An Interactive Tool for Placing Curved Surfaces without Interpenetration

John M. Snyder[†]
Microsoft Corporation

Abstract

We present a surface representation and a set of algorithms that allow interactive placement of curved parametric objects without interpenetration. Using these algorithms, a modeler can place an object within or on top of other objects, find a stable placement for it, and slide it into new stable placements. Novel algorithms are presented to track points of contact between bodies, detect new points of contact, and delete vanishing contacts. Interactive speeds are maintained even when the moving body touches several bodies at many contact points.

We describe a new algorithm that quickly brings a body into a stable configuration with respect to a set of external forces, subject to the constraint that it not penetrate a set of fixed bodies. This algorithm is made possible by sacrificing the requirement that a body behave physically over time. Intuitive control is still achieved by making incremental, "pseudo-physical" changes to the body's placement, while enforcing the non-interpenetration constraint after each change.

CR Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.
Key Words: object placement/assembly, collision detection, contact point.

1 Introduction

Geometric modeling can be dissected into two tasks: creating models for a set of "parts" or basic objects, and then assembling these parts into virtual combinations. For example, to model an automobile engine, one can first create a set of parts including the engine block, pistons, nuts, and bolts, and then assemble these into an engine. While the vast majority of work in geometric modeling has focused on the part-creation task, part-assembly is nonetheless an important and difficult problem. Parts can often be modeled independently, perhaps by different people; assembling them means having to deal explicitly with relationships between all the parts together, such as the constraint that no solid object penetrate another. Increasingly in research systems, a part is modeled not as a static geometric object, but as a high-level parameterized model such as an elastic sheet with given initial geometry and material properties [WITK87,META92]. This increases the complexity of the part-assembly task, which must compute model parameters from spatial relationships between the various parts (for example, the shape of the tablecloth after it is dropped on the table).

This paper focuses on a subset of the part-assembly problem: interactive placement of rigid, solid, curved objects into physically plausible configurations (Figure 1). Parts in this context are rigid bodies parameterized by a rigid motion in 3D (i.e., a rotation and translation). The goal is to arrive at a statically balanced configuration of solids near a user-specified initial configuration. Examples include placing a spoon in a cereal bowl or filling it with cereal, placing a phone receiver on its hook, putting ice cubes in a glass, modeling a pile of fruit, fitting oddly shaped objects into a rigid box, etc.. Many applications exist for a tool that performs such interactive placement: modeling of static virtual environments, interactive fitting/packing of 3D shapes, games (e.g. virtual 3D puzzles), and keyframe animation for physically plausible but still controllable motions.

[†] 1 Microsoft Way, Redmond WA 98052.
johnsny@microsoft.com

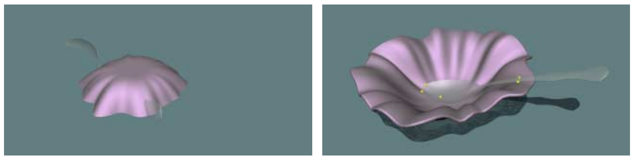


Figure 1: The Placement Problem – The algorithms described here allow, for example, repositioning of a bowl and spoon in an arbitrary configuration (left) to produce a configuration with the spoon resting stably in the bowl and the bowl resting stably on the floor (right). Points of contact computed by the algorithm are shown in yellow; the spoon is rendered transparently to make these visible.

Even this restricted problem is difficult without special tools. Traditionally, an object is placed by directly tying a graphics input device such as a mouse, trackball, or set of dials to its orientation and translation parameters. Users verify by eye that objects do not interpenetrate and are physically balanced. Because curved objects resting on each other can have many points of contact, modeling a configuration like the one in Figure 1 can take many minutes of tedious interaction: one portion of the spoon is satisfactorily placed only after some other part begins interpenetrating or pulls away. Traditional methods also provide no help in reaching a stable configuration.

To assist the user in the placement task, one option is to take advantage of the substantial work done in computer animation to make objects move physically. These techniques can be used for the placement problem by simulating through time until stability is achieved; that is, the objects stop moving. Unfortunately, none of these techniques is suitable for interactive placement of curved, non-convex, parametric shapes; the kind of shapes that many commercial modelers can produce. Most approaches restrict the kinds of shapes that can be simulated [MOOR88,BARA90,SCLA91, see SNYD93 for a survey of these restrictions]; another approach that handles general parametric surfaces is too slow for interactive applications [SNYD93]. An additional disadvantage is that physical behavior over time can actually hinder the placement problem. A physical card house can fall flat when we place the next card; a spoon dropped into a bowl accelerates, bounces, and can end up far from where we wanted it.

However, the placement problem is concerned with quick production of a desired configuration of objects, not with how they arrived there. We seek an approach that arrives at a stable state from some arbitrary state, making changes that are intuitive and controllable, but not necessarily physically accurate. To make the approach fast, we deviate from accurate physical simulation in several ways:

1. Bodies have no velocity. In fact there is no explicit notion of time in the system at all. Instead, a body is moved through a series of discrete states, each of which obeys the non-interpenetration constraint.

2. While forces and torques are solved for at each state, they aren't the real forces that a dynamic body would experience instantaneously. They are chosen to speed convergence of the body to a "nearby" stable placement, and are called *pseudo-forces* and *pseudo-torques.*

3. The exact time when contact point transitions occur (e.g., vanishing contacts) is not calculated. Instead, transitions are made after each discrete step.
4. Small, non-physical adjustments are made to an active body's position and orientation in order to preserve the non-interpenetration constraint.
5. Only one body at a time, the *active* body, can move. As it moves, it interacts with other bodies which remain fixed and are treated as infinitely massive. The user can change which body is currently active in order to move bodies sequentially into position.

As is implied in some of the above choices, the approach presented here is a limited solution to the general placement problem of rigid bodies – a very difficult problem for curved surfaces. Although the sequence of intermediate states does not correspond to a real motion of the object, the final state does represent a stable configuration according to static force balance laws, assuming immovable non-active bodies. The algorithm thus handles the common case of placing a single object on or within another that is essentially fixed, like a teapot on a table or a spoon in a bowl. Force balance is only maintained for the active body; the force it imparts to the rest of the world is ignored. The algorithm therefore can not automatically produce physically balanced heaps of multiple bodies. However, visually plausible heaps can still be modeled "from the ground up". A sequence of bodies is added to the accumulated heap; after a body reaches a stable state, it remains in that position as additional bodies are added (this technique was used to make Figures 12, 14, 15, and 16). Automatic maintenance of the non-interpenetration constraint and the ability to place the next object stably at least provide useful tools in such a modeling task.

The approach trades off robustness in order to achieve interactivity, as compared to an approach like [SNYD93]. Problems can happen when an object contacts another at a curve or surface, where one object is concave in this contact region (e.g., a torus on a cone).[1] Such configurations can still be produced, but sometimes require user intervention (see Section 6 for further discussion). As in [BARA94], our algorithms make use of heuristics for which a formal justification is lacking, but which seem to work in practice. The main example involves converting collisions on polygonal approximations to collisions on analytic surfaces using numerical iteration. We have not found selection criteria for the approximation that provide a theoretical guarantee for convergence of this iteration. In practice though, heuristic selection of the approximation gives a workable solution.

The placement algorithm described in this paper shares characteristics with optimization methods used previously [WITK87,BARZ88,GLEI92], but specializes these methods to the problem of interactive placement. The main difficulty with direct application of such methods involves handling discontinuities in the energy functional that arise from collisions. This paper explains how to slide downhill efficiently in the presence of collisions without resorting to a full-blown and very slow physical simulation. Essentially, the main contribution of this paper is a description of which physical characteristics to sacrifice in order to get good performance, and which to keep in order to get intuitive control. Several specific elements of the approach are new: we use a new surface representation that employs both polygonal and parametric representations, a new algorithm for fast tracking, creation, and deletion of contact points, and a new method that quickly converges to a stable, non-interpenetrating placement for a single body starting from some arbitrary non-interpenetrating state and subject to a set of external forces.

