Simulating Cartoon Style Animation

Stephen Chenney Mark Pingel Rob Iverson Marcin Szymanski

University of Wisconsin at Madison *

Abstract

Traditional hand animation is in many cases superior to simulated motion for conveying information about character and events. Much of this superiority comes from an animator's ability to abstract motion and play to human perceptual effects. However, experienced animators are difficult to come by and the resulting motion is typically not interactive. On the other hand, procedural models for generating motion, such as physical simulation, can create motion on the fly but are poor at stylizing movement. We start to bridge this gap with a technique that creates cartoon style deformations automatically while preserving desirable qualities of the object's appearance and motion. Our method is focused on squash-and-stretch deformations based on the velocity and collision parameters of the object, making it suitable for procedural animation systems. The user has direct control of the object's motion through a set of simple parameters that drive specific features of the motion, such as the degree of squash and stretch. We demonstrate our approach with examples from our prototype system.

Keywords: deformation, squash-and-stretch, physical simulation, stylized animation, stylized rendering

1 Introduction

Animators are expert at conveying information through moving imagery, be it the personality of a character, their actions, or the elements of a story. Procedural animation methods, such as physically-based simulation, also attempt to convey information, yet are typically less effective than hand animation. Users have traditionally faced a choice between the high-quality, high-cost of hand animation and the lower-quality, interactivity of simulated motion.

The superiority of hand animation for communication is not due to deficiencies in the procedural models, rather to two key animation skills:

Abstraction Animators extract the essence of a situation and direct a viewer to it, with exaggeration, timing, anticipation and a host of other techniques [Lasseter 1987]. These techniques serve in part to emphasize the key qualities of the situation, while simultaneously suppressing extraneous details. Procedural methods to date have offered no such flexibility of focus.

* {schenney|pingelm|riverson|marcin}@cs.wisc.edu

Playing to Perception Viewers experience animation through their imperfect human visual systems, which exhibit particular, and sometimes peculiar, behavior with respect to such things as spatial and temporal sampling, focus and distraction. With this in mind, animators deliberately manipulate timing, deformation and other aspects of motion to ensure that the desired perception results. While perceptual issues have been addressed in the rendering community, they have barely been touched on in the simulation community, primarily with motion blur but also in more recent papers [Barzel et al. 1996; O'Sullivan and Dingliana 2001].

Incorporating traditional animation techniques into procedural animation poses an extensive set of problems. Foremost is the problem of characterization: effective animation conveys not just events but also emotions and thoughts. Consider the look of dismay on Wile E. Coyote's face as he sits suspended in air above a canyon, or Luxo Jr's uncertainty as his ball deflates (or is it a her?). Even before attempting to convey such emotions procedurally, we require a way to represent them, which is itself a challenging topic. Another large problem is related to interdependence between rendering style and animation technique – deformations that are appropriate for 2D cell animation are not necessarily the best option for 3D animation.

In the face of these difficulties we concentrate on the problem of adding one animation principle, *squash-and-stretch*, to a simulation of inanimate, colliding objects. Squash-and-stretch deforms objects, even rigid ones, as they interact. Hand-animated squash-and-stretch of a colliding object, as described by Lasseter [1987], addresses two fundamental aspects of animation: the stretch anticipates the collision, and the squash exaggerates its effects. We hypothesize that an additional benefit of squash-and-stretch is that it extends the duration of a collision by replacing a short-lived collision event with a new event that extends over several frames. This ensures that viewers actually see the contact, rather than just inferring its occurrence. Squash-and-stretch may also convey information about the physical properties of objects (their mass, hardness and so on).

In this paper, we present a simulation system that uses a mixture of dynamic and kinematic techniques to squash and stretch objects with geometric deformations. Our goal is to automatically add dynamic, cartoon style deformations to interactive models with the focus on the final appearance of the motion, rather than a physical model. We expect work such as this to apply to interactive entertainment systems, such as computer games, where traditional animation is not easily used and physical models are too slow and unnecessarily complex. It may also be used in animation interfaces for novices or as aids to traditional animation.

