Proceedings of the Seventh Canadian Conference on Computational Geometry,

Laval University, Quebec, August, 1995, pp. 79-84.

Position-Based Physics: Animating and Packing Spheres inside Polyhedra

Victor Milenkovic*
University of Miami
Department of Mathematics and Computer Science

1 Introduction

For a number of years, we have worked on a project on finding tight packings of polygonal objects in two dimensions. Such packings have significant applications in the clothing industry and other industries where shapes are cut from stock material: sheet metal, leather hides, *etc.* We successfully applied computational geometry and mathematical programming to the task of *compaction*: quickly finding tight packings (local energy minima under gravity or other forces). Several researchers in fluid and solid mechanics have asked whether compaction can be generalized to sphere packings. Apparently the centers of spheres packed in a polyhedron might be a good starting point for the vertices of a mesh or tetrahedralization. Others have taken note that the compaction algorithm often generates an entertaining motion of the polygons. It occurred to us that a compaction algorithm might be useful as an animation tool.

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 wanted 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 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 plausible motions for sets of many highly interacting objects. These motions "come to rest" in locally optimal packings of the objects. We present an efficient and numerically stable algorithm for carrying out position-based physics on spheres and nonrotating polyhedra. This work introduces linear programming as a useful tool for graphics animation and packing in three dimensions. 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.

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 an algorithm that simulates position-based physics on set of spheres in 3D and applies it to two animation tasks: beads settling under gravity in a box and a heavy bead settling into lighter beads. Section 4 shows how to handle other potential functions such as attraction among the beads. This technique is applied to the animation of a "pearl" dropping through "shampoo" and a "lava lamp"

*The research of Victor 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. Email: vjm@cs.miami.edu.

[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.

containing a "cloud" of mutually attracting beads rising through a "fluid". Both the "shampoo" and "fluid" are sets of beads held in place by certain potential functions. Section 5 gives the algorithm for interacting spheres with nonrotating polyhedra and applies it to two other tasks: a foot stepping into beads and beads falling in an hourglass.

Section 6 presents results and running times. At present, more implementations are required to determine if the motions "look real" or if more modifications of the algorithms are required. At the appropriate point, we will separate out the graphics parts of this work and present it to the graphics community. At present, it is our hope to make the geometry (and perhaps meshing/triangulation) community aware of the possibility of fast sphere packing.

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 computationally. 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*) [9] [10] are typical acceleration-based methods. They allow the objects to overlap, and 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 nonconservation of energy or momentum.

Contact force model methods (also called *analytical methods*) [1] [2] are examples of velocity-based methods. 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. 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. This 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 simultaneously "jump" from their current configuration (positions) to a lower energy configuration. The motion consists of a sequence of jumps. The physics has two rules: 1) each jump must be maximal in a sense defined in Section 3.1, and 2) the jump configuration must be "visible" from the previous configuration in the configuration space. Rule 1 ensures that the objects rapidly reach a local minimum energy configuration. Rule 2 implies that two objects cannot appear to jump "through" each other and thus ensures a realistic looking animation. The jump calculation takes into account pairs of objects which are near each other. Thus, an object will not bounce between two other objects, and there is no local "rattle". Position-based physics also avoids global "rattle" since each object jumps a maximal amount. Even if two objects in the model require only a small jump to come into contact, this does not prevent other objects in the model from making larger jumps if they are able to.

In his Ph.D. thesis [6] (see also [7, 8]), Li introduced the concept of position-based modeling. His application is *compaction*: finding tight packings of polygonal objects in the plane. For this application, 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. Li and this author formulated a position-based model and algorithm. This algorithm uses Minkowski sums [11, 4] 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 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. Unlike the two dimensional compaction algorithm, they do not require explicit calculation of the Minkowski sum. They also do not use the locality heuristic which requires certain assumptions about the objects. The new algorithms generalize the set of objects from two-dimensional polygons to three-dimensional spheres and polyhedra.

3 Position-Based Physics Applied to Spheres

This section gives a formal definition of position-based physics. It then provides an algorithm for simulating the motion of spheres in an axis-parallel box under the influence of gravity. The spheres can have arbitrary masses and radii. This algorithm can thus simulate a large, massive sphere settling into a bed of lighter, smaller spheres.

