A Method for Cartoon-Style Rendering of Liquid Animations

Ashley M. Eden
UC Berkeley
eden@cs.berkeley.edu

Adam W. Bargteil
CMU
adamwb@cs.cmu.edu

Tolga G. Goktekin
UC Berkeley
goktekin@cs.berkeley.edu

Sarah Beth Eisinger
UC Berkeley
seisinger@gmail.com

James F. O'Brien
UC Berkeley
job@cs.berkeley.edu

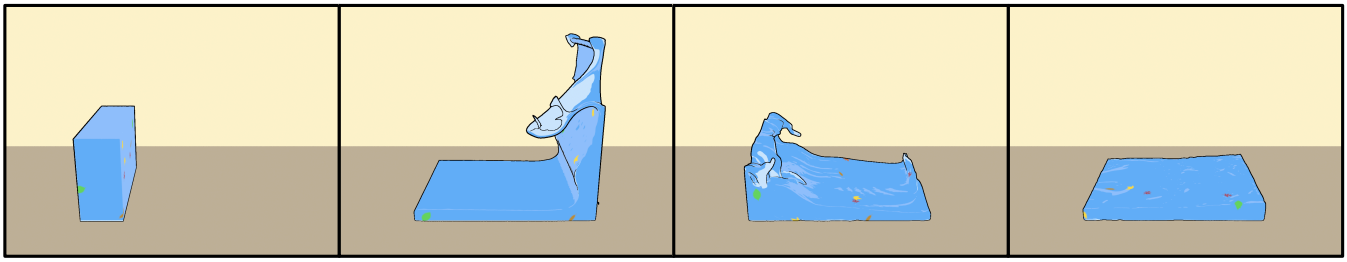

Figure 1: This sequence of water sloshing back and forth in an invisible tank demonstrates our cartoon rendering style for liquid animations. The sequence begins when the fluid is released from a cuboid shape, sloshes against the opposite wall, back against the first wall, and eventually settles on the ground.

Abstract

In this paper we present a visually compelling and informative cartoon rendering style for liquid animations. Our style is inspired by animations such as *Futurama*,[1] *The Little Mermaid,*[2] and *Bambi*[2]. We take as input a liquid surface obtained from a three-dimensional physically based liquid simulation system and output animations that evoke a cartoon style and convey liquid movement. Our method is based on four cues that emphasize properties of the liquid's shape and motion. We use bold outlines to emphasize depth discontinuities, patches of constant color to highlight near-silhouettes and areas of thinness, and, optionally place temporally coherent oriented textures on the liquid surface to help convey motion.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: Non-photorealistic rendering, image processing, natural phenomena, surface tracking, semi-Lagrangian contouring.

1 Introduction

Although liquids have long been portrayed in traditional artwork and animation, there are no techniques designed specifically for the cartoon-style rendering of three-dimensional liquid animations. However, non-photorealistic rendering techniques do exist for the animation of other fluids. For example, Selle et al. [32] present a method for cartoon rendering of smoke animations. Because these methods are for rendering particle-based fluid flow, they are not well-suited to free-surface liquid animations.

Liquid motion is often extremely difficult to generate by hand. Methods do exist, however, for simulating three-dimensional liquid motion, and have been quite successful in applications of photorealistic computer graphics. We leverage these simulation techniques for non-photorealistic rendering. In this paper, we present a method for cartoon-style rendering that demonstrates how liquid simulations can be used in non-photorealistic situations. In particular, we show that it is possible to use the output of a liquid simulator to drive a compelling cartoon-style liquid animation. The resulting animation can be used as is, or, if a more stylized motion is desired, it can serve as a quick, easy to generate starting point that an animator can modify. Our rendering style is inspired by observation of some classic hand-animations such as *Futurama,*[1] *The Little Mermaid,*[2] and *Bambi*[2]. We demonstrate our method on a variety of motion generated by a three-dimensional physically based liquid simulation system. An example of this can be seen in Figure 1.