An example animation from our system is shown in figure 1. Underlying our system is a simulator that generates ballistic motion for the center of mass of each object. It also includes an collision detection mechanism for affinely deformed bodies. On top of this we control the orientation and deformation of the body using rules that produce motion in a cartoon style. The rules are arrived at from stylistic, as opposed to physical, requirements. For instance, during ballistic motion we always wish to deform the object in the direction of travel, and during a collision we aim for continuity in the deformations. These goals come from stylistic decisions, not

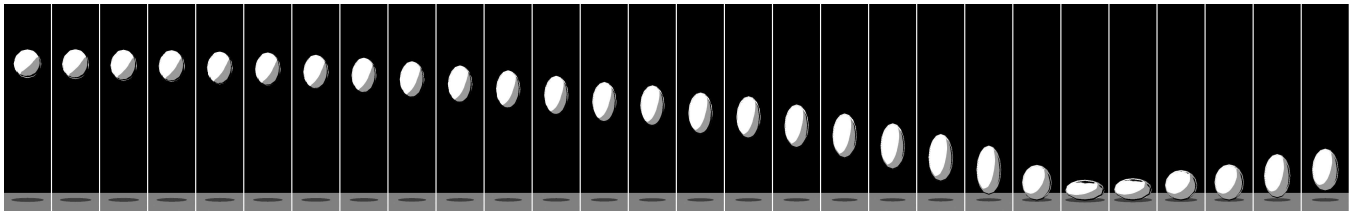

Figure 1: *A time-lapsed animation of a vertically bouncing ball produced by our system. The ball stretches in free space, squashes during collisions, and stretches again as it takes off. Our aim is to emulate the squash-and-stretch technique employed by traditional animators. The parameters (see section 3) for this sequence are:* $r = 0.5$, $s_{max} = 2.0$, $s_{min} = 0.5$, $k_{str} = 0.1$ *and* $k_{sq} = 1.0$.

the requirements of physics.

The greatest strength of our approach is its tight coupling between user controlled parameters and the appearance of the motion. For instance, the user has control over how much an object is stretched, with a single number mapping directly onto the deformation of the object. This makes it easy for a user to attain the desired style.

After a review of previous work, we describe our animation system in section 3, before concluding with a look at future research directions.

2 Related Work

Lasseter's landmark paper [1987] describes the basic principles of cartoon animation and their relationship to computer graphics. One of the techniques described is *squash-and-stretch*, in which an object is stretched as it approaches a collision, squashed through the collision, and then stretched again as it rebounds. Animators working with existing systems typically achieve squash-and-stretch by explicitly key-framing the deformation. To date, three general approaches have been proposed for simulating squash-and-stretch: physically-based models, implicit surface deformations, and time-warp methods.

Physically-based modeling has been used to produce cartoon style deformations, such as those demonstrated by Metaxas and Terzopoulos [1992]. A later system by Faloutos et al. [1997] adds interactivity and some control to the system, and succeeds in generating cartoon style motions for various objects. The system uses a set of free-form deformation modes that define how the object may be deformed and how much energy is involved in the deformation. The object then has masses distributed throughout and a dynamic simulation is run to animate the motion of the object under the influence of internal and external forces. The deformation modes can be designed to allow for squash and stretch, but to create convincing cartoon motion "artificial" forces must be defined to produce the stretch, as it has no physical counterpart. The control in this system is indirect, via parameters such as spring constants and masses, and it seems non-sensical to use a physical model and then deliberately subvert it to produce desirable effects.

Implicit surface based methods have also been used to create stylized motion. Wyvill [1997] describes a method for squash-and-stretch based on local deformations of implicit surfaces. The examples presented are collisions of arbitrary implicit surfaces with a plane. Opalach and Maddock [1994] describe a method for constructing a hierarchy of shape defining elements and associated interaction rules that mimic some traditional animation effects, particularly squash-and-stretch and follow-through. The final appearance of their models appears to be particularly difficult to control, and they address only internal body interactions. Unlike our approach, both implicit surface approaches control only the shape of the object, and ignore other aspects of the motion such as collision response and control of ballistic flight.

