Real-Time Rendering of Cartoon Smoke and Clouds

Morgan McGuire*
Brown University

Andi Fein
Microsoft Corporation

Abstract

We describe an algorithm for rendering animated smoke particle systems in a cartoon style. This style includes outlines and cel-shading. For efficient self-shadowing effects, we introduce nailboard shadow volumes that create complex shadows from few polygons. Including shadows, the renderer draws only three polygons per particle and entirely avoids the latency of depth buffer readback.

We combine the renderer with a fast simulator that generates the phenomenology of real smoke but has artistically controllable parameters. Together they produces real-time interactive smoke animations at over 30 fps. Thus our algorithm is well-suited to applications where interactive performance and expressive power are preferred to realism; for example, video games and rapid development of animation.

CR Categories: I.3.0 [Computer Graphics]: General

Keywords: smoke, cartoon, non-photorealistic rendering, shadows

1 Introduction

Cartoon styles include crisp, quantized 'cel' shading and black outlines. Typically the shading is a self-shadowing effect. This style uses minimalist, high-contrast visual cues to convey 3D shape information. Automated rendering in cartoon styles is interesting for the creation of animated films, and more significantly, animated television shows that need to quickly produce and revise many hours of animation. Cartoon styles also are increasingly popular in video games, where interaction makes it impossible to pre-draw the images by hand.

Many previous techniques exist for fast, automated outlines and two-tone shading (e.g., [Markosian et al. 1997; Raskar and Cohen 1999; Gooch et al. 1998, 1999; Buchanan and Sousa 2000; Gooch and Gooch 2001; McGuire and Hughes 2004]). Most of these require a mesh and do not extend well to amorphous shapes represented by particle systems or grids.

Shadowing and self-shadowing are important effects for conveying the volume and position of amorphous objects that otherwise lack a reference. The shadow volume algorithm is well-suited to cartoons because it provides sharp shadow boundaries and self-shadowing. However, like cartoon rendering, shadow volumes are typically applied to meshes that have stable topology and adjacency information.

To create animated cartoon sequences for entertainment applications we combine a compressible fluid particle simulator and Newtonian rigid body simulator with our renderer.

* morgan@cs.brown.edu

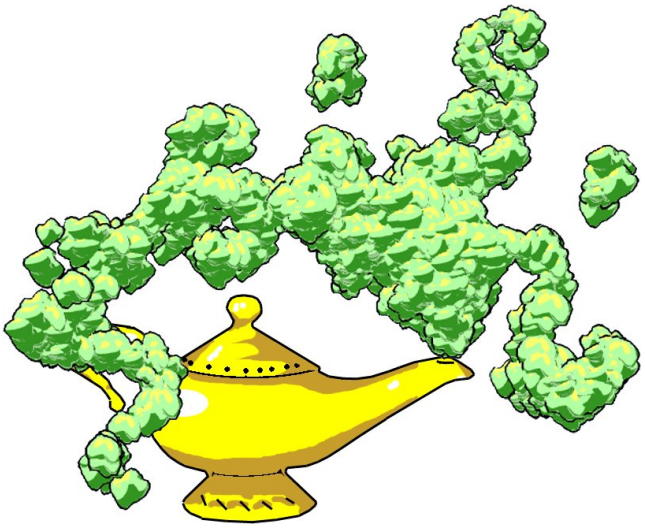

Figure 1. A magic oil lamp with real-time 3D cartoon smoke.

2 Related Work

Billboard-like objects called *graftals* have been used for many years in NPR to render blobby objects with complex boundaries; see Gooch and Gooch [2001] for details. However, Selle et al. [2004] created the first system for *simulating* and rendering cartoon animations of amorphous objects, allowing complicated animated scenes. They trace the movement of smoke particles through a grid space that is simulated via an offline realistic method by Fedkiw et al. [2001]. Selle et al. render the body of each particle with a large, unshaded billboard textured with a single smoke puff, and then draw 1-pixel outlines by reading back the depth buffer and applying depth differences. This method achieves about 1 fps for rendering, probably because reading the depth buffer is slow, but in practice is also limited by the simulation performance. The bandwidth cost of reading the depth buffer and writing to every pixel during outline rendering is high, but likely to become less significant on future hardware. However, the long-term performance cost is tied to the latency, which will increase because it is proportional to the length of the ever-growing hardware pipeline.

The core of our system builds on the particle and billboard framework of Selle et al. In implementing this, we extend the renderer with shading, shadows, and *nailboards*—billboards with depth maps [Schaufler 1997]. We use nailboards to create curved intersections between smoke and geometry, and to render outlines without reading from the current depth buffer. This is both fast (we render thousands of particles at 30 fps) and also extends to multi-pixel thick outlines.

Shadow volumes are a well-known real-time shadowing method; see Akenine-Möller and Haines' book [2002] for detailed review. The traditional algorithm computes the image space intersection of the scene and the geometric volume shadowed by an object. One of its drawbacks is that complex shadows require complex geometry. We apply the nailboards to the shadow volumes to create additional detail without geometry. Without

Figure 2. Clouds deflected by collisions with an airplane and swirling from interaction with propeller blades and vortices.

this, billboards would intersect their own shadows at straight lines and reveal the underlying simple rectangular geometry.