Traditional cartoon animators [23, 6] use abstractions, or simplifications, to convey the important aspects of a scene with a restricted set of visual cues. In the context of animating liquids, cartoon animators convey motion and geometry in a simplified manner, for example using very few colors and omitting some surface details. Our rendering style is based on four cues that emphasize properties of the liquid surface's shape and motion, while evoking a cartoon style. In particular, we use bold outlines to stress *depth discontinuities*, and patches of constant color to highlight *near-silhouettes* and *areas of thinness*. We also optionally apply temporally coherent *oriented textures* to help convey motion and/or stylize the animation.

As we will discuss, variations of some of the techniques used in our rendering method have been used before in other applications. The main contribution of our paper is in effectively combining a set of techniques to create cartoon-style simulation-based *liquid* animations. The animations created with our method look like cartoon-style liquids, even without changing the underlying simulated motion. In addition, our method is fast, easily tuned, and portable to a variety of mainstream liquid simulation systems and renderers.

2 Related Work

Several researchers have used fluid simulation techniques for non-photorealistic rendering. Curtis et al. [10] use a two-dimensional fluid simulation to drive a watercolorization system. The effects are convincing, but their goal of producing synthetic watercolor images is orthogonal to our goal of creating cartoon-style liquid animations. Witting [38] describes a two-dimensional fluid simulation system for creating smoke and liquid effects for traditionally-animated films. They are largely concerned, however, with integrating a fluid simulation system into their production pipeline, and present stylistic effects different than ours. Selle et al. [32] successfully create cartoon smoke renderings from fluid simulation infor-

Graphics Interface Conference 2007
28–30 May, Montreal, Canada

[1] ©Twentieth Century Fox Film Corporation
[2] ©Disney

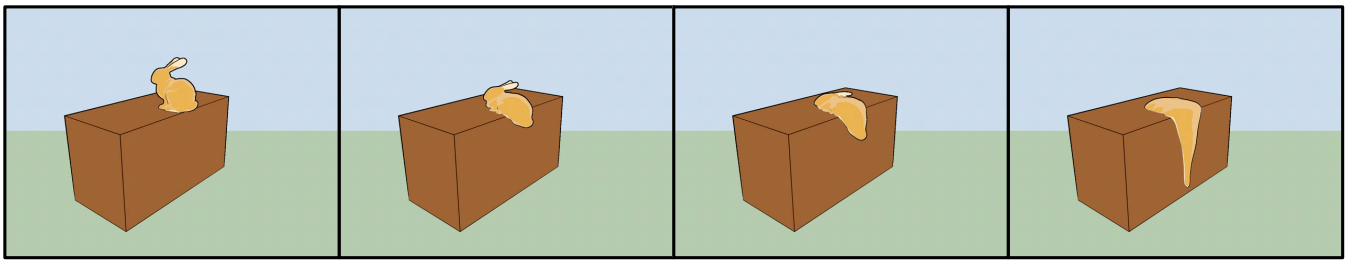

Figure 2: This figure shows a highly viscoelastic liquid bunny 'melting' over the side of a block.

mation. They use a particle-based fluid simulator to determine the placement of view-facing rendering primitives. By basing the visual traits of the primitives on simulation information such as density, they are able to produce quite compelling and informative animations. McGuire and Fein [29] use similar methods in their real-time extension. While the work of Selle et al. [32] is quite similar to ours in spirit, our underlying problems have a fundamental difference: liquids have a free surface that is the interface between the liquid and the surrounding air, while smoke does not. An effective cartoon rendering style for liquid animations must deal explicitly with this free surface. Bargteil et al. [4] and Kwatra et al. [25] synthesize temporally coherent textures on an animated liquid's free surface. These methods could provide an alternative way of achieving non-photorealistic effects, though the results would be different from the cartoon-style described in this paper.