Time-warping methods place different parts of an object at different points in time. For instance, the back end of a ball might lag in time behind the front, meaning it travels less in a given real-time instant and hence stretches the object. Platinum Pictures Multimedia Inc. [2000] have introduced a method for squashing and stretching objects using this approach. Campbell et al. [2000] describe a technique that performs time-warping by slicing a 4-dimensional space-time object in order to produce cartoon-style effects. While such time-warping techniques yield interesting results, it is not clear how to modify them to handle collisions between objects in a cartoon-like manner.

Rademacher [1999] alters the geometry of the object based on the direction from which it is viewed in order to capture another aspect of traditional animation. Every object has a base state and several deformed models keyed to specific views. At each frame these key deformations are interpolated to deform the geometry. While Rademacher's technique provides good view-dependent deformations, it does not produce animations in real time, requires extensive work to instrument a new animation sequence, and does not take into account the interactions between objects.

3 Implementing Cartoon Simulation

Our simulation model – the equations that control shape and motion – is driven by the visual style we wish to create. As we describe our system we will indicate those aspects of visual style that we are seeking to capture, such as stretch-and-squash behavior. Our elements of style are based on empirical observations of traditional hand animated behavior, and personal stylistic choices. We do however, provide a range of parameters for adjusting the style of the motion:

Gravity, g: Globally defined and influencing the ballistic motion of objects.

Restitution, r**:** The amount of "energy" lost by an object in a collision (defined on a per-object basis.)

Maximum Stretch, s_{max}**:** The maximum amount that an object can be stretched (defined per-object.)

Minimum Squash, s_{min}**:** The minimum size that an object can be squashed to during a collision (defined per-object.)

Stretch Rate, k_{str}**:** The rate at which an object approaches its maximum stretch as its velocity increases (defined per-object.)

Squash Rate, k_{sq}**:** The rate at which an object moves through a collision (defined per-object.)

All of these parameters map directly onto the appearance, allowing a user to rapidly define an appropriate style. For example, figures 2 and 3 show cylinders with varying s_{max} and s_{min} parameters. In motion, one appears stiff while the other appears soft and pliable.

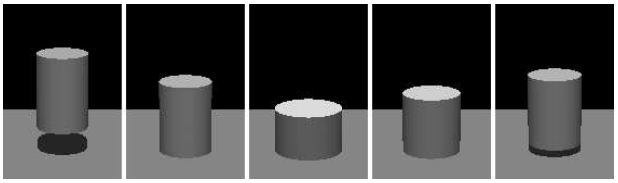

Figure 2: *Five frames from a bouncing cylinder animation showing a cylinder with a relatively small value of 1.5 for s_{max} and a relatively large value of 0.7 for s_{min}. The result is a cylinder that is perceived as rigid in motion and looks more appealing than physically simulated rigid-body motion.*

Both look more appealing than a completely rigid bouncing cylinder.

Our motion model is applicable to moving objects whose only interactions are through collisions. Due to collision detection restrictions, our current implementation only handles convex polygonal models. Our system also assumes that the only collisions will be between a deformable object and fixed, non-deformable objects, although we discuss ways to remove this restriction in the future work section.

Each object in our cartoon physics world operates in one of two modes:

- A free-space mode, in which motion is generated as if objects were point masses moving under the influence of gravity, while shape is driven by velocity.
- A collision mode, in which the motion and shape of objects is driven by the squash-and-stretch behavior.

The simulation is initialized with the positions and velocities for each object. We assume that no objects are inter-penetrating. Each simulation time step then performs the following steps:

1. Update all the objects in free space according to ballistic point mass equations, and set their deformations and alignment according to rules described below. Objects are updated to either the next rendering frame time or the next collision time, whichever occurs first.
2. Compute *collision interpolation parameters* for any new collisions found. Collision interpolations are based on velocities, contact conditions and our desired squash-and-stretch behavior. At each step, they serve as guidelines for the deformation and orientation of the colliding object. Our interpolations are cheap, simple and eliminate the need for complex physics solvers.
3. Update all objects involved in collisions, and set their deformations, orientations and positions.

3.1 Deformations

We use only non-uniform, affine scaling deformations in our system. We chose affine deformations because they are the simplest deformation that can generate stretch and squash effects. Other deformation models could be used, particularly Barr's bending deformations [1984], but different rules would be necessary to drive their parameters.

