***Proceedings of SIGGRAPH '96*, Computer Graphics Proceedings, Annual Conference Series, August 1996.**

Position-Based Physics: Simulating the Motion of Many Highly Interacting Spheres and Polyhedra

Victor J. Milenkovic*

University of Miami

Abstract

This paper proposes a simplified *position-based* physics that allows us to rapidly generate "piles" or "clumps" of many objects: local energy minima under a variety of potential energy functions. We can also generate plausible motions for many highly interacting objects from arbitrary starting positions to a local energy minimum. We present an efficient and numerically stable algorithm for carrying out position-based physics on spheres and non-rotating polyhedra through the use of linear programming. This algorithm is a generalization of an algorithm for finding tight packings of (non-rotating) polygons in two dimensions. This work introduces linear programming as a useful tool for graphics animation. As its name implies, position-based physics does not contain a notion of velocity, and thus it is not suitable for simulating the motion of free-flying, unencumbered objects. However, it generates realistic motions of "crowded" sets of objects in confined spaces, and it does so at least two orders of magnitude faster than other techniques for simulating the physical motions of objects. Even for unconfined objects, the new algorithm can rapidly generate realistic "piles" and "clumps."

CR Descriptors: G.1.6 [**Numerical Analysis**]: Optimization - *Linear Programming*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling - *Physically based modeling*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - *Animation*; I.6.8 [**Simulation and Modeling**]: Types of Simulation - *Continuous*.

1 Introduction

To generate realistic animation, recent work in computer graphics has focused on methods to simulate the motion of objects under the laws of physics. Suppose one wants to create an animated dancer. Instead of laboriously choosing a sequence of poses, one creates a model of a dancer with masses, joints, and forces, and lets the laws of physics do the dancing. The laws of physics are well understood (for this domain), and current computers can simulate physics for these types of models in near to real time. The main difficulty of this approach is choosing a set of forces (parameters for the model) that allow the dancer to dance and not fall on its face. However, there are other domains for which the forces are easily determined but the physics is very difficult to simulate. Consider the problem of generating a "pile" or "clump" of many objects under gravity or mutually attractive forces. Creating or even verifying an equilibrium state of the physical system is a complex problem. Consider further the problems of animating the sand in an hourglass or the sand on the beach as someone sets foot on it. Consider even the problem of animating the molecules of fluid in a lava lamp. These models involve many highly interacting three-dimensional objects. For even a modest number of grains of sand or molecules, the simulation outstrips our computational resources.[1]

This paper proposes a simplified *position-based* physics that allows us to rapidly generate "piles" or "clumps" of many objects: local energy minima under a variety of potential energy functions. Position-based physics allows us to rapidly generate plausible motions for sets of many highly interacting objects. We present an efficient and numerically stable algorithm, based on linear programming, for carrying out position-based physics on spheres and non-rotating polyhedra. This algorithm is a generalization of an algorithm for finding tight packings of (non-rotating) polygons in two dimensions. This work introduces linear programming as a useful tool for graphics animation. As its name implies, position-based physics does not contain a notion of velocity, and thus it is not suitable for simulating the motion of free-flying, unencumbered objects. However, it generates realistic motions of "crowded" sets of objects in confined spaces, and it does so at least two orders of magnitude faster than other techniques for simulating the physical motions of objects. Even for unconfined objects, the new algorithm can rapidly generate realistic "piles" and "clumps."

Section 2 compares position-based physics to other methods of physical simulation such as velocity-based contact force methods. It describes two problems which severely slow down velocity-based methods: local and global "rattling." Section 3 gives the formal definition of position-based physics and gives an algorithm to carry it out. For a system of translating objects that involves only sphere-sphere and sphere-polyhedron contacts, it is proved that the linear programming based algorithm converges to an equilibrium of the potential energy function: the algorithm cannot "stick" at a non-equilibrium configuration. Section 4 describes how to implement the algorithm using linear programming to simulate position-based physics on a set of spheres inside a polyhedron, in particular, a set of 1000 spheres inside an "hourglass". Section 5 shows how to handle non-gravitational potential functions such as attraction among

[1] Department of Mathematics and Computer Science, University of Miami, P.O. Box 249085, Coral Gables, FL 33124. The research of Victor J. Milenkovic was funded by the Textile/Clothing Technology Corporation from funds awarded to them by the Alfred P. Sloan Foundation and by NSF grants CCR-91-157993 and CCR-90-09272.

[1] Also, numerical instability, which is a minor problem for the simulation of robots or dancers, becomes a serious impediment when the number of interacting objects rises into the hundreds, thousands, or beyond.

the spheres. Section 6 presents results and running times, and Section 7 discusses the implication of this work and directions of future research.

2 Techniques for Physical Simulation

We categorize physical simulation techniques as acceleration-based, velocity-based, or position-based. Acceleration-based methods come closest to simulating true physics, and they are the most expensive to carry out in computation. Velocity-based methods are farther divorced from "reality" but are faster. Position-based methods are the farthest from reality and the fastest.

Spring model methods (also called *penalty methods*) [25] [26] are typical acceleration-based methods. They allow the objects to overlap. For each pair of overlapping objects, there is a repulsive force proportional to the amount of overlap. The resulting repulsive forces cause the objects to accelerate. Numerical integration converts acceleration to velocity and then to position. These methods require many small time-steps when the acceleration is high. The large number of steps results in a high computational cost. Also, it is often difficult to determine the correct step size. Incorrect discretization of time can cause unusual numerical results such as non-conservation of energy or momentum.