While a wide variety of non-photorealistic rendering styles exist [20, 34], we base our rendering style on that typical of some traditional cartoon animations. In particular, we use a simple set of colors and bold lines. DeCarlo et al. [12], for example, previously adopted a similar style in an effective and appealing abstraction of two-dimensional images. To achieve our cartoon style, we apply several techniques to three-dimensional liquid simulation data. Some of our techniques are similar to those used in previous NPR methods. Gooch et al. [19] demonstrate the importance of edge lines in the creation of understandable technical illustrations. We place lines at edges in the depth map, using a technique similar to that introduced by Deussen and Strothotte [14] for rendering trees, and used by Selle et al. [32] for rendering smoke. Many of our cues are view-dependent. DeCarlo et al. [11] introduce suggestive contours, and show the importance of highlighting view-dependent information. Chi and Lee [8] and Barla et al. [5] each define and check for near-silhouette regions. Researchers have developed a variety of toon shading techniques and software. Lake et al. [26] present a real-time method for cartoon shading, pencil sketching, and silhouette edge detection and rendering. Anjyo and Hiramitsu [1] present a cartoon shader with stylized highlights. In the context of toon rendering objects with properties common to liquids, Diepstraten and Ertl [15] present a method for cartoon shading transmissive and reflective surfaces.

There are also many techniques for the scientific visualization of flow information in two and three dimensions [7, 37]. Visualization methods seek to enhance certain features of the flow, for example flow lines, vortices, and velocity gradients, often for purposes of scientific analysis. Joshi and Rheingans [24] use features typical to comic book illustrations in order to convey motion and change over time in a single image. Their methods, such as the use of strobe silhouettes and speed lines, evoke a cartoon style while conveying important flow information. Although their methods are intended for single frame images, it would be an interesting area of future work to integrate their methods into cartoon-style liquid animations. This could similarly be said for the illustrative flow techniques of Svakhine et al. [35].

3 Methods

Before rendering, we first generate the liquid motion using a physically based liquid simulator. The liquid simulation data provides us with surfaces that we will later render in a cartoon-style. Our methods should work seamlessly with any fluid simulation system that can generate some representation of the surface and surface normals. This representation could be a polygon mesh (our chosen representation), a level-set, or a particle set treated as metaballs. Applying oriented textures means tracking points across timesteps, which also requires some notion of how the surface changes. Not all simulators provide this information, but many could be extended to do so. We use the staggered-grid data structure of Foster and Metaxis [17], the semi-Lagrangian advection method introduced by Stam [33], the extrapolation boundary condition of Enright et al. [16], the viscoelasticity model of Goktekin et al. [18] and the semi-Lagrangian contouring surface-tracking method of Bargteil et al. [3].

Once we have the surfaces, we render them in a cartoon style. By casting rays or using depth buffer information, one may check for points on the surface that are near-silhouettes and/or at places where the liquid is thin from the current viewpoint. In our implementation, we use Pixie [2], a RenderMan-compliant renderer, and check for the previously mentioned cues using two custom shaders. While we use a ray tracer, any renderer that can return two intersections per ray is sufficient. If we want to include oriented textures in the animation, prior to rendering we track oriented points on the surface through time, and map the textures onto the surface using standard texture mapping techniques [27].

To evoke a cartoon-style, we use large regions of constant color in the final animation. Aside from the bold black outlines and the oriented textures, we only use three colors for the liquid in the animation: one color for points at thin regions of the liquid surface, one color for points at near-silhouettes (that aren't also at thin regions), and one color for everything else. In most of our animations, these colors are medium blue, light blue, and dark blue, respectively. In our implementation, the renderer assigns colors based on the mapped textures (if we are including oriented textures in the final animation), and on the responses of the two shaders. In addition to a color image, the renderer also produces a depth map, which we then use to find depth discontinuities. We get a final frame once the outlines at depth discontinuities are composited with the color image. At no point is any lighting information required.

3.1 Near-Silhouettes

We seek to highlight areas of the surface near the silhouette. This cue is view-dependent and user-controllable. We define a near-silhouette as a point where $n \cdot v \leq t$, where n is the unit normal of the surface, v is the view vector, and t is a user-input threshold (see Figure 4). In order to achieve less detail farther from the camera and to avoid aliasing problems, we do not normalize v. The shapes of the near-silhouette regions give cues as to the geometry of the liquid, without the need for lighting information. The motion of the near-silhouettes also helps convey the motion of the surface – for example the motion of the ripples in Figures 1 and 3.