2 Summary

An object is placed near its desired position such that it does not touch any other objects. This is called a *noninterfering* state. Interactive collision detection, described in Section 3, makes it easy for the user to select a noninterfering initial state. In the simplest mode of interaction, the user lets the body stabilize under the influence of a gravitational pseudo-force. The body is automatically moved through a series of discrete states, each of which obeys the non-interpenetration constraint, until a stable state is reached. An independent pseudo-force calculation is made at each state transition: external pseudo-forces are computed, components balanced by contact pseudo-forces are subtracted away, and the residual is applied to produce an incremental change in the body's position. Contact points are relocated where the body now touches other bodies; the body's position and orientation may also be slightly adjusted to preserve all contacts. The relevant algorithms are described fully in Sections 4 and 5. The result of the algorithm is the final position of the active body and a list of contact points. The complete algorithm is outlined in Figure 2.

```
place body in a noninterfering initial state
loop
    check for tracked point transitions
    compute pseudo-forces at each contact point, and on body
    update active body's placement using $\epsilon$ (solution step)
    adjust tracked points for new placement
        if unable, $\epsilon \leftarrow \epsilon/2$ and start loop again (loop failure)
    while new polygonal collision exists and collisions < $N_{\text{simultaneous}}$
        if unable to convert collision to a contact point then
            $\epsilon \leftarrow \epsilon/2$ and start loop again (loop failure)
        else
            add tracked point
        endif
    endwhile
    if collisions were successfully added, restore old value of $\epsilon$
    if objective function increases, $\epsilon \leftarrow \epsilon/2$ (loop failure)
    if loop was completed successfully, increase $\epsilon$ by $\epsilon \leftarrow \max(\lambda\epsilon, \epsilon_{\max})$
    render current state
    complete user interaction requests: interrupt, undo, direct repositioning,
        force or parameter change, etc.
until stable
```

Figure 2: Automatic Placement Algorithm: This algorithm produces a series of placements of the body that never violate the non-interpenetration condition and converge to a stable state – a state in which contact forces balance user-specified external forces such as gravity.

The sequence of state transitions generated by the automatic placement algorithm can be interrupted at any point. The user can "undo" a portion of the sequence and change the forces acting upon the body in order to reach different resting states. In addition to turning gravity on or off, the user can apply external pseudo-forces to nudge the object in any direction or make it attach to specified points on other objects. Alternatively, the body can be manually positioned relative to its current state,[2] and the automatic placement algorithm resumed from the new position. As in the case of selecting an initial state for the body, the user must choose a noninterfering state from which to resume automatic placement.

2.1 Surface Representation

A *body* is a rigidly movable solid object that interacts with other bodies. It is represented as a set of *patches*, each of which is an analytic parametric surface, whose union contains the boundary of the body. Each patch is defined so that its surface normal points toward the exterior of the body.[3]

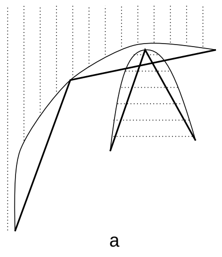
In choosing a representation for patches, we were faced with the problem that polygonal collision detection is extremely efficient, but can't be used alone to find accurate collisions between smooth surfaces. We therefore use a hybrid representation: a polygonal approximation is used to detect new points of contact that arise between bodies, and a functional description is used to adjust these points so that they lie on the actual curved surface, and to track them as they move.

The polygonal description is a mesh of triangles with (u, v) parametric coordinates per vertex. Each vertex lies on the analytic surface. The location of a collision can then be approximated by finding two intersecting triangles and, for each triangle, barycentrically interpolating the parametric coordinates at each vertex to obtain the (u, v) coordinate at the point of intersection. The two points are used as starting points in an iterative method (multidimensional Newton-Raphson) which produces contact points on the actual surfaces. The polygonal approximation is also used during manual positioning to determine whether a body is in a noninterfering state. When interference is detected, the body is made transparent and a small dot displayed at a point of intersection. The calculations take place at interactive

[1] Note that the problem arises only when the object is concave *at the contact region*. Concave objects that contact at isolated points or are concave only away from the region of contact are handled without difficulty.

[2] In the prototype system, the active body is moved by turning a series of 6 dials which represent translation and rotation around the coordinate axes.

[3] Although the system does not currently support open bodies, the changes required are straightforward. Contact points must record whether they are on surfaces, edges, or vertices. The systems of equations used in tracking must then make use of this information. Alternatively, surface boundaries can be handled by placing thin tubes around edges and spheres around vertices.



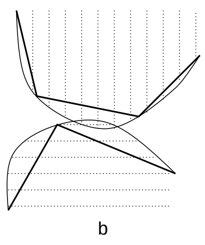


Figure 3: Collision Inconsistencies for Analytic Surfaces vs. Polygonal Approximations: (a) shows a collision between polygons but not surfaces, (b) shows a collision between surfaces but not polygons.

rates as the user moves the body. Finally, the polygonal approximation is used to select a body or a point on a body, using a ray casting algorithm as in [KAY86].

Functional descriptions for parametric surfaces are created with a symbolic language, as described in [SNYD92]. It is also possible to develop special purpose code for classes of parametric surfaces such as bicubic patches. Evaluation of surface points and derivatives up to second order is required.

Handling Inconsistencies

Since analytic surfaces are approximated by triangles for collision detection, two sorts of errors can occur as shown in Figure 3. When one of the objects is concave, a collision can happen between polygonal approximations but not the analytic surfaces (Figure 3a). This is detected by a failure in the numerical iteration to move the contact point onto the analytic surfaces (see Section 4.3) and ignored.[4] A collision between analytic surfaces can also be missed with the polygonal approximations (Figure 3b). There are three cases for what then happens: the surfaces can continue to interpenetrate until the polygonal approximations eventually collide, they can "tunnel" through each other, or they can reach a stable state in this situation. The first case is the typical one and causes no problem since the collision is eventually found. Users can ignore tunneling since the objects no longer violate the non-interpenetration condition, or undo tunneled placement states and restart the algorithm with a smaller ϵ (solution step) parameter, making it more likely a polygonal collision will be detected. The third case reduces the accuracy of the final placement: the objects violate the non-interpenetration condition by a distance bounded by the approximation errors of the two meshes. A conservative test for noninterpenetration can be developed using polygonal approximations for offset surfaces where the mesh is known to enclose the analytic surface.

Why not just use polygons?

There are three advantages of employing an analytic description of curved surfaces. First, much better accuracy can be achieved. The location of a surface is computed where it rests on points mathematically on the surface rather than on faces or edges of a faceted approximation. This can be important for CAD applications, or cases in which the camera is close to the resulting model. More accuracy is also achieved in reaching a stable state; meta-stable configurations that rest on facets are avoided.

Second, using the smooth surface makes it easier to incrementally change a body's state while enforcing the non-interpenetration condition. When one curved body can continuously slide over the surface of another, many states are unreachable when a polygonal approximation is used. Consider a cylinder approximated as an extruded regular polygon of n sides resting lengthwise on a flat plane. Only n discrete rotations of this cylinder around its axis are stable (where it rests on one of the extruded edges of the regular polygon); the real cylinder is stable for any such rotation.

Third, faster convergence to stability can be achieved. The analytic surfaces provide the exact normal vector at points of contact unavailable with a polygonal approximation. Using these normals in a force balance computation, we can move the surface a significant amount between steps. The size of the step used with a purely polygonal approximation would necessarily be tied to the size of polygonal facets in the neighborhood of the contact: an accurate approximation with many polygons would require a large number of steps.

[4]As will be discussed further in Section 4.3, such a collision is not really ignored, but is initialized as an extremal point: a point where surfaces are close but not in contact. Extremal points are tracked along with contact points. An extremal point is deleted if the separation distance increases, or is converted to a contact point if the distance becomes ≤ 0.

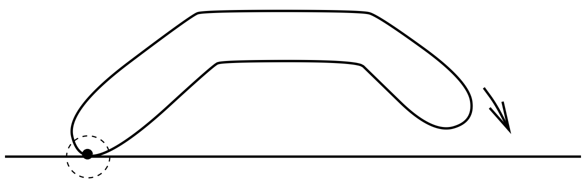


Figure 4: Spheres of neglect: A sphere is placed around current contact points within which collisions are disregarded, so that only new contact points are found.

3 Detecting Collisions Quickly

Our method for detecting collisions between triangulated bodies is similar to that presented in [GARC94], with several differences that tailor the algorithm to the interactive placement problem. See that reference for a detailed description of the collision detection problem.

We approximate each body with a set of triangles organized in an object-partitioning bounding box hierarchy as in [KAY86]. At the terminal nodes of the hierarchy are the triangles, each of which is marked with a (body,patch) identifier. Each node of the tree contains a bounding box in the form $(x_{\min}, x_{\max}, y_{\min}, y_{\max}, z_{\min}, z_{\max})$, and a list of child nodes, or information for a triangle in the case of a terminal node. Nodes can also contain information specifying a rigid motion for the subtree. In this case, the node's bounding box is in the post-transformed space; each of its child nodes store bounding boxes in local (pre-transformed) space. Bounding boxes are dynamically transformed during traversal of the hierarchy (see Section 3.2).