The stylistic choices we made in defining our deformations are:

- The deformations should be volume preserving.
- Each object has a natural set of deformation axes, including one principle axis, and scaling should always be done with respect to these axes. These axes define a deformation coordinate system with the x-axis aligned with the "forward" direction for the object.

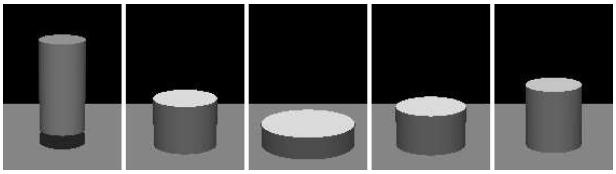

Figure 3: *Five frames from a bouncing cylinder animation showing a cylinder with a relatively large value of 2.0 for s_{max} and a relatively small value of 0.2 for s_{min}. This cylinder appear to be made of a softer material than that in figure 2.*

We find these choices generate pleasing motions for a range of objects, but other options clearly exist.

The deformation is controlled by a single parameter, s, which is the scaling coefficient along the principle axis. We scale the other dimensions equally according to our volume preserving requirement, resulting in the following scaling matrix which is applied in the deformation coordinate system:

$$\begin{bmatrix} s & 0 & 0 \\ 0 & \sqrt{1/s} & 0 \\ 0 & 0 & \sqrt{1/s} \end{bmatrix} \quad (1)$$

The parameter s is controlled by the simulator, as we will now describe.

3.2 Motion and Deformation in Free Space

The motion of objects in free-space is derived from the following stylistic choices:

- Objects should move with roughly ballistic trajectories, with the user retaining control of gravity.
- Objects should stretch according to their velocity: the stretch should be applied in the direction of travel and by an amount that increases with higher velocity.

The second stylistic choice, combined with our deformation approach, has a significant stylistic implication: objects must always stay aligned with their direction of travel, and so cannot rotate under the normal rules of rigid-body motion. We could relax this requirement by either stretching in directions not aligned with the direction of travel, or applying stretches without regard to the internal symmetries of the object. Each alternative would result in a different look for the animation. Finally, acceleration appears to be a desirable way to drive stretch. However, for ballistic motion the acceleration is constant, whereas stretch should not be constant for a bouncing ball.

To implement these design decisions, we update an object's shape and position by translating it according to Newtonian equations for a point mass, rotating it to align its principle deformation axis with its direction of travel, and deforming it according to our deformation model. The deformation parameter, s, is set according to the following equation:

$$s = \frac{k_{str}\|\mathbf{v}\|s_{max} + 1}{k_{str}\|\mathbf{v}\| + 1}$$

where s_{max} and k_{str} are user defined parameters (section 3), and $\mathbf{v}$ is the object's velocity vector. This equation gives $s = 1$ (no stretch) for zero velocity, and in the limit approaches $s = s_{max}$ (user defined maximum stretch) for infinite velocity. Both the maximum and the rate at which the stretch approaches the maximum is controlled by the user. Figure 1 shows a time-lapse sequence of a

ball bouncing vertically, illustrating the growth in stretch with increasing velocity for a ball dropped from a height of 10m under the influence of earth gravity with $s_{max} = 2$ and $k_{str} = 0.1$.

Our requirement that the object stay aligned with its direction of travel would introduce an instantaneous flip in the object's orientation when it reaches the top of a vertical bounce. We avoid this by detecting any sharp reversals in the object's velocity and flipping its "forward" direction.

3.3 Collision Detection

Our system must detect collisions between moving, deformed objects. We describe a solution to this problem for interactions between convex polygonal models. This convexity constraint arises only from our collision detection technique.

In order to accurately detect collisions for our convex polygonal models, we use a modified version of VClip [Mirtich 1998]. VClip reports the closest features of two objects by tracking those features over time. We modify VClip to perform all computations in the global coordinate space and do lazy transformation of Voronoi regions and other features. The modified VClip method is fast, robust and accurate, in keeping with our goal of interactive frame rates.