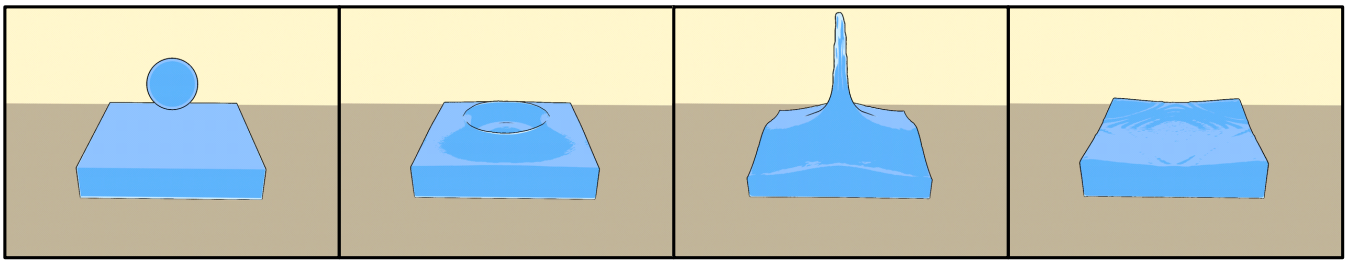

Figure 3: This figure shows a ball of liquid being thrown into a shallow liquid pool.

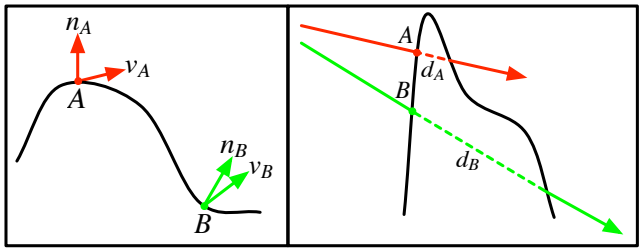

Figure 4: Our near-silhouette and thinness criteria. Left: Near-silhouette criteria. Based on the angle between the view and normal vectors, point A would be classified as a near-silhouette while point B would not. Right: Thinness criteria. Based on the distance a ray from the camera travels through the liquid, point A is classified as thin, while point B is classified as thick.

3.2 Thinness

We also highlight areas where the liquid is thin – another view-dependent, user-controllable cue. For our purposes, thinness refers to the distance a ray from the camera travels between entering and exiting the liquid after its first intersection. If the distance is less than a user-input threshold, we consider the liquid to be thin (see Figure 4). Highlighting thin regions of the liquid can give the impression of foam crests and transparent regions of otherwise murky liquids.

3.3 Outlines

Depth discontinuities provide extremely important cues that help humans understand shape. Consequently, in our rendering method, we highlight these discontinuities with bold outlines. This is accomplished by performing edge detection on the depth map generated during the rendering pass, similar to Deussen and Strothotte [14].

Before doing edge detection, we first filter the depth map: During rendering, multiple depth samples are taken for each pixel in the depth map and a Catmull-Rom filter is applied to these samples. The depth map is then convolved with a Gaussian filter. Both filtering steps help anti-alias the edges, produce thicker outlines at larger depth discontinuities, and avoid depth map resolution artifacts.

After filtering the depth map we apply our edge detection method. For each pixel we compute the maximum difference d between the inverse of its depth value and those of its four neighbors. We pass these differences through a non-linear sigmoid function,

$$\frac{1}{1+exp(\frac{d-t}{\kappa})} \tag{1}$$

with user-specified threshold t and steepness κ. This gives a grayscale image of the depth discontinuities. Using the inverse of the absolute depth helps achieve less detail farther away.

Since the flow is coherent, the depth discontinuities generally change smoothly and do not flicker. However, noise in the depth map on the magnitude of our threshold can result in some pixels erroneously being tagged as edges. These errors can cause flickering. To remove these pixels, we use a filter that looks at all the pixels a user-specified distance (in the max-norm) away from a given tagged pixel. If none of these pixels are also tagged as edge pixels, the current pixel is not along an edge and is untagged.