A similar hierarchy is then constructed for the set of bodies in the system (the *world*). The world is maintained by incrementally adjusting the active body's position in it; this is easier in an object partitioning hierarchy than in a spatial hierarchy. When a body moves, a transformation at a single node must be changed and bounding box changes propagated up the tree; the object's hierarchy remains unchanged.[5] Because a body's hierarchy is constructed just once, we can afford to use substantial computation organizing its triangles in a hierarchy that is efficient for collision detection. This processing is done before interaction begins (see Section 3.1).

Two forms of collision detection are required for interactive placement. Simple collision detection computes whether an object interferes with any other objects, and is used in choosing a noninterfering state from which to begin or resume convergence to stability. A second form of collision detection is used to compute whether a body already in contact with other bodies intersects at any additional points. To do this, we define *spheres of neglect* around the current contact points, as in [SNYD93]. During traversal, potential collisions within the spheres of neglect are discarded; the algorithm thus locates new collision points (Figure 4). The radius for each sphere is a small value determined by the distance the contact point moves from polygonal approximation to analytic surface during the contact creation process (Section 4.3). This is done because contact points are tracked on analytic surfaces but are used to discard polygonal collisions. Note that spheres of neglect are used only to ignore polygonal collisions; they do not effect contact point tracking or force calculations.

3.1 Constructing the Bounding Box Hierarchy

To construct a bounding box hierarchy for a body, we apply the recursive algorithm maketree to the flat list of triangles comprising it:

```
maketree($L$)
    partition $L$ into set of $n$ lists $L_i$
    create root node $R$
    for each $i$, insert maketree($L_i$) as child of $R$
    return $R$
```

[5]We note that lazy evaluation of bounding box changes improves efficiency in the general case. When a transformation changes, the appropriate node's dirty flag is set and propagated up the tree until the root or a node previously marked dirty is reached. At collision time, a dirty node's transformation is updated using a callback function, and its bounding box updated as the union of its children (which must be recursively updated if marked dirty). This sort of lazy evaluation avoids needless union-of-bounding-box computations as child nodes are sequentially marked dirty. A node's bounding box and transformation is updated exactly once, no matter how many of its children have changed. Of course, these subtleties are unimportant when only a single body is moved.

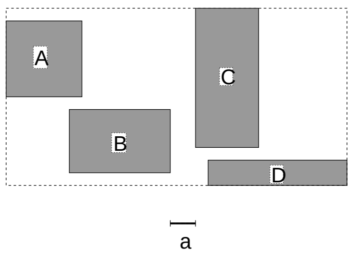


Figure 5: Object partitioning by gap finding: The bounding boxes around four objects (A, B, C, D) have a gap, a, when projected onto the horizontal axis. Partitioning around this gap yields the two lists (A, B) and (C, D). Note that the objects have no gap when projected onto the vertical axis.

At each invocation of maketree, the bounding box for the result node R is taken as the union of the bounding boxes of its child nodes.

The heart of the construction process is the partitioning of a list of nodes L into sublists L_i. Grouping nodes that are close together greatly improves culling during collision detection. We have used the following heuristic: the bounding boxes of each list element are projected into each of the three coordinate axes to form three lists of intervals. These lists are sorted in increasing order of interval lower bound, and then are searched for gaps (see Figure 5). We partition the list into two by using the coordinate axis that generates the gap of greatest width. All elements whose projected interval is less than the gap center are placed in one list, the rest in the other. If no gaps exist, which happens fairly frequently, we partition around the center of the interval in which projected bounding box centers have the greatest variance. The projection axis is chosen as the coordinate axis for which the standard deviation of interval centers is greatest. This kind of partitioning gives us a branching ratio $n = 2$ in the resulting hierarchy; other branching ratios are possible by partitioning along multiple axes. We have not tried other branching ratios.

3.2 Traversing the Bounding Box Hierarchy

Collision detection is computed by traversing the bounding box hierarchy of a pair of nodes to be collided, N_A and N_B. In our system, the active body is collided against the world, although the algorithm described here can compute collisions between any two collections of bodies or within a single collection.

Pairs of nodes from the two hierarchies are examined in depth-first order according to the following basic algorithm:

```
traversetrees($N_A$,$N_B$)
    initialize stack of active pairs with ($N_A, N_B$)
    while stack is nonempty
        pop off next pair ($A, B$)
        loop through child nodes of $A$ and $B$: ($A_i, B_j$)
            if ($A_i, B_j$) collide
                if both are triangles, record collision
                else push pair onto active list
```

Associated with each pair of nodes on the active stack is a transformation which transforms the second element of the pair into the coordinate system of the first. This transformation is updated whenever child nodes are inserted that contain transformations. Relatively few of these nodes are encountered during traversal since bodies typically contain hundreds or thousands of triangles, all of which are moved by changing one node's transformation.

Three types of collision computations occur: bounding box vs. bounding box, bounding box vs. triangle, and triangle vs. triangle. If both nodes are nonterminal, a bounding box vs. bounding box collision is done by transforming the bounding box of the second object to the coordinate system of the first, and testing whether the two bounds overlap. If only one is nonterminal, a bounding box/triangle collision test is done, again in the coordinate system of the first object. Otherwise, a test for the intersection of two triangles is done. When two triangles collide, the location of the collision and pointers to the two triangles are recorded.[6]

Several changes to the basic algorithm are needed. The first is to cull nodes based on a list of spheres of neglect. To do this, we test the intersection of the bounding boxes of A and B to see whether it is entirely inside some sphere of neglect. If so, we discard the pair. We also discard a collision between triangles if it lies within any sphere of neglect. The second change improves performance: when one node's bounding box is small with respect to the other, we do not subdivide it into its children, but compare it unsubdivided with the children of the bigger node. In our experiments, this sped up the algorithm up by factors of 2-10, the larger number attained when a very small object is collided against a much larger one.

[6]The algorithm currently records an arbitrary point of intersection between the two triangles.

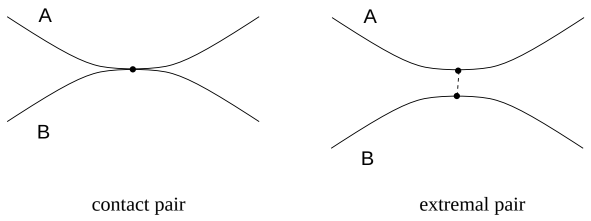


Figure 6: Types of interface points: Two non-interpenetrating bodies in close proximity, A and B, can have two types of interface points. On the left, the bodies are in tangent contact at a contact pair. On the right, the bodies are slightly apart; the pair of points at minimal distance is the extremal pair.

4 Computing Points of Contact

As in [SNYD93], places where bodies interact are handled using a finite set of points even when the region of contact forms a curve or surface. In [SNYD93], points were uniformly distributed over the contact region; the approach here creates only enough points to prevent interpenetration and reach a stable state. These points are called *interface points* and consist of a pair of points on two bodies. Interface points are tracked (incrementally updated) as the body moves from state to state using numerical iteration. During collision detection, a sphere of neglect is placed around each interface point to avoid detecting collisions already being handled. We therefore store, for each interface point, a (u, v) parametric coordinate pair, a pair of (body,patch) identifiers for the surfaces in contact, and a radius for the sphere of neglect.

Two types of interface points are used: *contact pairs* and *extremal pairs*, shown in Figure 6. Bodies actually touch at contact pairs. Forces that balance external forces such as gravity are applied at the contact pairs. These forces are only applied to push objects away from interpenetrating, not to "glue" objects together. When such a gluing force is obtained, it is not applied but instead causes the contact pair to transition to an extremal pair.

Extremal pairs are points on a pair of bodies at minimal distance. The bodies do not contact in a neighborhood around the extremal pair. Forces are not applied at extremal pairs. Extremal pairs allow interface points to vanish gradually. When a gluing contact force is detected, the contact pair is converted to an extremal pair, allowing the two bodies to separate. Contact can quickly be resumed if the separation distance becomes negative. Extremal pairs are deleted when the separation distance becomes large enough.

