BW92] D. Baraff and A. Witkin. Dynamic simulation of non-penetrating flexible bodies. *ACM Computer Graphics*, 26(2):303–308, 1992.

[CLMP95] 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.

[Coq90] S. Coquillart. Extended free form deformation: a sculpting tool for 3d geometric modeling. *ACM Computer Graphics*, 24(4):187–193, 1990.

[CP91] S. Coquillart and J. Pierre. Animated free form deformation: An interactive animation technique. *ACM Computer Graphics*, 25(4):23–26, 1991.

[CYTT92] M. Carigan, Y. Yang, N. Magnetat Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex deformable clothes. *ACM Computer Graphics*, 26(2):99–104, 1992.

[Duf92] Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *ACM Computer Graphics*, 26(2):131–139, 1992.

[ea93] J. Snyder et. al. Interval methods for multi-point collisions between time dependent curved surfaces. In *Proceedings of ACM Siggraph*, pages 321–334, 1993.

[Far93] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press Inc., 1993.

[FB88] D. Forsey and R.H. Bartels. Heirarchical b-spline refinement. In *Proc. of ACM Siggraph*, pages 205–212, 1988.

[GJK88] 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:pp. 193–203, 1988.

[Hah88] J. K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):pp. 299–308, 1988.

[HBZ90] B. V. Herzen, A. H. Barr, and H. R. Zatz. Geometric collisions for time-dependent parametric surfaces. *Computer Graphics*, 24(4):39–48, 1990.

[HHK92] W. M. Hsu, J. F. Hughes, and H. Kaufmann. Direct manipulation of free-form deformations. *ACM Computer Graphics*, 26(2):177–184, 1992.

[Hof89] C.M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, San Mateo, California, 1989.

[KAT93] T. Kurihara, K. Anjyo, and D. Thalmann. Hair animation with collision detection. In *Models and Techniques in Computer Animation*, pages 128–38. Springer-Verlag, 1993.

[KM94] S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on the lower dimensional formulation. Technical Report TR94-062, Department of Computer Science, University of North Carolina, 1994.

[LC91] M.C. Lin and John F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, 1991.

[LM93] M.C. Lin and Dinesh Manocha. Interference detection between curved objects for computer animation. In *Models and Techniques in Computer Animation*, pages 43–57. Springer-Verlag, 1993.

[LND92] B. Lafleur, Magentat-Thalmann N., and Thalmann D. Cloth animation with self-collision detection. *Proceedings of IFIP WG5.10 - Modeling in Computer Graphics*, pages 179–185, September 1992.

[MC91] 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.

[MW88] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, 1988.

[NM95] A. Narkhede and D. Manocha. Fast polygon triangulation based on seidel's algorithm. In A. Paeth, editor, *Graphics Gems V*, pages 394–397, Academic Press, 1995.

[Pen90] A. Pentland. Computational complexity versus simulated environment. *Computer Graphics*, 22(2):185–192, 1990.

[PML95] 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.

[PS85] F.P. Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, New York, 1985.

[PW90] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, 23(3):185–192, 1990.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

[Stu87] D. Sturman. A discussion on the development of motion control systems. In *SigGraph Course Notes: Computer Animation: 3-D Motion Specification and Control*, number 10, 1987.

[TQ94] D. Terzopoulos and H. Qin. Dynamic nurbs with geometric constraints for interactive sculpting. *ACM Trans. on Computer Graphics*, 13(2):103–136, 1994.

[TS86] Sederberg T.W and Parry S.R. Free-form deformation of solid geometric models. *ACM Computer Graphics*, 20(4):151–160, 1986.

[VCT95] P. Volino, M. Courchesne, and N. Magnetat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. *ACM Computer Graphics (SIGGRAPH)*, pages 137–144, 1995.

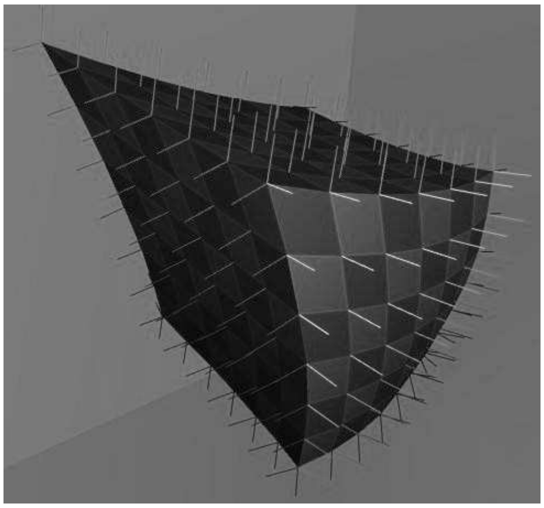
[VT94] P. Volino and N. Magnetat Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum (EuroGraphics Proc.)*, 13(3):155–166, 1994.