Highlighting depth discontinuities is an image-space way of highlighting silhouettes. Any pre-existing technique for detecting and highlighting silhouettes can be used [21, 30, 31, 9, 13, 28]. For our purposes, we found that using a depth discontinuity-based method offers several advantages. First, it's user-controllable: Users can adjust how much detail they want to highlight. It's also fast: The depth map can be computed, nearly for free, as part of the rendering process. Additionally, tuning parameters is near interactive since the depth map need not be recomputed. Since the method works in image space, it is also independent of the resolution of the mesh, and would work on other surface representations.

3.4 Tracking

To help convey motion, many cartoon animations add shapes that appear to be attached to the surface. These textures are particularly useful if there is motion but little change in geometry, for example the laminar flow of a calm river. We optionally place oriented textures on the surface of the liquid by tracking oriented points. For each texture to move faithfully with the surface, we must track not only position on the surface, but also orientation in the plane tangent to the surface.

We therefore track *oriented points*, each represented by a planar set of *un-oriented* points. By requiring that each point set moves rigidly, we can ensure that we are always able to compute orientation in the plane containing the points. We discuss how to track a single un-oriented point set, or equivalently one oriented point. Multiple oriented points may be tracked by tracking multiple point sets. Since we are using a semi-Lagrangian surface tracking method, it is more natural to track surface points back in time.

Our method for rigidly tracking a planar set of points from timestep i to $i-1$ comprises five stages (see Figure 5). We begin with a set of points, S_i, at timestep i which we wish to track back to timestep $i-1$ (see Figure 5a). Each point in S_i is projected to the nearest point on the surface, resulting in set P (see Figure 5b). Distortion is minimal because the points in S_i lie in the tangent plane to the surface. The semi-Lagrangian contouring surface-tracking method of Bargteil et al. [3] provides a mapping from points on the surface at timestep i to points on the surface at timestep $i-1$. Using this mapping, we find the image of each of the points in P on the surface at timestep $i-1$, yielding set Q (see Figure 5c). Because we are interested in the rigid transformation of the set S_i as a whole, we use Horn's absolute orientation algorithm [22] to find the best rigid transformation T from set S_i to set Q. Applying T to S_i yields the planar set R (see Figure 5d). To ensure that the centroid of the point set lies on the surface, and that we are able to compute an orientation in the plane tangent to the surface, we translate and rotate R such that its centroid is tangent to the surface, resulting in set S_{i-1} (see Figure 5e). Note that Figure 5 is a two-dimensional ver-

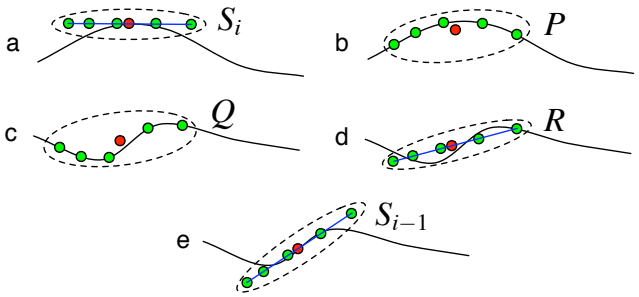

Figure 5: An two-dimensional overview of the five stages in our method for tracking a set of points from timestep i to timestep $i-1$. The green dots represent points in the set, and the red dots indicate centroids. The point set, S_i, is initially tangent to the surface at its centroid (a). We project each point to the surface (b). For each point in P, we find the corresponding point on the surface at the previous timestep (c). We find the best rigid-body transformation from S_i to Q (d). We rigidly transform so that the centroid lies on and is tangent to the surface (e).