4.1 Tracking Contacts

After the active body is moved, the automatic placement algorithm must track the interface points from their previous positions. The following two-phase tracking algorithm yields excellent numerical stability. The first phase, called the *conditioning phase*, is not necessary from a theoretical perspective, but makes the resulting system easier to solve in the next phase.[7] In this phase, each interface point, represented as a pair of (u, v) parametric locations, is independently adjusted to satisfy the extremal point conditions using numerical iteration (the equations are described in Section 4.4). The initial condition for this iteration is the parametric location of the interface point before the body was moved.

In the second phase, called the *contact adjustment phase*, we adjust both the body's rigid motion parameters and the locations of all the interface pairs to preserve the contact or extremal conditions. Numerical iteration takes place to simultaneously satisfy the extremal or contact conditions for all interface points. Initial conditions for this iteration are the body's current placement, and the parametric locations of the interface points after the conditioning phase. Note that the second phase may slightly alter the body's

[7]We are able to move bodies much more between states and still maintain the contact conditions with the use of the conditioning phase.

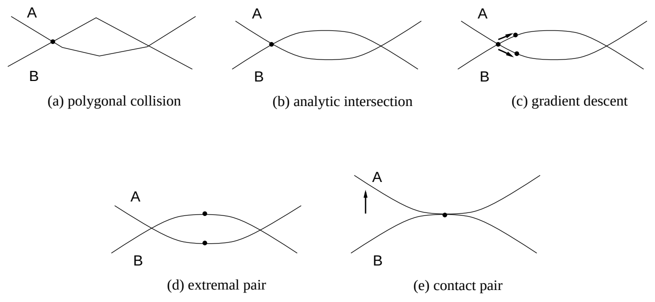


Figure 7: Steps in converting a polyhedral collision into a contact point.

placement in order to satisfy the contact point conditions.

When the numerical iteration in either tracking phase fails, the placement algorithm tries a smaller move of the active body from its last correct state (Figure 2).

4.2 Detecting Contact Transitions

Before a body is moved, the placement algorithm checks for transitions of contact points. When a transition is detected, the interface point is either deleted in the case of a vanishing transition, or switched between contact and extremal. The following transitions (with their conditions for occurrence) are detected:

1. *extremal to contact* – separation distance between bodies becomes negative
2. *contact to extremal* – contact force attracts rather than repels the bodies
3. *extremal vanishes* –
    - separation distance exceeds threshold
    - conditioning phase iteration fails
    - point becomes too close to another interface point
    - point falls off parametric domain of its patch
4. *contact vanishes* –
    - point becomes too close to another interface point
    - point falls off parametric domain of its patch

The user can optionally disable contact to extremal transitions to assure that the active body will stay in contact as it's moved. This effectively makes the selected contacts "sticky".

4.3 Creating Contacts

The automatic placement algorithm converts collision points on polygonal bodies to contact points on analytic parametric surfaces. A series of steps, illustrated in Figure 7, is performed, each of which uses the results of the previous step as the initial condition in a numerical iteration:

1. Interpolate the polygonal collision point to yield a pair of parametric points (u_1, v_1, u_2, v_2). The (body,patch) identifier for each of the intersecting triangles allows retrieval of the appropriate surface functions used in the following steps (Figure 7a).
2. Iterate onto the analytic intersection (Figure 7b).
3. Do a few steps of gradient descent to nudge the pair toward the points of maximal penetration (Figure 7c). This is necessary so that the next step iterates in the right direction (i.e., toward maximal penetration rather than maximal separation). The objective function is defined in Section 4.4, Equation 2.
4. Iterate onto the extremal pair (Figure 7d). This is a conditioning step exactly like the conditioning phase of tracking.
5. Iterate onto a contact pair (Figure 7e). In this step, the active body's position parameters and the locations of all interface points are adjusted. Equations for all interface points must be satisfied simultaneously, since the active body's placement affects the contact conditions of the previous interface points.

Steps 2, 4, and 5 involve numerical iteration which can fail. A failure in Step 2 may occur because the polygonal approximations intersect but the analytic surfaces do not (refer to Figure 3a). If failure occurs, we try iterating onto an extremal pair. If this succeeds, and the resulting separation distance (defined in Section 4.4) is positive, the interface point is added as an extremal pair rather than a contact pair. This allows the polygonal collision to be ignored during the next collision detection query. If a failure in Step 4 occurs, we disregard what amounts to a conditioning step, and go on to step 5. A failure elsewhere, or after taking the above measures, results in a collision failure and causes the placement algorithm to try a smaller solution step in its next iteration. After a successful contact point creation, the sphere of neglect radius is determined by the distance the point moved from Step 1 to Step 5. A collision failure is returned if this distance exceeds a user-settable threshold.

4.4 Contact Point Iteration

Interface point tracking and contact point creation both use multidimensional Newton iteration on a system of equations (see [PRES86] for a complete description).[8] Given a system of nonlinear equations $F(x) = 0$ and a point near the solution, x_0, a successive approximation to the true solution is achieved by solving the linear system of equations

$$0 = F(x_0) + \frac{\partial F}{\partial x}(x_0)(x_1 - x_0) \tag{1}$$

where $\partial F/\partial x$ is the Jacobian of F. We solve the above linear system using the singular value decomposition (or SVD, see [PRES86]), which allows non-square systems to be solved and is numerically robust.[9] The process is then repeated to improve the approximation, yielding a sequence of iterates x_i. The iteration fails if $F(x_i)$ does not get closer to 0 or a maximum number of iterations is exceeded without yielding a point sufficiently close (in the residual sense) to a solution.

It only remains to describe the relevant systems of equations (F from Equation 1), and their parameters, x, for each iterative procedure. In the following, we have two points each on a rigidly movable surface $S_i(Q_i, X_i, u_i, v_i)$, $i = 1, 2$, where Q_i is a rotation in 3D[10] and X_i is a translation in 3D:

$$S_i(Q_i, X_i, u_i, v_i) \equiv Q_i\, s_i(u_i, v_i) + X_i$$

$s_i(u_i, v_i)$ represents the surface in its local coordinate system. There are three systems of equations used in the above Newton iteration:

1. intersection [iteration over (u_1, v_1, u_2, v_2)]:

   $$S_1 - S_2 = 0$$

   The pair of parametric points is moved to become a true intersection of the bodies, used in Step 2 of the contact point creation sequence. Note that this is a non-square system of equations: 3 equations in 4 variables, which is handled using SVD-augmented Newton iteration.

2. extremal [iteration over (u_1, v_1, u_2, v_2)]:

   $$\begin{aligned} (S_1 - S_2) \times N_1 &= 0 \\ N_1 + N_2 &= 0 \end{aligned}$$

   where N_1 and N_2 are the outward-pointing unit normals of the two surfaces. These conditions imply that the distance between the point pair is a local extremum (the vector between the points is in the direction of one surface's normal, and the two normals are anti-parallel). If the bodies are separated by a small distance, the iteration will cause the two points to become the points of minimum distance between the two surfaces. If the bodies interpenetrate, the iteration will cause the two to become the points of furthest penetration. This iteration is used in tracking and in Step 4 of the contact point creation sequence.

[8] We use the numerical package LAPACK to compute the SVD.

[9] To solve the linear system $Ax = b$, the SVD of matrix A is first computed. This yields three matrices whose product is A, $A = UDV^T$, where U and V are orthonormal, and D is a diagonal matrix. An ill-conditioned system is adjusted by setting to 0 those elements for which $D_i/D_{\max} < \delta$, where $D_{\max}$ is the largest diagonal element. We then compute $x = VD^{-1}U^T$ taking into account the rank of D, since some elements have been set to 0. The result is a numerically robust solution that minimizes the solution, $||x||$, if solutions exist, and minimizes the residual, $||Mx - b||$, if not.

[10] We use a unit quaternion, $Q = (q_1, q_2, q_3, q_4)$, to represent this rotation. The coordinate system origin is at the center of mass of the body.

3. contact [iteration over $(Q_1, X_1, u_1, v_1, u_2, v_2)$]

$$\begin{aligned} S_1 - S_2 &= 0 \\ N_1 + N_2 &= 0 \end{aligned}$$

We change the position and orientation of the first body as well as the parametric location of the point of contact in order to achieve a true contact point where the surfaces touch and are tangent. This iteration is used in tracking and in Step 5 of the contact point creation sequence.

Note that the position and orientation of the second body (Q_2, X_2) is held constant for all systems of equations.

Contact point creation also involves gradient descent iteration in Step 3. The objective function whose gradient is descended, the *signed separation distance function G*, is given by

$$G \equiv (S_2 - S_1) \cdot N_1 \tag{2}$$

If the bodies don't touch, this function is positive within a neighborhood of the points of maximal separation. If the bodies interpenetrate, the function is negative inside the region of interpenetration.