3.1 Definition of Position-Based Physics. Under position-based physics, motion consists of a sequence of jumps in configuration space. 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. These *free* configurations are also referred to as *valid* or *nonoverlapping*. Free configuration c' is *visible* from free configuration c if the entire line segment cc' lies inside the free space.

The boundary of the free space consists of flat and/or curved (hyper-)surfaces in the $3n$-dimensional configuration space. A *tangent plane* for the free space is either the extension of a flat surface or a (standard) tangent plane to a curved surface. A region in the free space is *maximal* if it is a) convex and b) bounded by tangent planes.

A region R about a free configuration c is *maximal with respect to* c if 1) R is maximal and 2) no tangent plane bounding R passes through c *unless* c actually lies on the surface to which that plane is tangent. By definition, position-based physics allows a jump from the current configuration c to the minimum energy configuration in some region R which is maximal with respect to c. Condition (1) implies that the jump is as large as possible. Condition (2) assures that the jumping does not stop until the system reaches a local energy minimum.

3.2 Algorithm for Sets of Spheres. This section gives the algorithm for simulating the motion of a set of spheres in an axis-parallel box in a constant gravitational field.

The box is expressed simply,

$$x_{\min} \leq x \leq x_{\max},\ y_{\min} \leq y \leq y_{\max},\ z_{\min} \leq z \leq z_{\max}.$$

For $i = 1, 2, 3 \ldots, n$, let sphere S_i have position $p_i = (x_i, y_i, z_i)$, radius r_i, and mass m_i. A configuration $c = \langle p_1, p_2, \ldots, p_n \rangle$, is a concatenation of sphere positions.

There are two sets of linear constraints on the configuration. The first,

$$x_{\min} + r_i \leq x_i \leq x_{\max} - r_i,\ y_{\min} + r_i \leq y_i \leq y_{\max} - r_i,$$

$$z_{\min} + r_i \leq z_i \leq z_{\max} - r_i, \tag{3.1}$$

for $i = 1, 2, \ldots, n$, keeps the spheres in the box. To define the second set, first let p_i^{cur} and p_j^{cur} be the current positions of S_i and S_j. Define $n_{ij}^{\text{cur}} = (p_j^{\text{cur}} - p_i^{\text{cur}})/|p_j^{\text{cur}} - p_i^{\text{cur}}|$ to be the unit vector pointing from p_i^{cur} to p_j^{cur}. The second set of constraints,

$$(p_j - p_i) \cdot n_{ij}^{\text{cur}} \geq r_i + r_j, \quad \text{for } 1 \leq i < j \leq n, \tag{3.2}$$

prevents the spheres from overlapping.

It can be proven that the set of configurations satisfying the constraints of Equations 3.1 and 3.2 is a maximal region with respect to the current configuration $c^{\text{cur}} = \langle p_1^{\text{cur}}, p_2^{\text{cur}}, \ldots, p_n^{\text{cur}} \rangle$.

Under the influence of gravity, the potential energy function is

$$E(c) = \sum_{i=1}^{n} m_i z_i.$$

All bounds and constraints are *linear*, and they are devised in such a way to allow us to apply linear programming. We use a commercial linear programming package CPLEX[2] to solve for the configuration that minimizes the potential energy within the maximal region. This yields a single "jump", and the algorithm iterates these jumps to yield a motion.

3.3 Possible Applications. The following are two examples of tasks that can be solved using gravitational potential functions: 1) a set of beads settling in a box under the influence of gravity, 2) a heavy bead sinking into a bed of lighter beads.

4 Nongravitational Potential Functions

The simulated beads act much like a real box of beads (as well they should) and do not allow even a heavy object to sink into them very far. To simulate a pearl falling through a bottle of shampoo or a lava lamp, we need non-gravitational potential functions. These

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

forces do not act on the pearl but on the beads through which the pearl is falling. This section describes these functions and how they are used to animate these applications.