Contact force model methods (also called *analytical methods*) [14] [1] [8] are examples of velocity-based methods. See [4] for a discussion of the many issues involved in contact force models. Rigid bodies are allowed to contact but not overlap. Given the current set of contacts, the method computes a set of consistent velocities such that no two contacting objects penetrate each other. The objects move with these velocities until a new contact occurs. The velocity-based method is much more stable and faster than the acceleration-based method for two reasons: 1) it can exactly compute the time of the next contact, and 2) the resulting time-step tends to be much larger than that needed to accurately carry out numerical integration. Even though these methods do not simulate acceleration, they can handle increasingly sophisticated types of objects and forces: curved surfaces [2], friction [3], and flexible bodies [7]. Unfortunately, the velocity-based method is subject to two problems which cause small time-steps and thus high computational cost. *Local rattle* occurs when one object bounces between two others (such as the rapid bouncing that occurs when you bring a paddle down on a bouncing ping-pong ball). *Global rattle* occurs when there are many interacting objects. Since there are so many, it is inevitable that *some* pair will make contact in a short amount of time. Even systems which can rapidly detect the next collision [16] cannot reduce the number of collisions. Each new contact forces us to recalculate the velocities.

Just as a velocity-based method eliminates accelerations, a position-based method eliminates velocities (and also time, momentum, and kinetic energy). The model only needs to have a potential energy function. Under position-based physics, the objects are allowed to move from their current configuration (positions) to a lower energy configuration along any valid (non-overlapping) energy-diminishing path. Under a linear programming based algorithm for position-based physics, the motion consists of a sequence of steps yielding a piecewise linear path. Each step diminishes the energy as much as possible within some maximal valid convex set of configurations surrounding the current configuration. This convex set depends both on contacts that are occurring and on all contacts that might occur. Therefore the algorithm does not have to stop prematurely to handle a new contact, and there is no local "rattle". Position-based physics also avoids global "rattle" since each object moves a maximal amount. Even if two objects in the model require only a small motion to come into contact, this does not prevent other objects in the model from moving farther, if they are able to.

In his Ph.D. thesis [19] and in joint work with this author [22, 23], Li introduced the concept of position-based modeling. His application is *compaction*: finding tight packings of polygonal objects in the plane. As he and others have noted [29, 30], for most layout applications the motion of the objects is immaterial, and only the final configuration matters. He attempted to carry out compaction using a velocity-based method similar to Baraff's [1], but he found this to be very expensive computationally and also numerically unstable. He formulated a position-based model and algorithm. This algorithm uses Minkowski sums [27, 13] and a *locality heuristic* to calculate a maximum convex region of the configuration space visible to the current configuration. Linear programming finds the lowest energy configuration in this region, and the model jumps to this configuration. According to his experiments, the method typically reaches a local energy minimum in five or fewer jumps even for a layout of more than 100 polygons with up to 75 vertices per polygon. For the examples which were simple enough for him to carry out the velocity-based minimization, the position-based method was at least two orders of magnitude faster.

The algorithms presented in this paper also use position-based physics. For sphere-sphere and sphere-polyhedron interactions, they do not require explicit calculation of the Minkowski sum (unlike Li's method for polygons). For polyhedron-polyhedron interactions, they do require calculation of the Minkowski sum, but unlike Li's *locality heuristic*, they do not require that the polyhedra be decomposed into star-shaped[2] components. The new algorithms can solve for the motions of spheres and polyhedra, whereas Li's algorithm is restricted to two-dimensional polygons.

Interestingly enough, some recent work has moved *away* from the use of large complex optimization systems [5]. Position-based physics reduces motion planning to linear programming. Instead of dealing with implementing optimization code ourselves, we simply use a commercial linear programming library. Goldsmith *et al.* [12] discuss a number of ways optimization is applied in the field of graphics. Most of these involve "either starting or finishing problems." Furthermore, most are non-linear. We believe that position-based physics is unique in the way it uses linear programming to generate a complex motion from start to finish.

Position-based physics yields realistic motion for crowded sets of objects. For free-flying objects, the motion can be less reasonable. However, even in situations for which position-based methods give an unrealistic motion, there may be other applications. Recent work on manipulation of models makes use of non-physical motion [6] [15].

Finally, it is important to note the difference between position-based physics and particle-based systems. Examples of particle-based systems are too numerous to note all of them. See [32] [17] [31] [33] [10] [9] [21] for recent work. In general, particle-based systems model moving particles with forces between them, not rigid colliding objects. Simulating rigid objects using particles requires a very steep repulsive energy gradient, and hence particle-based systems are subject to the same difficulties as other acceleration-based methods: small time-steps and long running times.

3 Position-Based Physics

The philosophy behind position-based physics is to dispense with acceleration, force, velocity, and time. The state of the system is the current configuration (position). The system has a potential energy that depends only on the configuration. There is no notion of kinetic energy.

Section 3.1 defines position-based physics and describes how it can be used for modeling and animation. Section 3.2 gives an algorithm for simulating position-based physics on models with a

[2] A region is *star-shaped* if it contains at least one point which can "see" the entire boundary.

smoothly convex decomposable overlap space: the set of forbidden configurations is a union of convex regions with smooth boundaries. This algorithm uses linear programming. Section 3.3 proves that several types of models have smoothly convex decomposable overlap spaces. Among these are models involving translating spheres inside a fixed polyhedron. We describe what modifications are necessary to handle multiple translating polyhedra.

3.1 Definitions

As per typical usage, *configuration space* denotes the concatenation of the degrees of freedom of the model. For a set of n spheres, the configuration space has $3n$ dimensions. The *free space* is the set of configurations for which no pair of objects overlap.[3] These *free* configurations are also referred to as *valid* or *non-overlapping*. The complement of the free space is the set of *overlapping*, *invalid*, or *forbidden* configurations. We denote the free space by $\mathcal{F}$.

We assume that the model has a potential energy which is a continuous and differentiable real-valued function on the configuration space. For configuration c, the energy is denoted $E(c)$.

A valid motion under position-based physics is an energy-diminishing path in $\mathcal{F}$. Given a starting configuration $c \in \mathcal{F}$, the goal is to generate a continuous and piecewise differentiable curve $\gamma(t)$ such that 1) $\gamma(0) = c$, 2) $\gamma(t) \in \mathcal{F}$ for $t \geq 0$, 3) $E(\gamma(t))$ is monotonically decreasing for $t \geq 0$, 3) $\lim_{t\to\infty} \gamma(t) = c_\infty$ is an *equilibrium configuration*. A configuration c_{equil} is an *equilibrium configuration* if every valid curve $\sigma(t)$ out of c_{equil} ($\sigma(0) = c_{\text{equil}}$ and $\sigma(t) \in \mathcal{F}$ for $t \geq 0$) is non-energy-decreasing: $\frac{\mathrm{d}}{\mathrm{d}t}E(\sigma(t)) \geq 0$.