5 Converging to a Stable Configuration

Convergence to a stable state is achieved by the application of pseudo-forces that cause an incremental change in the position of the active body. Using force-like quantities accomplishes two things: it leads to changes in object placement that the user can predict, and it provides much faster convergence to stable placement than other update rules.[11] Pseudo-forces differ from physical forces because they are solved for statically and do not accelerate the object but are applied directly to update the object's position (see Section 5.3).

Three kinds of pseudo-forces are used: *external*, which represent gravity and user-specified "nudges", *contact*, which are applied at contact pairs to balance the external forces as much as possible, and *residual*, which represent the resulting force (external - contact) applied to adjust the body's position. The body is considered to be at rest and automatic placement halted when the residual force is sufficiently small.

It sometimes happens that exact force balance can not be achieved.[12] We therefore use a second criterion to determine stability: the *psuedo potential energy objective*. The placement algorithm tries to minimize the energy objective: a state that increases the objective is rejected. When gravity is the sole external force, the energy objective is simply the height of the center of mass of the active body. The placement algorithm thus drops the object as far as possible. When the change in energy objective is sufficiently small, the automatic placement algorithm is terminated.[13] Other forces have different objectives, discussed in Section 5.1. The total energy objective is the sum of the energy objectives for all external psuedo-forces.

5.1 Computing External Pseudo-Forces

The user can apply combinations of three types of external pseudo-forces to the active body: *gravity*, *local*, and *connection* forces (Figure 8). Pseudo-forces from these three categories can be added or deleted at any time during automatic placement to position the active body. For each external pseudo-force, the algorithm computes three quantities: a force (F) and torque (T) on the body, and an objective term (E). These are summed to produce the total force, torque, and objective.

In the following, the current location of the active body is represented by the rotation/translation pair (Q, X), as in Section 4.4. Force parameters in capital letters are in world coordinates; noncapitalized parameters are in body coordinates. External pseudo-forces always have unit length; the

[11] For example, to stabilize an object under the influence of gravity, the first update rule we tried was to translate the body down in z (direct gradient descent). Iteration was then done to satisfy the interface point conditions, which typically moves the body back up in z. In many experiments this algorithm was more than 10 times slower then the one proposed, which balances gravity with an approximation to the contact forces.

[12] For example, consider dropping a torus onto a flat ground plane such that the contact region in the resting state forms a circle. The approach advocated here will create a number of contact points whose configuration depends on the polygonal approximation used for the torus. It can easily happen that two contact points arise that do not form an exact diameter of the circle of contact. Any assignment of contact forces at these points will therefore produce a residual torque (note that the contact forces point up, normal to the plane).

[13] Using the torus/plane example, this termination criterion guarantees that the torus comes to rest in a stable state as soon as more than one contact point is generated. This is because it can no longer move in z; the energy objective remains unchanged, triggering the termination condition.

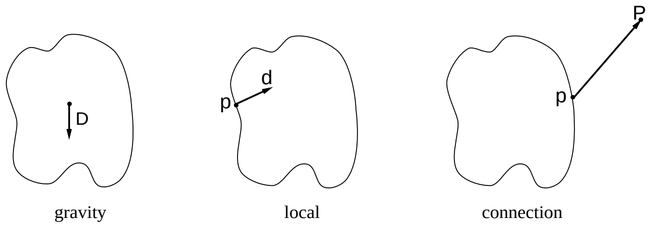


Figure 8: Types of external pseudo-forces.

amount of movement is determined by the body movement step (ϵ from Figure 2). Length measurements are scaled relative to the radius of the smallest enclosing sphere around the active object, R.

A gravity pseudo-force is applied at the body's center of mass in a given direction D (e.g., $D = (0, 0, -1)$). It thus produces no torque, and tends to translate the body along the D direction. The force, torque, and objective terms for this pseudo-force are given by:

$$\begin{aligned} F_{\text{gravity}} &\equiv D \\ T_{\text{gravity}} &\equiv 0 \\ E_{\text{gravity}} &\equiv -(D \cdot X)/R \end{aligned}$$

A local pseudo-force is applied at a given point on the active body, p, in some direction d in its local coordinate system. It allows the user to push and pull on the body. The force, torque, and objective terms are:

$$\begin{aligned} F_{\text{local}} &\equiv Qd \\ T_{\text{local}} &\equiv (Qp)/R \times F_{\text{local}} \\ E_{\text{local}} &\equiv 0 \end{aligned}$$

To specify a local pseudo-force, the user picks a point p on the active body and specifies a "push" or "pull" force. A push force assigns to d the negative of the unit normal vector at p; a pull force assigns it the unnegated normal vector. The user can also directly control the direction with a 3D widget. Note that the objective term for a local pseudo-force is 0. In fact, the user typically turns off objective function processing when using local pseudo-forces, so that objects can be moved freely against gravity or other external forces without causing a loop failure when the objective function increases (refer to Figure 2).[14]

A connection pseudo-force is applied at a local point on the body, p in order to connect that point to a given point in world space P. The force, torque, and objective terms are:

$$\begin{aligned} F_{\text{connect}} &\equiv \frac{P - (Qp + X)}{\|P - (Qp + X)\|} \\ T_{\text{connect}} &\equiv (Qp)/R \times F_{\text{connect}} \\ E_{\text{connect}} &\equiv \|P - (Qp + X)\|/R \end{aligned}$$

5.2 Solving for Contact and Residual Pseudo-Forces

Let the total external pseudo-force and pseudo-torque on the body be given by F_e and T_e. The contact pseudo-forces are given by n scalars f_i where n represents the number of contact pairs. The direction of these forces is given by N_i where N_i is a unit vector representing the negative of the normal vector on the active body at the point of contact. To find the f_i's, we solve the linear system of force balance equations given by:

$$\begin{aligned} \sum_{i=1}^{n} f_i \, N_i &= F_e \\ \sum_{i=1}^{n} f_i \, (p_i \times N_i) &= T_e \end{aligned}$$

where p_i is the vector from the body's center of mass to the contact point. This system has no solution if the the body isn't in a balanced state. We solve using SVD to obtain a solution which minimizes the L^2 norm of the residual.

[14] In a typical placement scenario, the user first lets a body drop under the influence of gravity. When it is stable, objective function processing is turned off and local pseudo-forces are used to slide the contacting body where it's desired. Then local pseudo-forces can be deleted and objective processing re-enabled to reach a truly stable placement.

While the contact forces produced in this way are a coarse approximation, unstable objects still fall over much as they might in the real world.

To solve for the residual pseudo-force and pseudo-torque, F_r and T_r, we subtract the sum of the previously computed contact forces from the external pseudo-forces and pseudo-torques:

$$\begin{aligned} F_r &\equiv F_e - \sum_{i=1}^{n} f_i\, N_i \\ T_r &\equiv T_e - \sum_{i=1}^{n} f_i\, (p_i \times N_i) \end{aligned}$$

The termination criterion $\|F_r\| < \delta, \|T_r\| < \delta$ stops the body when it is balanced with respect to purely normal contact forces. Note that this criterion implies the body is *physically* stable, since a static force balance has been achieved.

The user may also want to stop the body when it is stable with respect to frictional forces.[15] To do this, we compute a frictional pseudo-force and pseudo-torque residual, F_f and T_f, by first solving the frictional force balance problem

$$\begin{aligned} \sum_{i=1}^{n} f_i^U\, U_i + f_i^V\, V_i &= F_r \\ \sum_{i=1}^{n} f_i^U\, (p_i \times U_i) + f_i^V\, (p_i \times V_i) &= T_r \end{aligned}$$

where U_i and V_i are independent tangent vectors at each contact point, and f_i^U and f_i^V are the $2n$ frictional forces along these directions to be solved for. This problem is solved using SVD and the resulting minimal L^2 solution subtracted from F_r and T_r respectively to yield the new residual F_f, T_f. This approximation to the frictional force and torque is simpler, and less physical, than used in [BARA94].

When allowing friction forces in the termination criteria, we make the additional constraint that the magnitude of the frictional pseudo-force at each contact must be less than a fixed ratio of the magnitude of the normal pseudo-force (Coulomb model of friction).[16] The final pseudo-force and pseudo-torque applied to the active body, F_N and T_N, is given by

$$\begin{aligned} F_N &\equiv (1-\alpha)F_r - \alpha F_f \\ T_N &\equiv (1-\alpha)T_r - \alpha F_f \end{aligned}$$

where α is a user-specified constant related to the amount of friction desired.