4.1 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 n_{ij}^{cur} (the unit vector from p_i^{cur} to p_j^{cur}) 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}:

$$u \cdot (p_j - p_i) \leq d_{ij}, \quad \text{for } u \in U. \tag{4.3}$$

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 n_{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 a 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.

Incidentally, we have not tested the following in practice, but 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 $f_{ij} d_{ij}^2$.

4.2 Applications. It is possible to apply these new potential functions to interesting animations. For example, a pearl falling in "shampoo" and a "lava lamp". The shampoo or "lamp fluid" is a gridlike "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 we use the appropriate technique.

5 Interactions with Polyhedra

Our original compaction algorithm, generalized to three dimensions, could handle interacting (nonrotating) polyhedra. However, we do not have good algorithms for computing the necessary Minkowski sums of polyhedral regions. In this section, we give an algorithm for modeling the interaction of spheres with polyhedra (but not polyhedra with polyhedra) which does not require the Minkowski sum. These can be applied to two interesting animations: a foot stepping into a bed of beads and beads falling in an hourglass.

5.1 Interaction of Spheres and Polyhedra. To improve the running time and to generate more realistic motions, we limit the "jump" of p_i to a cube of width B centered at p_i^{cur}, the current position of sphere S_i, $1 \leq i \leq n$.[3]

Let P be a polyhedron. An *element* of P is a vertex, edge, or face. Point p *projects onto* a face if the perpendicular projection q of p onto the plane containing the face lies in the face itself. The definition

[3] We also limit the motions of the spheres in Sections 3.2 and 4.1, but it was not necessary to clutter the math with this detail.

of p *projecting onto* an edge replaces "plane" by "line." Point p projects onto all vertices.

Let S_i be a sphere of radius r_i and let p_i^{cur} be its current position. An element e of polygon P *potentially interacts with* S_i if a) some point p in the cube of width $2B$ centered at p_i^{cur} projects onto e; b) the cube of width $r_i + 2B$ centered at p_i^{cur} intersects e; c) the distance of p_i^{cur} to its projection q on the point, line, or plane containing the element is at least r_i. If P has fixed position, then replace $2B$ by B in these definitions.

Here is the actual algorithm. For each sphere S_i and each element e of some polyhedron P with which S_i potentially interacts, set up the constraints as if there is a sphere of radius 0 attached to P at position q. As above, q is the projection of the current position p_i^{cur} of S_i onto e (or its extension to a line or plane, if necessary). The algorithms of the previous sections generate the jump.

5.2 Possible Applications. An hourglass is a set of spheres inside a fixed polyhedron. A foot stepping into a bed of beads is a massive polyhedron "falling" under gravity into a box of beads.

6 Implementation and Results

As of the time of this writing, we had not finished implementing non-gravitational potential functions. However, what we have is enough to test the speed of the method and demonstrate the realistic motions it generates for spheres falling under gravity.

6.1 Spheres in a Box. Figure 1 illustrates 27 spheres falling under gravity in a box. We were able to run this example in just a few seconds. We also ran examples with 4^3, 5^3, and 6^3 spheres. These experiments are an "acid test" for the method. All were run on a 50 Mhz Sun workstation.

Spheres	27	64	125	216
Iterations	21	25	39	54
Time (sec.)	4	37	428	3011
Aver. Time	0.19	1.48	10.97	55.76

What is surprising is how slowly the number of iterations rises. This is the number required to reach a local energy minimum. The running time grows roughly cubic in the number of spheres. In any case, it compares very favorably with other simulation methods.

For even larger number of spheres, one would have to break the box up into "zones" and simulate within each zone. By switching between overlapping zones, one could still generate a good animation.

6.2 Spheres and Polyhedra. The other example we show is the interaction of spheres and polyhedra. Actually, we are only doing circles and polygons here, but the algorithm is about the same. Figure 2 shows 10 circles falling in a polygonal hourglass.

7 Conclusion

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, but Li [6] has found ways to allow rotation in two dimensions, and it may be possible to generalize this work 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 of primary importance, and these methods rapidly generate motions which appear realistic. Since no other method can currently generate such motions with so little computation, position-based physic and linear programming based simulations warrant consideration as a useful tool of computer graphics and packing.