Note that t is not time and c_∞ may not be the equilibrium state reached by a velocity-based or acceleration-based physics with the same potential energy function. However, a limiting value of $\gamma(t)$ is a valid resting position: a "pile" or "clump". To generate a decent looking animation, we can sample $\gamma(t)$ in a way that makes it appear that the objects in the model are moving with constant or varying velocity, as desired. Also, as shown in Section 4.4, it is possible to add additional constraints on the path to make it appear more natural.

3.2 Linear Programming Based Algorithm

This section gives an algorithm for performing position-based physics on models with a particular type of free/forbidden space. The algorithm is proved to have an equilibrium position as a limit point. If the energy function is linear, then the algorithm can be implemented using linear programming.

3.2.1 Smooth Convex Decompositions

A set S of $\mathbf{R}^n$ is defined to be *convex decomposable* if it is the union of a finite number of convex sets. It is *smoothly convex decomposable* if it is the union of a finite number of convex sets with smooth boundaries. We use a standard definition of smooth: a convex set C is smooth if each point on the boundary has a unique tangent plane and outward unit normal vector and if this plane/normal is a continuous function on the boundary. Actually, it is easy to show that if C is convex and if each point on the boundary of C has a unique outward unit normal, then the normal vector varies continuously on the boundary.

3.2.2 Algorithm

The following is an algorithm for position-based physics when the overlap (forbidden) space $\mathcal{O} = \overline{\mathcal{F}}$ is smoothly convex decomposable. Region $\mathcal{O}$ is the union of m smooth convex regions $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \ldots, \mathcal{O}_m$.

Let $c = c_0$ be the initial configuration. The algorithm sets $\gamma(0) = c$. The algorithm proceeds in a sequence of steps. During the ith step, it constructs the portion of the curve from $\gamma(i-1)$ to $\gamma(i)$. Let us suppose the algorithm has constructed the curve up to configuration $\gamma(i) = c_i$. Here is how it performs the next step.

- For each convex region $\mathcal{O}_j$, it constructs the half-space $H(c_i, \mathcal{O}_j)$ as follows. First it projects c_i onto the nearest point PROJ$(c_i, \mathcal{O}_j)$ on $\mathcal{O}_j$. It computes the tangent plane to $\mathcal{O}_j$ at PROJ$(c_i, \mathcal{O}_j)$. This tangent plane bounds two half-spaces, and $H(c_i, \mathcal{O}_j)$ is the one which does *not* contain the interior of $\mathcal{O}_j$.

- The algorithm computes the convex region
$$I(c_i, \mathcal{F}) = \bigcap_{j=1}^{m} H(c_i, \mathcal{O}_j).$$
Assuming $\mathcal{F}$ is closed (objects are allowed to touch), it follows that $I(c_i, \mathcal{F}) \subseteq \mathcal{F}$.

- The algorithm computes the minimum energy configuration $c_{i+1} \in I(c_i, \mathcal{F})$ which is reachable by an energy-diminishing path from $\gamma(i) = c_i$ to $\gamma(i+1) = c_{i+1}$ within $I(c_i, \mathcal{F})$.

In general, the last step requires some sort of numerical integration. However, if the energy function is linear, it can be accomplished using linear programming. Linear programming can easily compute the point $c_{i+1} \in I(c_i, \mathcal{F})$ which minimizes the energy. The algorithm can set $\gamma(t)$, $i \leq t \leq i+1$ equal to the line segment from c_i to c_{i+1}. Since $I(c_i, \mathcal{F})$ is convex, this segment is valid.

3.2.3 Correctness and Convergence

The following theorem states that the limit points of the algorithm are equilibrium points. This does not mean that the algorithm has a unique limit. If we have a room full of falling objects, some objects might already be on the floor. Any arbitrary motion of an object along the floor does not change the gravitational energy. For odd i, configuration c_i could have a particular object at one end of the room. For even i, it could be at the other end of the room. Thus, as the rest of the objects settled into a "pile", this one object may never come to rest. Nothing in the definition of position-based physics prevents it from bouncing from one wall to the other forever. In this case, the system will have two limit points.

Any particular linear programming library will "break ties" consistently. Therefore, it will not allow an object to bounce from wall to wall forever. It is also possible to introduce a *conservative* energy term: each object is attracted to its current location. Both of these facts can be used to ensure that the algorithm converges to a unique limit. Section 4 illustrates both of these for spheres inside a polyhedron. It is probably possible to modify the definition of position-based physics and the algorithm to disallow "bouncing". However, there may be cases in which we *want* this behavior. Note that "bouncing" is not like rattling. Bouncing does not affect the amount by which the energy is reduced in one step, and therefore it does not increase the number of steps required to reach an energy minimum. For velocity-based methods, frequent collisions diminish the time-step and greatly increase the time required to reach a minimum.

We remind the reader that the definition of an equilibrium point depends only on $\mathcal{F}$ and the energy function $E()$. An equilibrium point for position-based physics is the same as an equilibrium point for any other type of physics. Therefore, the following theorem asserts that the algorithm converges to physically correct "piles" and "clumps". It does not get stuck at a non-equilibrium configuration.

[3] We consider the *exterior* of the container to be a fixed object, and hence the definition of free space includes the container constraint.

However, most models have very many equilibrium configurations. The algorithm does not necessarily converge to the same equilibrium as a "true" simulation of physics.