5.3 Updating Body Placement

The active body's position is updated by rotating through a small angle θ in radians around an axis A and translating through a small displacement Δ. The total amount of change is governed by the scale-invariant parameter ϵ – the maximum distance to move the active body in one step, relative to the radius of its smallest enclosing sphere, R. Actual values for the update parameters are obtained from the rigid body equations of motion assuming the body was at rest before applying forces, which implies that the linear and angular velocities of the body are 0.

We first scale the pseudo-forces and pseudo-torques to be applied, F_N and T_N, by the sum of their lengths so that roughly the same amount of change happens at each step:

$$\begin{aligned} \hat{F}_N &\equiv \frac{F_N}{\|F_N\| + \|T_N\|} \\ \hat{T}_N &\equiv \frac{T_N}{\|F_N\| + \|T_N\|} \end{aligned}$$

Since for small ϵ, $\sin(\epsilon) \approx \epsilon$, normalizing by $\|F_N\| + \|T_N\|$ implies that the amount of movement of a point on the active body's enclosing sphere due to both rotation and translation, scaled by $1/R$, will be roughly ϵ.

[15]This allows a body to lean on others even though external forces are unbalanced by purely normal contact forces.

[16]Note that an assignment of friction forces that produces a very small residual and does not violate the Coulomb constraint may exist, yet we may not find it. To find such a friction force assignment would require solving a quadratic optimization problem. In practice, we frequently find an acceptable force assignment with this simple method. The user can also stop the system at any time manually.

experiment	time	iterations
torus on cone	9.89	13
block into goblet	8.0	25
goblet sideways on ground	6.34	28
knot on ground	3.86	12
jack on ground	6.6	36
bumpy sphere on ground	3.43	14
torus on ground	10.72	37
cone on ground	16.15	23
teapot on ground	6.89	20
lid on teapot	51.23	29
straw in glass	14.03	40
ice cube in glass	13.78	39
44th ball in bowl	6.58	9
bumpy sphere on cone	21.26	50
apple on ground	3.1	12

Figure 9: Results for Various Experiments. Each experiment was run until the placement algorithm terminated. Time is clock time in seconds on a SGI Indigo Extreme 2 workstation (R4400 150MHz processor). Iteration counts are the number of main loops, including failure loops, from the placement algorithm in Figure 2.

We then derive the linear and angular accelerations from the normalized force and torque. The linear acceleration, a, is given by

$$a \equiv \hat{F}_N$$

assuming the body has unit mass. The derivative of the angular velocity, $\dot{\omega}$, (under the at-rest assumption) is given by

$$\dot{\omega} \equiv I^{-1}\, \hat{T}_N$$

where I is the inertia tensor of the body in world coordinates.[17]

A differential update is obtained through direct use of the linear and angular accelerations, scaled by ϵ, to yield

$$\begin{aligned} \Delta &\equiv \epsilon\, R\, a \\ \theta &\equiv \epsilon\, \|\dot{\omega}\| \\ A &\equiv \frac{\dot{\omega}}{\|\dot{\omega}\|} \end{aligned}$$

This provides a "memoryless" change to the body placement, akin to simulation in an extremely viscous fluid. Note that Δ is scaled by R since it is a length parameter in world coordinates. As before, the body state is encoded by a quaternion/vector pair (Q, X). After moving by ϵ, the new body state, (Q', X'), is given by

$$\begin{aligned} Q' &\equiv (\sin(\theta/2), \cos(\theta/2)\, A) \cdot Q \\ X' &\equiv X + \Delta \end{aligned}$$

where $\cdot$ denotes quaternion multiplication.

6 Results

Figure 9 shows performance results for some simple experiments. In these experiments, time per iteration on an SGI Indigo Extreme 2 workstation varied from 0.1 seconds to several seconds. A few tens of iterations are typically necessary to stabilize an object from a nearby placement. Proportionally more iterations are required when an object must traverse a circuitous route over many bodies and contacts. The majority of the computation time (more than 90% excluding rendering) is consumed by collision detection and numerical iteration involved in contact point creation and tracking.

Figure 1 and Figures 10-16 show some results of the placement algorithm. The placement tool allows the user to replicate the active body, making it easy to place multiple instances like the ice cubes or balls in Figures 14 and 15. Parameter values used in both the performance experiments and the modeling examples were $\lambda = 1.33$, $N_{\text{simultaneous}} = 3$, $\alpha = 0$, and $\epsilon_{\max} = 0.05$. The recovery parameter, λ from Figure 2, was chosen so that the solution step size recovers somewhat more slowly than it is refined.[18] Using $N_{\text{simultaneous}} = 3$ handles the vast majority of simultaneous collisions,

[17]The inverse of the inertia tensor I^{-1} in world coordinates is equal to $QI_b^{-1}Q^{-1}$ where Q is the rotation of the body, and I_b^{-1} is the inertia tensor in body coordinates. I_b is appropriately scaled so that the body has unit mass. Note that an accurate inertia tensor is often not required for computer graphics applications in which only the appearance of stability matters. The identity tensor often suffices.

[18]Recall from Figure 2 that ϵ is halved if a loop failure occurs.

but when regions of contact form curves or surfaces, allows the algorithm to check for stability without distributing points over the entire contact manifold. The value $\epsilon_{\text{max}} = 0.05$ is a good compromise between speed of convergence and predictability of movement. User setting of ϵ_{max} is sometimes useful to increase the speed of convergence or progress more slowly through a portion of the optimization in which there is an undesirably large change in the body's placement.

Polygonal tessellations of surfaces were computed by uniform sampling in parameter space. The sampling density was typically chosen so that the surfaces appeared mostly free of polygonal artifacts when viewed from distances convenient for modeling. For example, the sphere used in the test of Figure 15 was tessellated using a 41×21 mesh, the bowl with a 81×121 mesh. We have reason to believe that the algorithm functions over a wide range of mesh accuracy. In one experiment, a knot-like shape was repeatedly dropped from the same position on a plane, using successively coarser uniformly sampled polygonal meshes. A stable placement involving three contact points was achieved using meshes containing from 9600 triangles to 56 triangles. Time to convergence varied for these experiments varied from 7.27 seconds to 6.12 seconds. Below 56 triangles (mesh size 15×5), the mesh was too poor an approximation to allow convergence of polygonal collisions to analytic contacts. A reasonable deterministic heuristic for tessellation would be to bound a measure of the approximation error such as maximum length of deviation; we have not investigated such a heuristic.

The placement algorithm has been surprisingly robust in our experiments: almost all modeling tasks in the figures were performed without interrupting the algorithm and without changing parameters from their default values. Occasional problems do occur. The algorithm can get "stuck" when it is not able to convert a polygonal collision to a contact point, so that ϵ goes to 0. This happened in certain experiments when dropping a torus over a cone and the lid over the teapot. Both situations involve bodies that form a curve of contact at the stable configuration, where one body is concave at the contact. The problem occurs after one contact has been created. As the body approaches the stable state, additional polygonal collisions are found and converted to analytic contact points. The iteration often moves the contact point a significant distance over the surface, violating the maximum distance threshold. If the distance threshold is increased, the large radius of neglect for the point may cause missed collisions. If the radius of neglect is manually decreased, too many polygonal collisions can be computed, slowing the algorithm to a crawl.

With additional intervention though, modeling tasks can still be performed in these situations. When the problem occurs, the user can interrupt the placement algorithm. Increasing the distance threshold parameter and then resuming often solves the problem, though the user must watch out for missed collisions. Collision detection can also be temporarily disabled after interruption and convergence attempted with the current contacts. Again, missed collisions are a possibility. Since the problems often occur very near to the stable position, a third strategy is to interactively reposition the object slightly from its stuck position and try again. If the problem is not corrected or results in missed collisions, the user can undo the bad states and try different strategies. The desired placement is usually achieved after two or three attempts. Such measures, although clearly not ideal, may not be intolerable in an interactive modeling environment.

As might be expected, the algorithm also suffers from convergence to meta-stable placements. For example, when a sphere is dropped onto another precisely underneath, the algorithm converges with the sphere balanced on top. This is easily remedied by interactively giving the sphere a nudge (by applying a local force for one step) and resuming.