[WW90] A. Witkin and W. Welch. Fast animation and control of non-rigid structures. *ACM Computer Graphics*, 24:243–252, 1990.

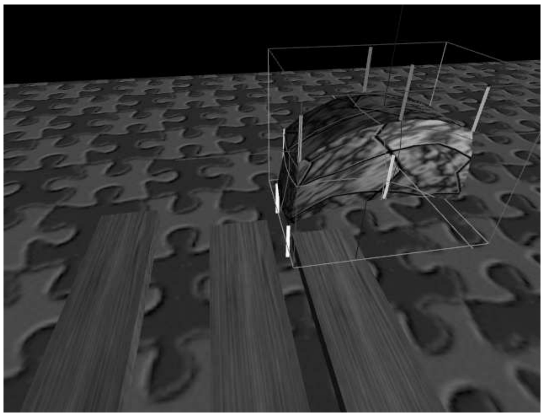
[WW92a] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques*. Addison Wesley, 1992.

[WW92b] W. Welch and A. Witkin. Variational surface modeling. *ACM Computer Graphics*, 26(2):157–166, 1992.

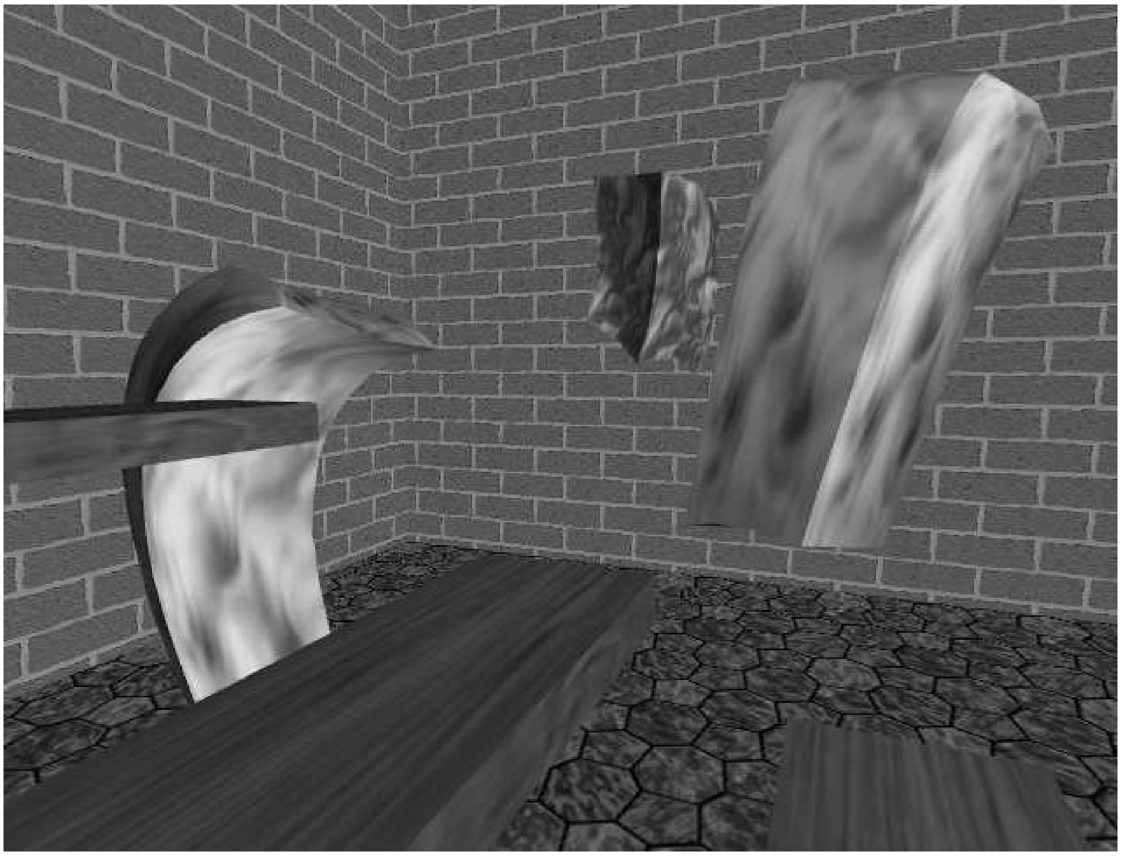
[YT93] Y. Yang and N. Magnetat Thalmann. An improved algorithm for collision detection in cloth animation with human body. *Computer Graphics and Applications (Pacific Graphics Procs)*, pages 237–251, 1993.