Theorem 3.1 *For compact (closed and bounded) $\mathcal{F}$, the sequence $c_0, c_1, c_2, c_3, \ldots$ generated by the algorithm has at least one limit point. Each limit point is an equilibrium point.*

Proof: The existence of a limit point is a property of any sequence in a compact set. Let $c_{\lim}$ be a limit point. Suppose $c_{\lim}$ is not an equilibrium point. Therefore, there is a curve $\sigma(t)$ out of $c_{\lim}$ which diminishes the energy. It follows that the energy decreases out of $c_{\lim}$ in the direction $v = \sigma'(0)$ (the tangent vector to $\sigma(t)$ at $t = 0$). For each j, $c_{\lim}$ either lies outside or on the boundary of $\mathcal{O}_j$. If it lies outside, then there exists $\epsilon > 0$ such that $c_{\lim} + tv \notin \mathcal{O}_j$ for $0 \leq t < \epsilon$. If $c_{\lim}$ lies on the boundary of $\mathcal{O}_j$, then v must lie in the tangent plane to $\mathcal{O}_j$ at $c_{\lim}$ or it must point into $H(c_{\lim}, \mathcal{O}_j)$. Hence, there exists $\epsilon > 0$ such that $c_{\lim} + tv \in I(c_{\lim}, \mathcal{F})$ for $0 \leq t < \epsilon$. Also, $E(c_{\lim} + tv)$ has the same derivative at $t = 0$ as $E(\sigma(t))$. The conclusion is that *if* the algorithm reached $c_{\lim}$, the next step would be able to diminish the energy. Let $\delta_{\lim}$ be the amount of decrease.

The algorithm has $c_{\lim}$ as a limit point. The construction of $I(c_i, \mathcal{F})$ uses only continuous functions. For c_i sufficiently close to $c_{\lim}$, the energy decrease $\delta_i = E(c_i) - E(c_{i+1})$ can be arbitrarily close to $\delta_{\lim}$. Since $E()$ is also continuous, there must exist some i such that

$$E(c_i) - E(c_{\lim}) < \frac{\delta_{\lim}}{2} \qquad \text{and} \qquad E(c_i) - E(c_{i+1}) = \delta_i > \frac{\delta_{\lim}}{2}.$$

It follows that $E(c_{i+1}) < E(c_{\lim})$. Since the algorithm always decreases the energy, $c_{\lim}$ cannot be a limit point. This contradicts the assumption that $c_{\lim}$ is not an equilibrium point. ■

3.3 Models with Convex Decompositions

This section examines some models with smoothly convex decomposable overlap spaces. It is shown how to apply the algorithm of Section 3.2.2 to the problems of animating and generating "piles" of spheres and polyhedra under translation.

3.3.1 Minkowski Sum.

The *Minkowski sum* [24, 13, 27, 28] of two point-sets (of $\mathbf{R}^3$ in the case of this paper) is defined

$$A \oplus B = \{a + b \mid a \in A,\, b \in B\}.$$

For a point-set A, let $\overline{A}$ denote the set complement of A and define $-A = \{-a \mid a \in A\}$. For a vector v, define $A + v = \{a + v \mid a \in A\}$. Note that $A + v = A \oplus \{v\}$.

Suppose we have n translating objects $A_1, A_2, A_3, \ldots, A_n$. It can easily be shown that $A_i + v_i$ and $A_j + v_j$ overlap if and only if $v_j - v_i \in A_i \oplus -A_j$.

Lemma 3.2 *If C is convex and S is smoothly convex, then $C \oplus S$ is smoothly convex.*

Proof: It is easy to show that $C \oplus S$ is convex. Suppose p is a point on the boundary of $C \oplus S$ such that p has two distinct unit normal vectors u and u'. Since u is a normal at p, p maximizes the dot product $u \cdot p$ over all points in $C \oplus S$. Since $p = c + s$ where $c \in C$ and $s \in S$ can be chosen independently, s must maximize $u \cdot s$ over all points of S. Yet, $p = c' + s'$ where $c' \in C$ and $s' \in S$ have maximum dot products with u'. Since s has a unique normal, u' is not a normal at s, and $u' \cdot s < u' \cdot s'$. Therefore $u' \cdot (c + s) < u' \cdot (c + s')$. Therefore, $p = c + s$ does not maximize the dot product with u'. This contradicts the assumption that p has two unit normals. Therefore, p has a unique unit normal vector. It is easily shown that if a convex region with unique unit normal vectors has continuous unit normal vectors: it is smooth. ■

Corollary 3.3 *If C is convex decomposable and if S is smoothly convex decomposable, then $C \oplus S$ is smoothly convex decomposable.*

Proof: Let the decompositions be,

$$C = C_1 \cup C_2 \cup \cdots \cup C_l \qquad \text{and} \qquad S = S_1 \cup S_2 \cup \cdots \cup S_m.$$

It can easily be shown that,

$$C \oplus S = \bigcup_{g=1}^{l} \bigcup_{h=1}^{m} C_g \oplus S_h.$$

In other words, the Minkowski sum is the union of the Minkowski sum of each possible pair. By Lemma 3.2, each of these sums is smoothly convex. ■

3.3.2 "Good" Models

The following theorem describes the type of problem to which we can apply the algorithm of Section 3.2.

Theorem 3.4 *If 1) only translation is allowed, 2) all objects are convex decomposable, and 3) at most one object is not smoothly convex decomposable, then the overlap space is smoothly convex decomposable.*

Proof: For each pair of objects, A_i and A_j, at least one is smoothly convex decomposable. Corollary 3.3 implies that the pairwise overlap space

$$\{\langle v_i, v_j\rangle \mid v_j - v_i \in A_i \oplus -A_j\}$$

is smoothly convex decomposable. This transforms to a cylinder

$$\{\langle v_1, v_2, \ldots, v_n\rangle \mid v_j - v_i \in A_i \oplus -A_j\}$$

in the configuration space for the entire model which is decomposable into a union of smooth convex cylinders. ■

3.3.3 Applications

The main application of this paper is a collection of translating spheres in a polyhedral container. Clearly each sphere is smoothly convex decomposable. The remaining object, the complement of the container, is polyhedral and therefore convex decomposable: simply cut it along every plane of every face.