7 Conclusion

Placing curved objects in physically plausible configurations has always been a difficult task for modeling systems, but one that can add much visual richness. This paper describes a new tool for interactive placement of non-interpenetrating curved objects that makes this task easier and more accurate. Two ideas make such a tool practical. The first is the use of a hybrid surface representation. A polygonal approximation allows quick detection of contacts that arise; a functional description converts these using numerical iteration to accurate points where the analytic surfaces touch. The second idea is to reach stability using an optimization technique that passes through a discrete series of states, based on a simple static force balance law rather than a dynamic simulation. This paper describes a way of moving a body toward a stable state which is predictable and fast.

A number of areas for extending this work remain. Handling multiple active bodies is a straightforward extension. A slightly more general contact force solver is needed; the collision detection and interface point tracking algorithms described here require no modification. We believe such an extension would be practical in an interactive setting only for a limited number of active bodies. For example, simultaneously manipulating all 44 balls from the model in Figure 15, including well over a hundred tracked contact points, does not seem practical for the near future. Nevertheless, the extension would be useful for placement of mechanical linkages containing a few curved parts. Handling interactions between rigid and flexible bodies may also be possible. Another problem we have only begun to investigate is placement from an interfering initial state to allow a tool that can extricate a penetrating body. Finally, we are investigating ways to automatically handle the problem case discussed in Section 6 without resorting to user intervention.

Acknowledgements

I wish to thank Al Barr for getting me interested in this topic, and for his many insights into physical simulation. I also owe thanks to Jim Kajiya for many helpful discussions about the role of physical simulation in object assembly. James O'Brien wrote most of the collision detection code. Michael Cohen provided an extremely valuable critical read of the document as did Jim Kajiya. I also acknowledge Bena Currin, Adam Woodbury, Kurt Fleischer, and the other members of Caltech's graphics group for geometric models used in this work and for helpful early discussions. This work wold not have been possible without the support of Microsoft, especially my manager Dan Ling.

References

[BARA90] Baraff, David, "Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation," Computer Graphics, 24(4), pp. 19-28, August 1990.

[BARA91] Baraff, David, "Coping with Friction for Non-penetrating Rigid Body Simulation," Computer Graphics, 25(4), pp. 31-39, July 1991.

[BARA94] Baraff, David, "Fast Contact Force Computation for Nonpenetrating Rigid Bodies," Computer Graphics, 28(2), pp. 23-42, July 1994.

[BARZ88] Barzel, Ronen, and A. Barr, "A Modeling System Based On Dynamic Constraints," Computer Graphics, 22(4), pp. 179-188, August 1988.

[GARC94] Garcia-Alonso, Alejandro, N. Serrano, and J. Flaquer, "Solving the Collision Detection Problem," IEEE Computer Graphics and Applications, pp. 36-43, May 1994.

[KAY86] Kay, Tim, and K. Kajiya, "Ray Tracing Complex Scenes," Computer Graphics, 20(4), pp. 269-278, August 1986.

[GLEI92] Gleicher, Michael, and A. Witkin, "Through-the-Lens Camera Control," Computer Graphics, 26(2), pp. 331-340, July 1992.

[META92] Metaxas, Dimitri, and D. Terzopoulos, "Dynamic Deformation of Solid Primitives with Constraints," Computer Graphics, 26(2), pp. 309-312, July 1992.

[MOOR88] Moore, M. and Wilhelms, J., "Collision Detection and Response for Computer Animation," Computer Graphics, 22(4), pp. 289-298, August 1988.

[PRES86] Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes,* Cambridge University Press, Cambridge, England, 1986.

[SCLA91] Sclaroff, Stan, and A. Pentland, "Generalized Implicit Functions for Computer Graphics," Computer Graphics, 25(4), pp. 247-250, July 1991.

[SNYD92] Snyder, John, and J. Kajiya, "Generative Modeling: A Symbolic System for Geometric Modeling," Computer Graphics, 26(2), pp. 369-378, July 1992.

[SNYD93] Snyder, John, A. Woodbury, K. Fleischer, B. Currin, and A. Barr, "Interval Methods for Multi-point Collisions between Time-Dependent Curved Surfaces", Computer Graphics, 27(2), pp. 321-334, Aug. 1993.

[WITK87] Witkin, Andrew, K. Fleischer, and A. Barr, "Energy Constraints on Parameterized Models," Computer Graphics, 21(4), pp. 225-232, July 1987.

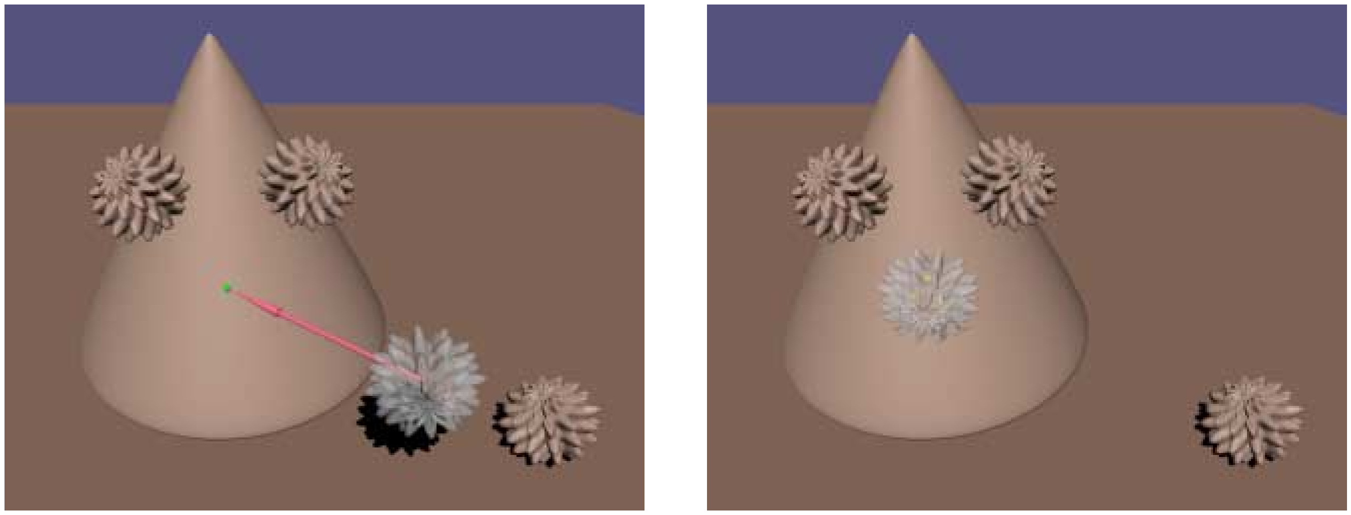


Figure 10: Sticking bodies together with connection forces.

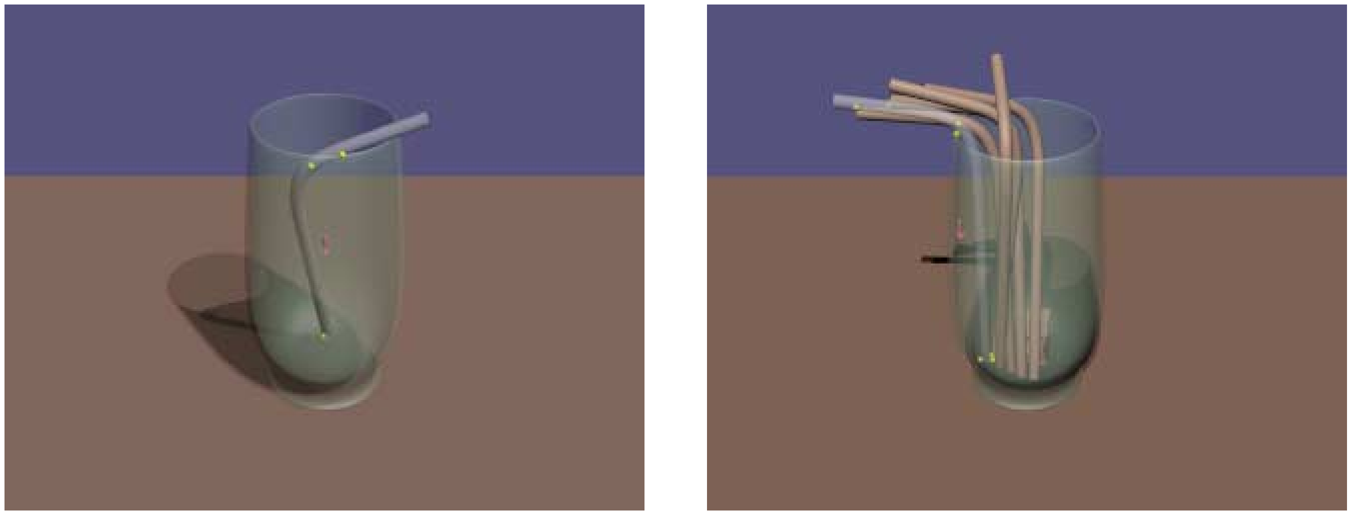


Figure 11: Dropping straws into a glass.