We need to isolate the exact time of the collision in order to keep with our goal of non-penetration during collisions. Therefore, when two objects are determined to be colliding, we perform a binary search on the time parameter to find the exact time of the collision, as described by Moore and Wilhelms [1988]. At the collision time, the collision interpolation parameters are computed (see section 3.4) and the objects cease moving ballistically.

3.4 Interpolation for Collision Deformations

Our primary goal in designing a collision deformation scheme is to create a motion that is smooth and looks "good", but not necessarily realistic. For example, if a bouncing ball does not deform as it hits the ground, or if the ball is only touching the ground for an instant, it is perceived as jarring. This perception remains even though many types of balls (pool balls for example) would in real life exhibit such jarring motion. We summarize our requirements with the following rules:

- The object should squash during the collision by an amount that depends on how hard it hits and the user defined squash parameters.
- The deformation should vary smoothly through the collision, and should be continuous through the transition between ballistic and colliding motion.
- The object should appear to "stick" through the collision, rather than slide.

The object must also rotate during the collision, to align its deformation axis with the outgoing direction of travel. We also switch the forward direction of the object, so a vertically bouncing object does not flip as it collides, but rather appears to roll.

We must control the position, orientation and deformation of an object through a collision. We do this with interpolation schemes that drive the deformation and rotation of the object. These, combined with the non-sliding constraint, also imply the translation of the object. The parameters for the interpolations are computed at the start of a collision. At that time the system has available the current velocity, $\mathbf{v}_{in}$, the current deformation factor, s_{in}, and the collision normal, $\mathbf{n}$.

A collision alters the velocity of the object by reflecting it about the collision normal and multiplying its normal component by the user defined restitution coefficient: $\mathbf{v}_{out} = \mathbf{v}_{in\perp} - r\mathbf{v}_{in\parallel}$. During

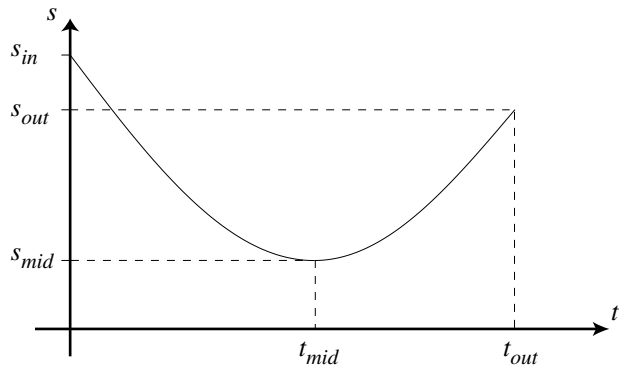

Figure 4: *The interpolation function used to control deformation during a collision consists of two sinusoidal pieces: one controlling the squash as the object compresses and the other controlling its outgoing stretch.*

the course of the collision the velocity is ignored. All the translation of the object happens as a result of the interpolations.

We choose to use sinusoidal interpolation functions to control the deformation. This choice was motivated by a desire to appear spring-like, although we do not use mass or spring constants to derive the deformation. Figure 4 illustrates the interpolation function and marks some key points. At the start of the collision, $s = s_{in}$ and the local time parameter is $t = 0$. At the point of maximal squash for this collision, $s = s_{mid}$ and $t = t_{mid}$. When the collision completes, $s = s_{out}$ and $t = t_{out}$.

The maximum squash for this collision, s_{mid}, is computed based on the ratio of s_{in} to the maximum stretch, s_{max} with the formula

$$s_{mid} = 1 - (1 - s_{min}) \frac{s_{in}}{s_{max}}$$

This will give the user defined maximum possible squash when the incoming stretch is at its maximum, and less squash with decreasing incoming stretch.