We can not directly apply Theorem 3.4 to the problem of multiple translating polyhedra since a polyhedron is convex decomposable but not smoothly convex decomposable. However, we can "smooth" a polyhedral region P by adding a small spherical region S. By Corollary 3.3, $P \oplus S$ is smoothly convex decomposable. Of course, when we render the motion, we will display the original polyhedra, not the smoothed polyhedra. They will have small gaps between them equal to the diameter of S. For moving objects, this will not be noticeable. Once, the objects form a "pile" or "clump", we can run the algorithm with smaller and smaller S, perhaps halving the radius each time, until the desired accuracy is attained.

Li and Milenkovic's algorithm (for translating polygons in the plane) uses a somewhat different framework than that of the algorithm in Section 3.2.2. In essence, it chooses an arbitrary tangent line/unit normal when the normal is not unique. This means, for instance, that our compaction algorithm for polygons in the plane might get "stuck" at a non-equilibrium configuration. We have never seen it get stuck in practice (but we really have no independent way to verify an equilibrium other than visual inspection). It is likely that one could safely animate multiple translating polyhedra without getting stuck. However, we have not yet run any experiments. Baraff [1] indicates that correctly choosing a set of tangents at each non-unique contact is NP-complete.

4 Simulating an Hourglass using Position-Based Physics

The first section of the accompanying video tape demonstrates the simulation of an "hourglass" or "egg-timer" using position-based physics. The main body of the hourglass is shaded as a curved surface, but it is actually a polyhedron. Each horizontal cross section is a 32-gon, and thus the sides of the hourglass have 160 faces. Initially, 1000 spheres are arranged in a 10 by 10 by 10 grid in the upper part of the hourglass. Position-based physics calculates a (local) gravitational energy minimum for the spheres in the base. As a side-effect, it simulates the flow through the narrow "waist" of the hourglass. The video presents two hourglass simulations. Both are shown at 30 frames per second. Each frame is an actual energy minimization step. Rendering was done with *rendrib*.[4] The first video has 750 frames, and the second has 812 frames.

Sections 4.1 through 4.3 describe how the first video was generated. Section 4.4 shows how extra constraints where added to generate a more realistic motion in the second video. The modified algorithm has acceleration and conservative forces. This video illustrates how the path generated by position-based physics can be controlled to increase the realism.

4.1 Pairwise Constraints

Instead of working in $3n$-dimensional space, we choose a convex subset of the pairwise free spaces. We must do this for each pair of spheres and each sphere with respect to the hourglass polyhedron. Taken together, these constraints are equivalent to $I(c, \mathcal{F})$ of Section 3.2.2.

For a pair of spheres S_i and S_j with radii r_i and r_j and current positions (centers) p_i^{cur} and p_j^{cur}, define

$$u_{ij}^{\mathrm{cur}} = \frac{p_j^{\mathrm{cur}} - p_i^{\mathrm{cur}}}{|p_j^{\mathrm{cur}} - p_i^{\mathrm{cur}}|}$$

to be the unit vector pointing from S_i to S_j. The convex region $\mathcal{R}_{ij}$ is the set of configurations satisfying

$$(p_j - p_i) \cdot u_{ij}^{\mathrm{cur}} \geq r_i + r_j, \quad \text{for } 1 \leq i < j \leq n. \tag{1}$$

The half-space constraint $p_j - p_i \in \mathcal{R}_{ij}$ prevents the spheres from overlapping, and it is exactly equivalent to one of the half-spaces in the first step of the algorithm in Section 3.2.2.

For a sphere S_i and the hourglass polyhedron G, we do the following. Set $h = 1$. Let q_h be the point on the boundary of G closest to p_i^{cur}. Let,

$$v_{hi}^{\mathrm{cur}} = \frac{p_i^{\mathrm{cur}} - q_h}{|p_i^{\mathrm{cur}} - q_h|}$$

be the unit vector from q_h to p_i^{cur}. We create the constraint,

$$(p_i - q_h) \cdot v_{hi}^{\mathrm{cur}} \geq r_i, \tag{2}$$

and we throw away all points p of the boundary of G which do not satisfy

$$(p - q_h) \cdot v_{hi}^{\mathrm{cur}} > 0.$$

(Intersecting a half-space with a polyhedron is easy if the faces are all convex: the intersection of each face with the half-space is also convex.) If some part of the boundary remains, then we increment h and repeat this process. We stop when no point on the boundary of the polyhedron remains. The resulting set of linear constraints on p_i given by all instances of Equation 2 defines a convex region $\mathcal{R}_i$ which is a projection of $I(c_i, \mathcal{F})$ of Section 3.2.2 (actually, it can be a superset, but that can only improve the convergence).

4.2 Minimizing the Potential Energy

For the hourglass example, the potential energy is the sum of the z-coordinates of the spheres. This corresponds to the gravitational energy of a set of spheres with equal mass.

How do we find the next configuration? We need to solve for the configuration that minimizes the gravitational energy under the linear constraints of Equations 1 and 2. This is linear programming. We simply pass this problem to a commercial linear programming package, CPLEX.[5]

Theorem 3.4 implies that the hourglass algorithm cannot "stick" unless the actually physical system would also. Hence, the spheres flow down the hourglass without "clogging" in the middle.

4.3 Box Constraints

We also bound each sphere to lie in a rotated cube centered at the current position of the sphere. The cube has width two times the sphere radius, and it is oriented to have a vertex at minimum z-coordinate. This constraint serves to put an upper bound on the maximum distance a sphere can move in any one step.

This extra constraint serves two purposes. First, it keeps the spheres from falling too fast. In the absence of this constraint, a solitary sphere could fall to rest on the ground from an arbitrary height in a single step. Second, by limiting the motion of the spheres, we limit the pairs of spheres which can collide in the current step. That permits us to reduce the size of the linear program. We do not add a pairwise constraint for two spheres that are too far apart to collide in the next step. We can use bucketing to detect nearby pairs of spheres in nearly linear time.