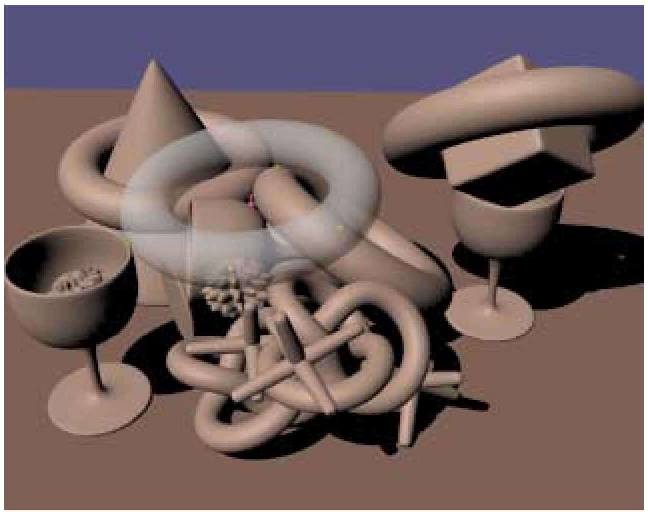


Figure 12: Cluttered scene.

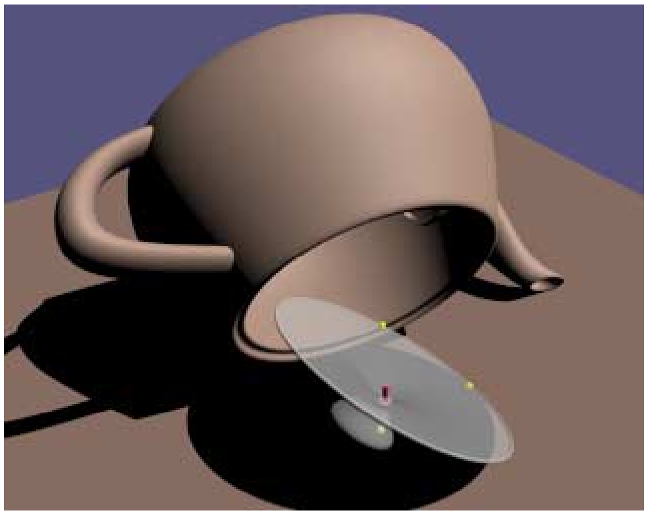


Figure 13: Fallen teapot.

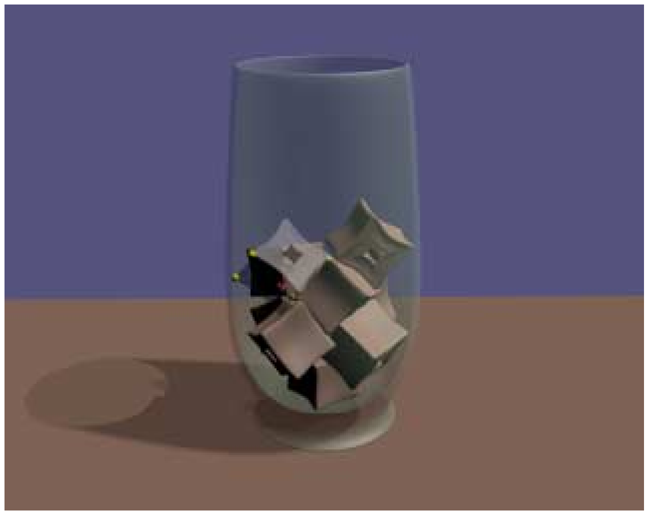


Figure 14: Ice in a glass.

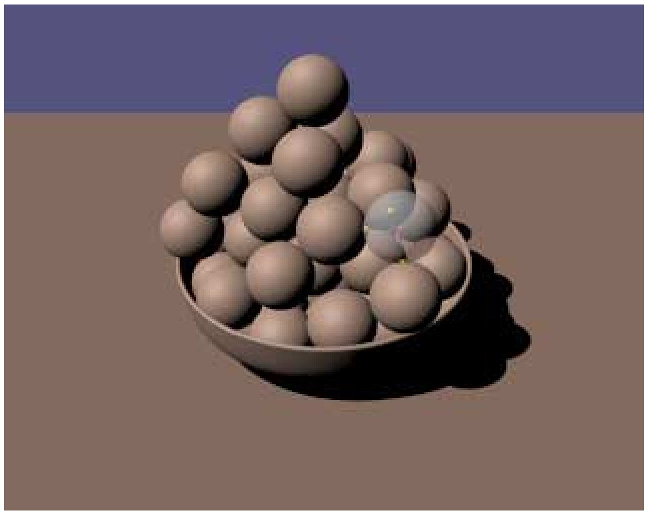


Figure 15: 44 balls in a bowl.

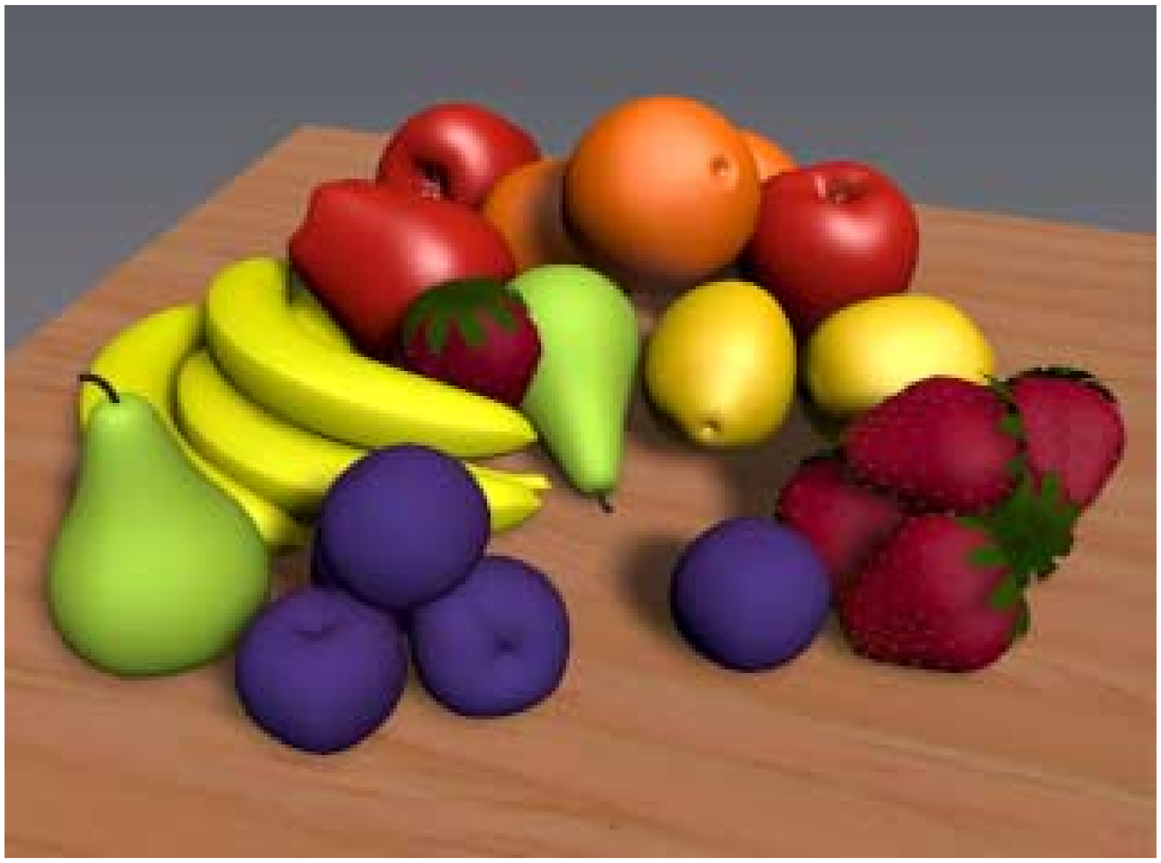


Figure 16: Still life with fruit.