For the squash phase of the collision, we use an interpolation function of the form

$$s = s_{in} - (s_{in} - s_{mid}) \sin \omega_{in} t$$

The parameter ω is chosen to achieve a smooth transition from ballistic motion to collision squash. Consider the point on the object furthest from the collision point, which we assume to be at a distance l, computed as the maximum extent of the object in the principle deformation direction. As it collides, the top point is moving with speed approximately $\|v_{in}\|$ (ignoring the motion of the point due to the deformation changing.) As the collision takes over, the point will be moving with speed $-l\frac{ds}{dt}$. Equating these speeds places a constraint on the derivative of the interpolation function, from which we can derive

$$\omega_{in} = \frac{\|v_{in}\|}{l\,(s_{in} - s_{mid})}$$

To complete the incoming squash computations, we calculate $t_{mid} = \frac{\pi}{2\omega_{in}}$. We find that the continuity constraint is sometimes stronger than necessary, and users would rather directly control the collision timing. To this end we provide a user controlled parameter, k_{sq} which changes the computation of ω and hence t_{mid}:

$$\omega_{in} = \frac{k_{sq}\|v_{in}\|}{l\,(s_{in} - s_{mid})}$$

Higher values for k_{in} result in faster, sharper looking collisions, suggesting a light-weight colliding object. Smaller values give

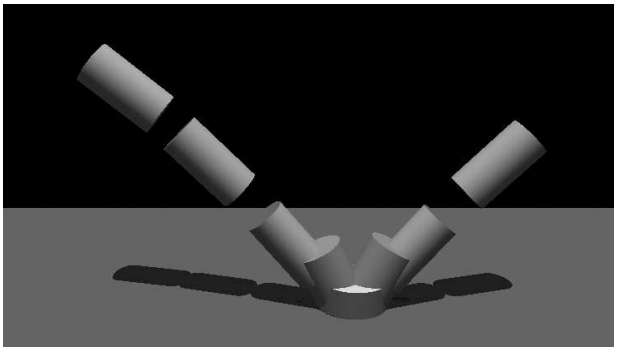

Figure 5: *A few frames, overlaid, of a collision in which a cylinder strikes a plane at an angle. The cylinder is stretched at the initial contact, and orientated along its direction of travel. As the collision proceeds, it simultaneously squashes and rotates about the contact point, before stretching out again and taking off. This sequence also illustrates the alignment of the cylinder as it follows its ballistic trajectory. The frames in this composite were not sampled at uniform time intervals. The ballistic motion frames are less densely sampled.*

longer collision times making an object appear heavier, or in extreme cases giving the sense that time is slowed during the collision (which also conveys a sense of mass.)

The outgoing stretch parameters are computed in a similar manner, using

$$s = s_{out} - (s_{out} - s_{mid}) \cos \omega_{out} t$$

During the course of a collision, the object is rotated to move from its initial alignment with the incoming velocity to its final alignment with the outgoing velocity. We use linear interpolation for rotation, broken into two stages such that the object rotates through half the required angle while squashing, and the other half while stretching.

Having set the collision parameters at the impact occurs, at each subsequent simulation time-step the interpolation scheme is evaluated to set the deformation for the object. The object is then rotated about the contact point according to the interpolated rotation. This combination of motions will generally result in the colliding objects inter-penetrating or losing contact. We resolve this by moving the object in the collision normal direction to re-establish contact without penetration. As a result of these manipulations the object appears to rotate about its contact point while simultaneously squashing and stretch. Figure 5 depicts a few snapshots of a cylinder colliding at an angle.

Up to this point we have made two implicit assumptions. The first is that the object does not come to rest as a result of its collision. We would like to manage this case as objects with $r < 1$ will always come to rest. Secondly, our collision interpolation schemes are computed at the start of the collision. We have not addressed our handling of cases in which an object is involved in a second collision before completing the first. The two collision case is very common for objects bouncing in enclosed spaces – there must be corners where the object can hit two bounding surfaces at the same time. In the following two sections we discuss these cases.

3.5 Coming to Rest

Objects that collide with low normal velocity are brought to rest by the simulator. In most cases, this simply sets the outgoing velocity to be zero, and the outgoing orientation to be aligned with the collision normal. A flag is also set to indicate that the object should no longer be considered moving. The interpolation schemes described above are then computed as usual and reasonable behavior results.

Under some circumstances an object can collide with a surface while moving slowly away from it. This happens at low velocities as the object is rotated and stretched according to its ballistic motion rules. In such situations we stabilize the object with retrograde rotation such that it appears to fall back along its path, rather than flip over.