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

[3] 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.

[4] 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.

[5] 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.

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

[7] V. Milenkovic, K. Daniels, and Z. Li. Placement and Compaction of Nonconvex Polygons for Clothing Man-

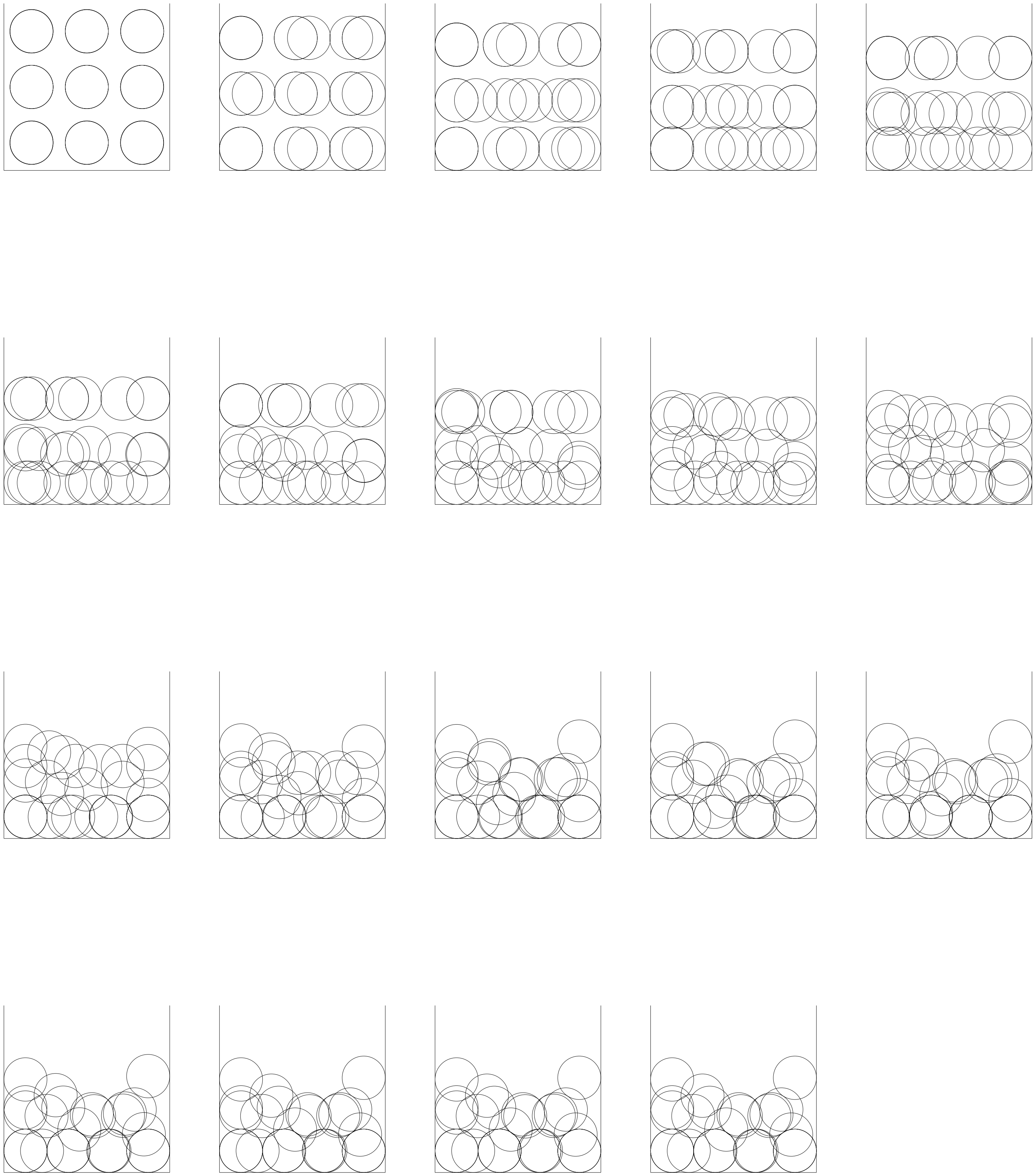