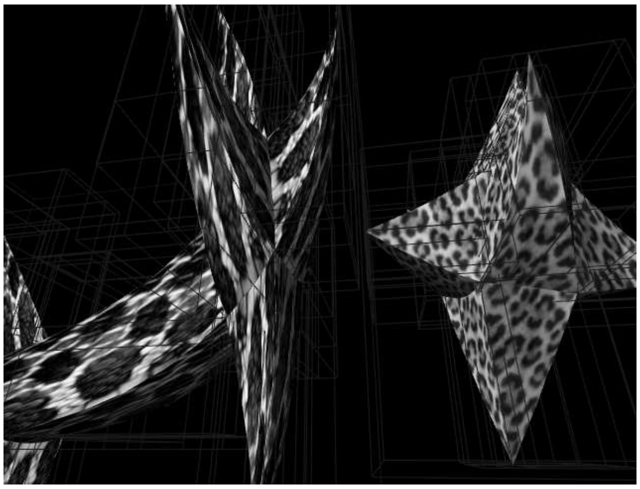
Color Plate 1.
Quadratically Deformed Cube.



Color Plate 2.
Zero-Derivative Curves.

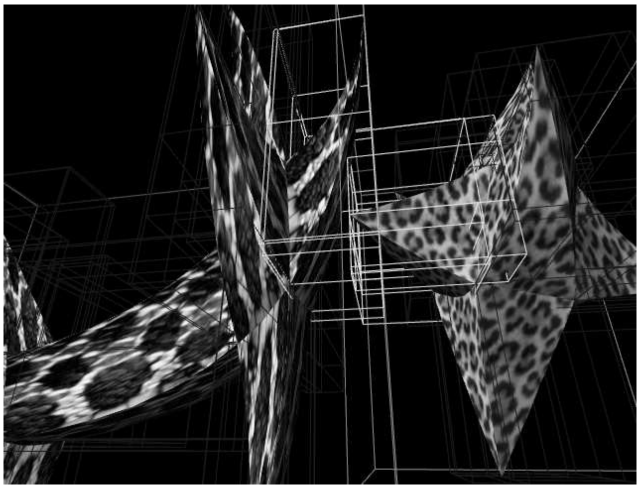


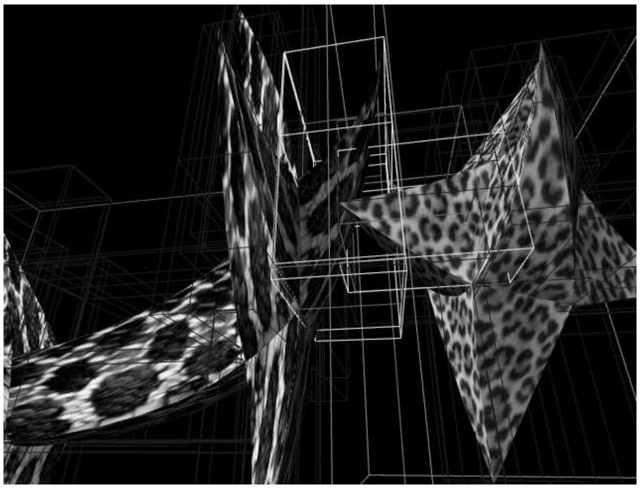
Color Plate 3. Quadratically Deforming Objects.



Color Plate 4a.
Before Collision, No Overlap.

Color Plate 4b.
Bounding Box Overlap.





Color Plate 4c.
Collision Detected.

Color Plate 5a. Animation with Colliding Deformable Bodies.

Color Plate 5b. Animation with Colliding Deformable Bodies.