3.6 Simultaneous Collisions

Simultaneous collisions frequently occur when an object collides in a corner. It is difficult to come up with a consistent style rule for defining the object's behavior in such cases. For instance, if a ball comes into a 90° corner at a 45° angle, it should probably squeeze in and be reflected back along its path (figure 6). But what if the collision is glancing, or the object does not hit both faces simultaneously? Traditional hand animators have the option of avoiding such cases, but as the designers of an interactive simulator we must handle any cases that arise. We use a rule that works well with our approach, but is not as visually pleasing as we would like.

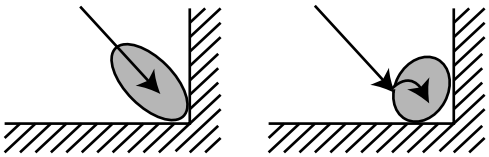

Figure 6: *Two examples of a corner collision, when the ball contacts multiple surfaces during its deformation. In the left case, the ball should probably be reflected backward, while in the situation on the right it is less clear what a traditional animator would do.*

Our approach serializes collisions. When the simulator detects a collision involving an object already involved in a primary collision, it adds the second contact surface to a queue of pending contacts. As the object squashes and rotates, if it penetrates the second contact surface it is pushed back in a direction tangential to the primary collision surface, thus ensuring no inter-penetration occurs with either contact. When the primary collision completes, the pending contact takes over and new interpolation parameters are computed. This produces reasonable motion, as shown in figure 7. Under some circumstances the pending collision may become unnecessary as the object moves away from the second face, in which case it is subsequently ignored.

3.7 Applications

To test our motion model, we implemented it in the form of a simulation library that can be used by applications to produce cartoon style motion. One application is a demonstration environment, used to produce the figures in this paper. We have also implemented a simple game with the library. Similar to *Breakout*, users control a paddle to guide a bouncing ball around a play-field. The aim is to collide with and eliminate blocks. This illustrates the use of user controlled objects, a non-trivial environment and the ability to add and delete objects on the fly. Most importantly, it demonstrates the robustness of our motion model in the face of practical problems.

4 Conclusion

We have presented a mixed dynamic and kinematic model for simulating cartoon style squash-and-stretch motions in real time. The greatest advantage of our approach is its clear user controls that map directly onto properties of the motion, allowing the easy specification of particular styles. For instance, we can readily define

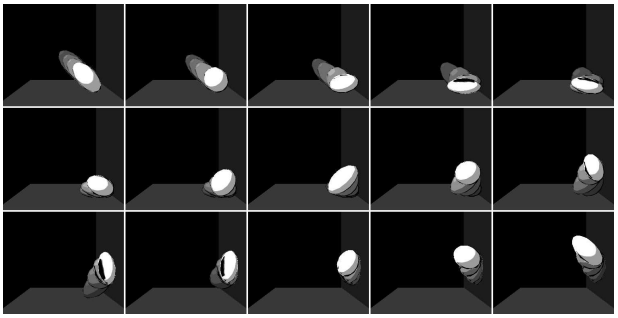

Figure 7: *A sequence showing a corner collision, running left to right and top to bottom. The ball initially hits the floor, and begins its collision interpolation. After it hits the right wall, in the top center frame, the ball is pushed back away from the wall while it completes its initial collision. The initial collision with the floor completes in the center frame, at which point it begins processing the second collision with the wall. The ghost images are intended to represent the motion of the ball, and are not present in the animated sequence.*

parameters that make an object appear light and rigid, or soft and heavy. This enhances the expressive power of procedural simulations.

There are many desirable extensions to our system. Foremost, we would like to enable multiple moving, deforming objects. It is relatively clear how to perform squash-and-stretch on two moving objects. All the key ingredients for our interpolation scheme are present in such a case, including a collision normal and a natural model for the change in velocity. The difficulty comes in the secondary effects of multiple moving objects, such as handling simultaneous collisions between two moving objects and a static object. Other extensions include special handling of small-angle bounces and incorporation of concave objects. Producing reasonable results in a more generalized setting requires more flexibility than our interpolation schemes currently allow. A different approach may be required to handle systems of many moving cartoon style objects.