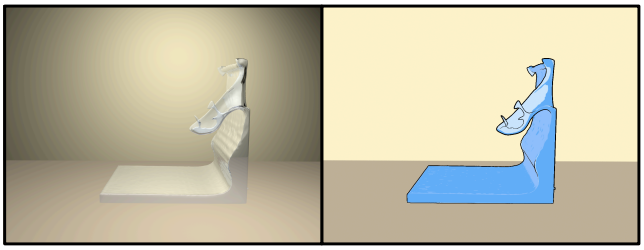

Figure 6: This figure compares a clear rendering of sloshing water to our cartoon rendering style.

sion of a three-dimensional process. For the rotation step depicted in Figure 5e, we are only rotating the plane so that its normal lines up with the surface normal; We do not rotate in-plane.

4 Results

We have tested our cartoon-style rendering method on a variety of examples of liquid behavior, such as the sloshing water example in Figure 1. Here, near-silhouettes are especially effective at portraying the motion of surface ripples. We attached oriented textures of leaves to the liquid surface in this example to help convey the motion of the surface. A user can easily substitute any desired texture or set of textures.

Figure 2 shows frames from an animation of a viscoelastic liquid bunny. Due to the elastic forces the bunny retains its shape throughout most of the animation. Our rendering style highlights both the bunny shape and the liquid motion. Here, highlighting areas of thinness, most notable on the bunny's ears, and the near-silhouette along the back, are especially effective. Figure 3 shows frames from a ball of liquid being thrown into a shallow liquid pool. The near-silhouettes convey much of the important shape information without making the frames appear too detailed.

Figure 6 shows a comparison of a clear photorealistic rendering style with our cartoon rendering style. Both are able to convey information about the surface geometry, but in very different ways. Additionally, even though the same simulated motion drove both animations, many viewers commented that the cartoon rendering made the liquid motion appear more cartoonish. Some viewers, however, said they expected the cartoon rendering to have more cartoon-like motion. Generating cartoon motion would be an interesting area of future work.

Figure 7 shows a frame from an animation of a fountain with two different rendering styles. For the right image we extended the

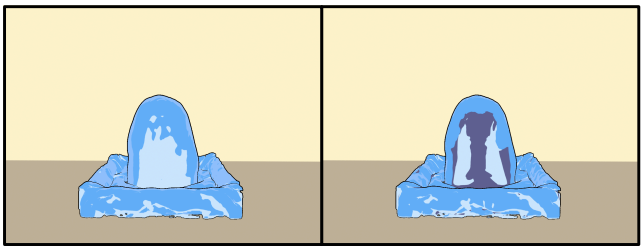

Figure 7: This figure shows frames from two versions of a spurting fountain animation. Although the liquid does not separate as it falls, one can see that parts of the liquid do get very thin. Left: Uses the depth-discontinuity, near-silhouette, and region of thinness methods. Right: We've enhanced the animation with a technique that highlights the surface behind thin regions.

thinness technique (Section 3.2) to show geometry behind thin regions of the liquid. Wherever the liquid was thin, we checked the distance the view vector traveled from exiting the liquid after its first intersection, to entering the liquid again as its second intersection. If this distance was within a user-defined range, we colored that point purple. Finally, Figure 8 shows an animation of a block of viscoelastic fluid being dropped on the Stanford Bunny.

Our method is fast and easily generalized to a variety of standard simulators and renderers. Our simulations take approximately two to four minutes per timestep, with four to five timesteps per frame. Tracking takes less than one minute per timestep, even when tracking several hundred points, and is a pre-processing step; in order to change which specific textures are used, one just has to remap the textures. Rendering each frame, including the depth map, takes three minutes on average. Each frame was rendered at 1280x960 and subsampled to 640x480 for additional anti-aliasing. Finding, filtering, and compositing the bold outlines takes about three seconds total per frame. These timings were taken without hardware accelerations or optimizations.

5 Conclusion

We have presented a method for rendering compelling cartoon-style liquid animations, based on four cues that emphasize properties of a liquid surface's shape and motion. The animations are driven by the output of a liquid simulator, thus extending the possible uses of liquid simulation systems to include non-photorealistic applications. Our method is quick, easy to use, and simple to implement. The user-specified parameters are intuitive and can be tuned at a rate that is near-interactive.