4.4 Acceleration and Conservative Forces

The reader will notice that the spheres do not accelerate as they fall. We could fix this by detecting if a sphere has fallen the maximum amount, and if so, increasing the height of its bounding box by a fixed amount. To make this work properly, we must use unrotated boxes, unlike the rotated cubes of the previous section.

The reader will also notice that the spheres roll to the back of the hourglass. Using unrotated boxes makes this effect worse: the spheres fall to the lower-left-rear corner of the box, making them fall at an angle. This is an artifact of the simplex method used to solve the linear program. We can add a "conservative" energy term that penalizes each sphere for changing any of its coordinates. In particular, we express each variable x_i as $x_i = x_i^+ - x_i^-$, where both x_i^+ and x_i^- are constrained to be positive (x_i could represent the x, y, or z coordinate of a sphere). To the objective function, we add

$$c_{\mathrm{conserve}} \sum_i (x_i^+ + x_i^-),$$

where c_{conserve} is small compared to the "gravitational constant". In our system, the gravitational constant is 1 and $c_{\mathrm{conserve}} = 0.001$. The second hourglass video illustrates the addition of acceleration and "conservative forces".

[4] The Blue Moon Rendering Tools by Larry I. Gritz.

[5] Version 3.0. CPLEX Optimization Inc. Suite 279. 930 Tahoe Boulevard, Building 802. Incline Village, Nevada 89451-9436.

4.5 Polyhedron-Polyhedron Constraints

The hourglass example does not require polyhedron-polyhedron constraints. For the record, we describe how one could add them to the model.

Li's method for constructing a convex subset of the free space, the *locality heuristic*, requires that the interacting polygons be star-shaped. If they are not, they must be decomposed into star-shaped components. Extra constraints must be added to ensure that the components move as one object.

We describe here a method for selecting a convex free region $\mathcal{R}_{ij}$ for a pair of polyhedra P_i and P_j under translation. As in the case of modeling spheres, we can construct these regions for each pair of polyhedra instead of having to work in $\mathbf{R}^{3n}$ as implied by the algorithm of Section 3.2.2. Note that this method does not require that P_i and P_j be star-shaped, which is an improvement of Li's result. Let P_i represent the "resting" position of P_i, and let $P_i + p_i$ represent P_i translated by p_i from its resting position. As stated in Section 3.3.1 $P_i + p_i$ and $P_j + p_j$ do not overlap if and only if $p_j - p_i$ lies in $F = \overline{P_i \oplus -P_j}$. This is all following theory developed by Li to handle the two-dimensional case.

For a given pairwise configuration $c_{ij}^{\text{cur}} = \langle p_i^{\text{cur}}, p_j^{\text{cur}} \rangle$, we construct convex region $\mathcal{R}_{ij}$ as follows. Set $h = 1$ and let q_h be the point on the boundary of F which is closest to $p_j^{\text{cur}} - p_i^{\text{cur}}$. Define

$$w_{hij}^{\text{cur}} = \frac{p_j^{\text{cur}} - p_i^{\text{cur}} - q_h}{|p_j^{\text{cur}} - p_i^{\text{cur}} - q_h|}$$

to be the unit vector from q_h to to $p_j^{\text{cur}} - p_i^{\text{cur}}$. Add the constraint,

$$(p_j - p_i - q_h) \cdot w_{hij}^{\text{cur}} \geq 0. \tag{3}$$

Throw away all points q of the boundary of F that do not satisfy

$$(q - q_h) \cdot w_{hij}^{\text{cur}} \geq 0.$$

Increment h and repeat until no points of the boundary of F remain. The set of constraints given by all instances of Equation 3 define a convex region $\mathcal{R}_{ij}$ which is a subset of the free space for these two polyhedra. This region is the projection of $I(c_i, \mathcal{F})$ of Section 3.2.2 into the configuration space of these two polyhedra (actually, it can be a superset). For this reason, it is sufficient to use these pairwise constraints.

5 Non-gravitational Potential Functions

For some applications, it may be necessary to simulate a constant force, a "spring" force, or an inverse-square law force between spheres. In this section, we describe how this can be done, and give some examples.

5.1 Attraction between Spheres.

To define an attractive force between spheres, we must first define the distance between spheres in a way that can be represented in a linear program. In the following, $1 \leq i < j \leq n$, where n is the number of spheres. Let S_i and S_j be spheres which are to attract each other.

Create a new variable d_{ij} which represents an approximation to the distance from p_i to p_j. The value of d_{ij} will always be a lower bound on the Euclidean distance. Select a set U of unit vectors. The set U should at least include u_{ij}^{cur} (the unit vector from p_i to p_j) and the six axis-parallel vectors $(\pm 1, 0, 0)$, $(0, \pm 1, 0)$, $(0, 0, \pm 1)$. Apply the following constraints on p_i, p_j, and d_{ij}:

$$u \cdot (p_j - p_i) \leq d_{ij}, \quad \text{for } u \in U. \tag{4}$$

Adding more vectors to U makes d_{ij} a better approximation to the Euclidean distance $|p_j - p_i|$. However, the given U is sufficient for realistic motion, and the presence of u_{ij}^{cur} ensures correct convergence.

For a constant force f_{ij} of attraction between S_i and S_j (independent of distance), we can add the term $f_{ij} d_{ij}$ to the potential function for the model. Often, however, one desires a force which "dies off" with distance, such as the inverse-square law. The corresponding potential function $-f_{ij}/d_{ij}$ is nonlinear. In this case, we use a linear approximation,

$$E_{\text{approx}}(d_{ij}) = f_{ij}\left(-\frac{1}{d_{ij}^{\text{cur}}} + \frac{d_{ij} - d_{ij}^{\text{cur}}}{(d_{ij}^{\text{cur}})^2}\right).$$

For any convex potential function, such as the inverse-square law, the linear approximation is an upper bound on the actual potential energy. The configuration to which the system "jumps" will therefore have *lower* energy than expected, and thus the system will converge even if it uses this approximation.