A broader issue is the incorporation of other traditional animation principles into simulated motion. To date, almost all researchers have concentrated on aspects of shape, and ignored the broader question of global motion. For instance, anticipation and follow-through require changes to trajectories and timing, something that our system does not address. We see stylized motion, rather than shape, as the most important area for future work.

References

BARR, A. H. 1984. Global and local deformations of solid primitives. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, ACM SIGGRAPH, 21–30.

BARZEL, R., HUGHES, J. F., AND WOOD, D. N. 1996. Plausible motion simulation for computer graphics animation. In *Computer Animation and Simulation '96*, Eurographics, 184–197. Proceedings of the Eurographics Workshop in Poitiers, France, August 31-September 1, 1996.

CAMPBELL, N., DALTON, C., AND MULLER, H. 2000. 4d swathing to automatically inject character into animations. In *SIGGRAPH 2000 Conference Abstracts and Applications*, ACM SIGGRAPH, 174. Technical sketch.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 1997. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics 3*, 3 (July - September), 201–214.

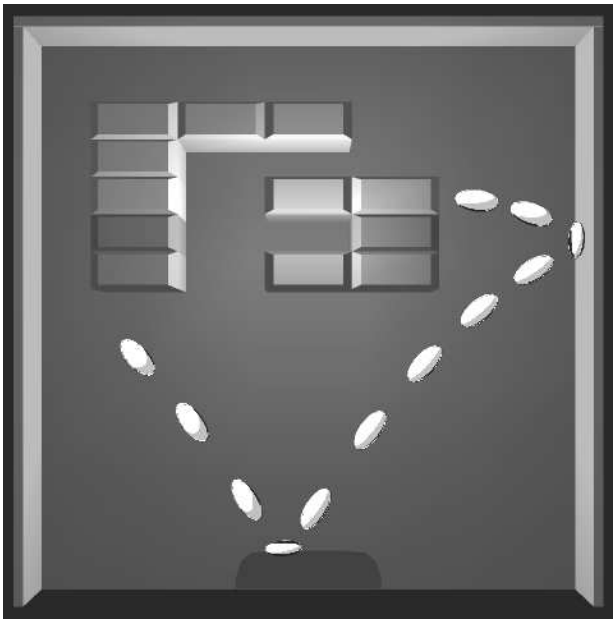

Figure 8: *A strobe sequence from a simple game implemented with our cartoon simulator. Similar to* Breakout*, the user controls a paddle to guide the ball that eliminates blocks upon contact.*

GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *Computer Graphics, Proceedings of SIGGRAPH 98*, ACM SIGGRAPH, 447–452.

LASSETER, J. 1987. Principles of traditional animation applied to 3D computer animation. In *Computer Graphics: SIGGRAPH '87 Conference Proceedings*, ACM SIGGRAPH, 35–44.

METAXAS, D., AND TERZOPOULOS, D. 1992. Dynamic deformation of solid primitives with constraints. In *Computer Graphics: Proceedings of SIGGRAPH 92*, ACM SIGGRAPH, 309–312.

MIRTICH, B. 1998. V-clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics 17*, 3, 177–208.

MOORE, M., AND WILHELMS, J. 1988. Collision detection and response for computer animation. In *Computer Graphics*, vol. 22(4), ACM SIGGRAPH, 289–298.

OPALACH, A., AND MADDOCK, S. 1994. Disney effects using implicit surfaces. In *Proceedings of the Fifth Eurographics Workshop on Animation and Simulation*, Eurographics.

O'SULLIVAN, C., AND DINGLIANA, J. 2001. Collisions and perceptions. *ACM Transactions on Graphics 20*, 3 (July), 151–168.

PLATINUM PICTURES MULTIMEDIA INC., 2000. Motion pack plug-in. Computer program.

RADEMACHER, P. 1999. View-dependent geometry. In *Computer Graphics: Proceedings of SIGGRAPH 99*, ACM SIGGRAPH, 439–446.

WYVILL, B. 1997. Animation and special effects. In *Introduction to Implicit Surfaces*, J. Bloomenthal, Ed. Morgan Kaufmann, ch. 8, 242–269.