We note that our method doesn't produce animations which are indistinguishable from hand-drawn cartoons. Rather, they are *inspired by* hand-drawn cartoon animations. One point of note is that we use realistic motion with unrealistic rendering. We find that the cartoon rendering style alters the viewer's perception of the liquid so that it appears more cartoonish. However, if an even more cartoonish feel is needed, creating cartoon motion or altering the simulator to achieve it would be an interesting area of future work. Thorton [36], for example, demonstrates a directable particle simulation system for creating stylized water splash effects in three-dimensions.

6 Acknowledgments

We thank the other members of the Berkeley Graphics and Vision Groups for their helpful criticism and comments, particularly Alex Berg, Bryan Feldman, Xiaoyang Mao, Maneesh Agrawala, and Jitendra Malik. This work was supported in part by California MICRO 04-066 and 05-044, and by generous support from Apple Computer, Pixar Animation Studios, Autodesk, Intel Corporation, Sony Computer Entertainment America, and the Alfred P. Sloan

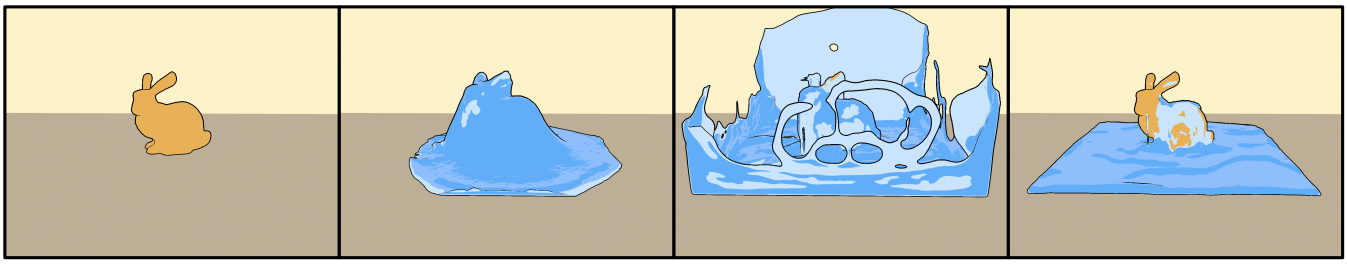

Figure 8: This figure shows a block of viscoelastic liquid falling on the Stanford Bunny.

Foundation. Ashley Eden was supported in part by an NSF Graduate Research Fellowship.

REFERENCES

[1] Ken-ichi Anjyo and Katsuaki Hiramitsu. Stylized highlights for cartoon rendering and animation. *IEEE Comput. Graph. Appl.*, 23(4):54–61, 2003.

[2] Okan Arikan. Pixie: Photorealistic renderer, 2005.

[3] Adam W. Bargteil, Tolga G. Goktekin, James F. O'Brien, and John A. Strain. A semi-Lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25(1), 2006.

[4] Adam W. Bargteil, Funshing Sin, Jonathan E. Michaels, Tolga G. Goktekin, and James F. O'Brien. A texture synthesis method for liquid animations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Sept 2006.

[5] Pascal Barla, Joëlle Thollot, and Lee Markosian. X-toon: An extended toon shader. In *NPAR*, 2006.

[6] Prestor Blair. *Cartoon Animation*. Walter Foster, 1994.

[7] Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH 1993*, pages 263–270, New York, NY, USA, 1993. ACM Press.

[8] Ming-Te Chi. Stylized and abstract painterly rendering system using a multiscale segmented sphere hierarchy. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):61–72, 2006. Member-Tong-Yee Lee.

[9] Cassidy Curtis. Loose and sketchy animation. In *SIGGRAPH 1998: Conference Abstracts and Applications*, 1999.

[10] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-generated watercolor. In *Proceedings of ACM SIGGRAPH 1997*, pages 421–430. ACM Press/Addison-Wesley Publishing Co., 1997.

[11] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics*, 22(3):848–855, July 2003.

[12] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. In *Proceedings of ACM SIGGRAPH 2002*, pages 769–776. ACM Press, 2002.