5.2 Spring Force

It is possible to model forces which *increase* with distance such as a spring force. In this case, the potential function is $E(d_{ij}) = f_{ij} d_{ij}^2$. This type of function is concave (upwards), and thus the method in the previous paragraph does not work. To solve such a model using linear programming, we replace the function by a piecewise linear approximation. First, define l variables $0 \leq d_{ij1}, d_{ij2}, \ldots, d_{ijl} \leq 1$ and add the constraint $d_{ij} = d_{ij1} + d_{ij2} + \cdots + d_{ijl}$. The piecewise linear approximation to the energy function is

$$E_{\text{approx}}(d_{ij}) = f_{ij} \sum_{k=1}^{l} (2k - 1) d_{ijk}.$$

For $k \leq d_{ij} < k + 1$, this energy is minimized when $d_{ij1} = d_{ij2} = \cdots = d_{ijk} = 1$ and $d_{ij(k+1)} = d_{ij} - k$. The value of the approximate function is $f_{ij}(k^2 + (2k+1)(d_{ij} - k))$ which is a good approximation to $f_{ij} d_{ij}^2$.

5.3 Examples.

The video illustrates the application of the constant force, inverse-square law force, and spring force.

In the "trampoline" example, the border of the 30 by 30 grid of spheres is fixed. Each internal grid sphere is attracted to its four immediate neighbors under the constant force. The large sphere falls into this grid and comes to rest. We fake the "bouncing" by playing the sequence forwards and backwards.

5.4 Other Possible Applications.

The examples we have implemented only scratch the surface of what one could do with the current formulation. For example, we could apply these new potential functions to two animations: a pearl falling in "shampoo" and a "lava lamp". The shampoo or "lamp fluid" is a grid-like "gas" of spheres. A constant force attracts sphere S_i to a fixed grid point g_i. The algorithm for modeling attraction of a moving point p_i to a fixed point q_i is straightforward from the math given above.

The pearl is a single sphere in a gravitational potential falling through a "shampoo." The lava lamp fluid uses the same model. We also add a rising "blob" of lava fluid subject to an upwards gravitational field. The beads in the "blob" fluid are subject to a mutually attractive force. For this we choose a potential function which rises linearly to a particular value and then stops increasing. This potential corresponds to a constant, short-range force. This potential function is convex, and thus linear programming can minimize it.

6 Implementation and Results

All examples in the video are a simple mapping of optimization steps to frames. We believe that these demonstrate a variety of realistic looking motions. If necessary, we could modulate the velocity by interpolating between frames. Since consecutive frames correspond to motion from c to c' in the same convex region, every interpolated configuration would also be valid.

6.1 Running Times and Scalability

As usual, our program is always in flux, and it is difficult to generate meaningful timings. The running time depends greatly on the settings of the parameters to the linear programming library. Currently, we use the simplex method. We find that reaching 80-90% of optimum requires only about one-quarter the time as reaching the optimum objective value. Hence, each step can be accomplished in one-quarter the time using only $1/0.8 = 1.25$ times as many steps. This is clearly a good trade-off. At present, we run each linear program for 5000 simplex steps, and this always brings us to within 80% of optimal.

Using these particular methods, we can compute each frame of the second hourglass video in about 1.1 minutes on a DEC Alpha 3000/700 (tm). The entire video can be generated in about 19 hours.

We ran the program with only 500 spheres, and each frame was computed 2.4 times faster than with 1000 spheres. Some of the cost is a naive $\mathrm{O}(n^2)$ geometric algorithm, and the cost of this can be improved. In essence, the cost per step appears to be roughly linear in the number of spheres. Of course, about half as many steps are required to simulate the hourglass. Therefore, the cost appears to be roughly quadratic in the number of spheres. As we discuss in Section 7, this is not necessarily the last work. It is unlikely that distant spheres interact over small numbers of steps. Therefore it should be possible to decompose the problem and/or apply multi-scale methods.

6.2 Number of Iterations

If we are simply generating a "pile" or "clump", we usually do not have an obstruction like the neck of the hourglass through which the spheres must pass more or less sequentially. The following table gives the number of iterations for a k by k by k grid of spheres falling to rest in a box. The number iterations rises surprisingly slowly. This compares very favorably with acceleration or velocity-based methods. For these methods "time until next collision" appears to be inversely proportional to the number of spheres and the number of steps to be at least linear in the number of spheres.

Number of Spheres	27	64	125	216
Number of Iterations	21	25	39	54

6.3 Comparison with Velocity-Based Methods

We attempted to simulate the hourglass with a velocity-based method. As expected, collisions caused the time steps to be very small (around 10^{-5}). It takes about 150 steps to reduce the energy by one unit. For this example, it would require 2 or 3 million steps to reach minimum energy. Each step is faster, but only by a factor of two. We estimate it would take about 1000 times as long to reach an energy minimum than it does using the position-based method of this paper.

7 Conclusion and Future Work

Position-based physics and the linear programming algorithms we use to simulate it are very good ways to rapidly find local energy minima for many interacting objects. They are much faster than other physical simulation techniques, and they are certainly useful for CAD/CAM applications for which only the final configuration matters. The current techniques do not allow rotation in three dimensions (a moot issue for spheres but not for moving polyhedra), but Li [19] has found ways to allow rotation in two dimensions, and it may be possible to generalize this work to three dimensions or devise other methods. The algorithms presented here do not simulate true physical motion: 1) the physics is only semi-Newtonian, and 2) the algorithms use a number of approximations to allow us to apply linear programming. However, in graphics appearance and speed are really all that matters, and these methods rapidly generate motions which appear realistic. Since no other method can currently generate such motions with so little computation, position-based physics and linear programming based simulations warrant consideration as useful tools of computer graphics.

Even for 1000 spheres in a polyhedron, the simulation time is faster than the rendering time. For even larger number of spheres, one would have to break the set of spheres into "zones" and simulate within each zone. By switching between overlapping zones, one could still generate a good animation. Multi-scale techniques might also be applicable [20].