Figure 1: 27 Spheres Falling in a Cubic Container

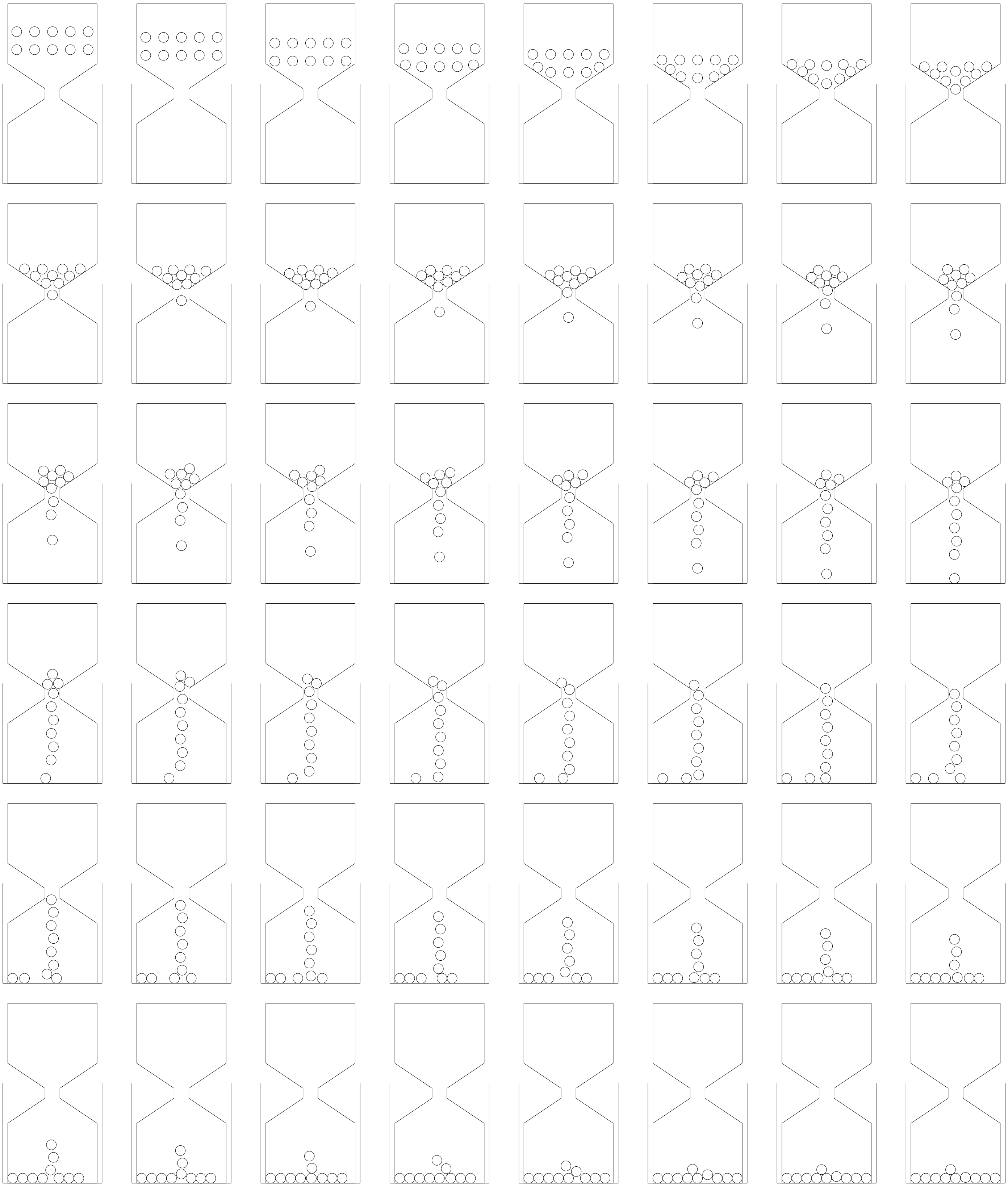

Figure 2: Ten Circles Falling in an Hourglass

ufacture. 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.

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

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

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

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