[13] Philippe Decaudin. Cartoon looking rendering of 3D scenes. Research Report 2919, INRIA, June 1996.

[14] Oliver Deussen and Thomas Strothotte. Computer-generated pen-and-ink illustration of trees. In *Proceedings of ACM SIGGRAPH 2000*, pages 13–18, 2000.

[15] J. Diepstraten and T. Ertl. Interactive Rendering of Reflective and Transmissive Surfaces in 3D Toon Shading. In *Procceedings of GI Workshop Methoden und Werkzeuge zukuenftiger Computerspiele '04*, 2004.

[16] Douglas P. Enright, Stephen R. Marschner, and Ronald P. Fedkiw. Animation and rendering of complex water surfaces. In *the Proceedings of ACM SIGGRAPH 2002*, pages 736–744, July 2002.

[17] Nick Foster and Demitri Metaxas. Realistic animation of liquids. In *Graphics Interface 1996*, pages 204–212, May 1996.

[18] Tolga G. Goktekin, Adam W. Bargteil, and James F. O'Brien. A method for animating viscoelastic fluids. In *Proceedings of ACM SIGGRAPH 2004*, pages 463–468. ACM Press, August 2004.

[19] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of ACM SIGGRAPH 1998*, pages 447–452. ACM Press, 1998.

[20] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering*. AK Peters, Ltd., 2001.

[21] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive technical illustration. In *Proc. 1999 ACM Symposium on Interactive 3D Graphics*, April 1999.

[22] B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society A*, 4(4):629–642, April 1987.

[23] Ollie Johnston and Frank Thomas. *The Illusion of Life*. Abbeville Press, 1981.

[24] Alark Joshi and Penny Rheingans. Illustration-inspired techniques for visualizing time-varying data. In *IEEE Visualization*, page 86, 2005.

[25] Vivek Kwatra, David Adalsteinsson, Theodore Kim, Nipun Kwatra, Mark Carlson, and Ming Lin. Texturing fluids. *To appear in IEEE Transactions on Visualization and Computer Graphics*.

[26] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of NPAR 2000*, pages 13–20, New York, NY, USA, 2000. ACM Press.

[27] Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. In *Proceedings of ACM SIGGRAPH 1993*, pages 27–34. ACM Press, 1993.

[28] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-time non-photorealistic rendering. *Computer Graphics*, 31(Annual Conference Series):415–420, 1997.

[29] Morgan McGuire and Andi Fein. Real-time cartoon rendering of smoke and clouds. In *Proceedings of NPAR 2006*, June 2006.

[30] Ramesh Raskar and Michael Cohen. Image precision silhouette edges. In *Proc. 1999 ACM Symposium on Interactive 3D Graphics*, April 1999.

[31] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of ACM SIGGRAPH 1990*, pages 197–206. ACM Press, 1990.

[32] Andrew Selle, Alex Mohr, and Stephen Chenney. Cartoon rendering of smoke animations. In *Proceedings of NPAR 2004*, pages 57–60, New York, NY, USA, 2004. ACM Press.

[33] J. Stam. Stable fluids. In *the Proceedings of ACM SIGGRAPH 1999*, pages 121–128, August 1999.

[34] Thomas Strothotte and Stefan Schlechtweg. *Non-photorealistic computer graphics : modeling, rendering and animation*. Morgan Kaufmann, 2002.

[35] Nikolai A. Svakhine, Yun Jang, David S. Ebert, and Kelly P. Gaither. Illustration and photography inspired visualization of flows and volumes. In *IEEE Visualization 2005*, 2005.

[36] John David Thornton. Directable simulation of stylized water splash effects in 3d space. In *SIGGRAPH '06: Material presented at the ACM SIGGRAPH 2006 conference*, page 94, 2006.

[37] Jarke J. van Wijk. Rendering surface-particles. In *Proceedings of VIS 1992*, pages 54–61, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[38] Patrick Witting. Computational fluid dynamics in a traditional animation environment. In *Proceedings of ACM SIGGRAPH 1999*, pages 129–136. ACM Press/Addison-Wesley Publishing Co., 1999.