The issue of non-convex sets of valid directions arises for sets of translating polyhedra. However, we believe that this potential difficulty will not have a practical impact. A difficult practical problem is that of explicitly computing configuration spaces (Minkowski sums) for pairs of translating non-convex polyhedra. Dealing with rotations will be even more difficult. However, we believe these difficulties can be overcome. Position-based physics may one day simulate many highly interacting, translating and rotating polyhedra, with links and attractive and repulsive forces.

Another direction of future work is to handle the transition from crowding to freedom. Position-based physics does not do a good job on freely moving objects. Section 4.4 describes how to make the falling spheres appear to accelerate. We use another "trick" to make the ball appear to bounce in the trampoline video. A more general solution would somehow switch between the position-based method and an acceleration/velocity-based method in a way that is low-cost yet realistic.

Acknowledgements: Thanks to Kevin Duffy for implementing many of the algorithms as a summer job. Thanks to Lisa Ventry Milenkovic for proofreading and literature search. Thanks to Karen Daniels for proofreading and suggestions.

References

[1] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics (Proceedings of SIGGRAPH)*, 23(3):223–232, 1989.

[2] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 19–28, August 1990.

[3] David Baraff. Coping with friction for non-penetrating rigid body simulation. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 31–40, July 1991.

[4] David Baraff. Issues in computing contact forces for nonpenetrating rigid bodies. *Algorithmica*, 10(2–4):292–352, 1993.

[5] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24-29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 23–34. ACM SIGGRAPH, ACM Press, July 1994.

[6] David Baraff. Interactive simulation of solid rigid bodies. *IEEE Computer Graphics and Applications*, 15(3):63–75, May 1995.

[7] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. In Edwin E. Catmull, editor, *Computer Graphics (SIGRAPH '92 Proceedings)*, volume 26, pages 303–308, July 1992.

[8] R. Barzel and A. H. Barr. A modeling system based on dynamics constraints. *Computer Graphics (Proceedings of SIGGRAPH)*, 22(4):179–187, 1988.

[9] Jim X. Chen and Niels Da Vitoria Lobo. Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations. *Graphical Models and Image Processing*, 57(2):107–116, March 1995.

[10] N. Chiba, S. Sanakanishi, K. Yokoyama, I. Ootawara, K. Muraoka, and N. Saito. Visual simulation of water currents using a particle-based behavioural model. *Journal of Visualization and Computer Animation*, 6(3):155–172, July 1995.

[11] D. Dobkin, J. Hershberger, D. Kirkpatrik, and S. Suri. Implicitly searching convolutions and computing depth of collision. In *Proceedings of the Second SIGAL*, pages 165–180, 1990.

[12] Jeff Goldsmith and Alan H. Barr. Applying constrained optimization to computer graphics. *SMPTE Journal - Society of Motion Picture and Television Engineers*, 102(10):910–912, October 1993.

[13] L. Guibas, L. Ramshaw, and J. Stolfi. A Kinetic Framework for Computational Geometry. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 100–111. IEEE, 1983.

[14] James K. Hahn. Realistic animation of rigid bodies. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 299–308, August 1988.

[15] Mikako Harada, Andrew Witkin, and David Baraff. Interactive physically-based manipulation of discrete/continuous models. In Robert Cook, editor, *Proceedings of SIGGRAPH '95*, Computer Graphics Proceedings, Annual Conference Series, pages 199–208. ACM SIGGRAPH, ACM Press, August 1995.

[16] Vincent Hayward, Stephane Aubry, Andre Foisy, and Yasmine Ghallab. Efficient collision prediction among many moving objects. *The International Journal of Robotics Research*, 14(2):129–143, April 1995.

[17] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 49–57, August 1990.

[18] A. Kaul, M.A. O'Connor, and V. Srinivasan. Computing Minkowski Sums of Regular Polygons. In Thomas Shermer, editor, *Proceedings of the Third Canadian Conference on Computational Geometry*, pages 74–77, Vancouver, British Columbia, 1991. Simon Frasier University.

[19] Z. Li. *Compaction Algorithms for Non-Convex Polygons and Their Applications*. PhD thesis, Harvard University, Division of Applied Sciences, 1994.

[20] A. Luciani, A. Habibi, and E. Manzotti. A multi scale physical model of granular materials. In *Proceedings of Graphics Interface '95*, pages 136–145, May 1995.

[21] H. Mallinder. The modelling of large waterfalls using string texture. *Journal of Visualization and Computer Animation*, 6(1):3–10, January 1995.

[22] V. Milenkovic, K. Daniels, and Z. Li. Placement and Compaction of Nonconvex Polygons for Clothing Manufacture. In *Proceedings of the Fourth Canadian Conference on Computational Geometry*, pages 236–243, St. Johns, Newfoundland, August 1992. Department of Computer Science, Memorial University of Newfoundland.

[23] V. J. Milenkovic and Z. Li. A Compaction Algorithm for Nonconvex Polygons and Its Application. *European Journal of Operations Research*, 84:539–560, 1995.

[24] H. Minkowski. Volumen und Oberfläche. *Mathematische Annalen*, 57:447–495, 1903.

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

[26] J. C. Platt and A. H. Barr. Constraint methods for flexible models. *Computer Graphics (Proceedings of SIGGRAPH)*, 22(4):279–287, 1988.

[27] J. Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, New York, 1982.

[28] J. Serra, editor. *Image Analysis and Mathematical Morphology*, volume 2: Theoretical Advances. Academic Press, New York, 1988.

[29] Mikio Shinya and Marie-Claire Forgue. Layout out objects with geometric and pysical constraints. *Visual Computer*, 11:188–201, August 1995.

[30] John Snyder. An interactive tool for placing curved surfaces with interpenetration. In Robert Cook, editor, *Proceedings of SIGGRAPH '95*, Computer Graphics Proceedings, Annual Conference Series, pages 209–218. ACM SIGGRAPH, ACM Press, August 1995.

[31] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 185–194, July 1992.

[32] Demitri Terzopoulos, John Platt, and Kurt Fleischer. Heating and melting deformable models (from goop to glop). In *Proceedings of Graphics Interface '89*, pages 219–226, June 1989.

[33] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 269–278. ACM SIGGRAPH, ACM Press, July 1994.