Following a seminal paper by Foster and Metaxes [1997], most realistic smoke simulation uses a grid-based approach. Grids are not yet fast enough for real-time interactive simulation, so we return to the classic particle-system model introduced by Reeves [1983]. Specifically, we use an extended version Sim's [1990] artistically controllable method inspired by recent work by Selle, Rasmussen, and Fedkiw [2005]. The following section briefly overviews the simulator. For more implementation details, see the technical report by McGuire [2006].

3 Simulation

Individual smoke molecules have little mass or volume, so smoke simulation is actually fluid simulation of the surrounding medium (air), with smoke tracers to make currents visible. We simulate a set of spherical particles that each represent many smoke molecules.

The smoke is approximated as a compressible fluid acted upon by six environment forces: *vortex*, *buoyancy*, *gravity*, *wind*, *turbulence,* and *drag*. An artist directly adjusts these parameters or rigs models to produce them during simulation to achieve a desired effect. For example, linking a vortex generator to a propeller causes a plane to shed vortices in its slipstream during flight, as demonstrated in Figure 2.

The simulator that we use [McGuire 2006] also models interaction of smoke with rigid bodies. Particle motion is constrained by collisions with the scene, however particles do not constrain each other because the fluid is compressible (and particle-particle collisions are too complex to solve in real time).

Particles should dissipate over time, reflecting the diffusion of the molecules they represent. We simulate diffusion as a linear reduction of density over time (making diffusion proportional to velocity is more realistic but is aesthetically less pleasing). As described in the next section, this causes particles to visibly shrink over time so that they have zero size when they are removed from the simulation and do not appear to pop.

The simulator applies first-order forward Euler integration to the forces and enforces constraints using an iterative solver, both of which are implemented using the Open Source ODE (http://ode.org) physics library maintained by Russell Smith.

4 Rendering

Cartoon artists have individual styles. The renderer implements four common effects: quantized shading, shadowing, black outlines, and gradient fills. These can be selectively enabled to approximate different styles.

4.1 Cel-shading

As in Selle et al. [2004], we render the body of each particle with a textured billboard parallel to the image plane, where the size of the billboard is proportional to the density of the smoke represented by that particle.

To provide variety, we permanently assigned each particle one of four smoke puff variations at creation time.

The rendering process uses four values at each pixel: a unit camera-space surface normal N_{xyz}, signed camera space depth d, diffuse color C_{rgb}, and coverage mask α. We encode these values in the eight 8-bit channels of two RGBA texture maps, as shown in Figure 3. This encoding allows the renderer to efficiently read all properties at a pixel using only two 32-bit memory operations at runtime. In the first texture, the surface normal N and depth d values are stored as integers by the mapping:

$$(r,g,b,a) = \left(\frac{Nx+1}{2}, \frac{Ny+1}{2}, \frac{Nz+1}{2}, d+0.5\right) \cdot \qquad \text{Eq. 1}$$

In the second texture, channels C and α together act like a conventional texture map with an alpha channel.

Figure 3 shows the RGB and A channels of the two textures, which are created as a preprocessing step. Each subimage contains four smoke puff variations. To create these images, the render rasterizes four 3D smoke puff meshes, coloring it by the surface normal, depth, and unshaded diffuse color as appropriate for N, d, and C. The α map is computed by rendering all-white puffs on a black background, and then shifting and compositing that image in order to dilate the resulting shapes.

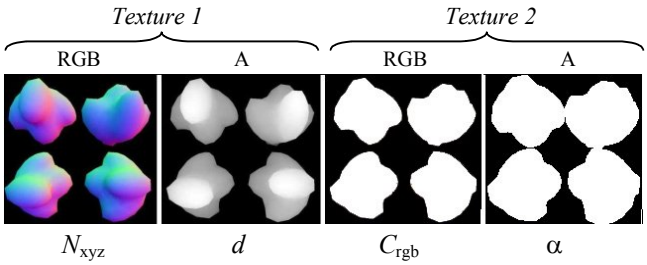

Figure 3. Surface normal, depth, color, and alpha maps for four smoke puff variations.

During smoke rendering, a pixel shader reads all components at a pixel and unpacks them into floating point values. It then computes the camera space depth and pixel color as follows.

For ambient light color K_a, light color K_L, camera-space direction L to the light source, and diffuse color C from the texture map, the pixel at (x, y) is colored by a pixel shader using quantized Lambertian shading:

$$\begin{aligned} pixel_{rgb} &= K_a(x,y) + C * K_L * q(\max(0, N \cdot L)) \\ pixel_{\alpha} &= \alpha \end{aligned} \qquad \text{Eq. 2}$$

The actual quantization function q is implemented by a 1D texture map, allowing easy artist control. A typical two-tone map is black on the left and white on the right, so that $q(x) = 0$ for $x < ½$, and $q(x) = 1$ for $x \geq ½$. Figure 1 shows a three-tone map with built-in hue. Creating q with `GL_CLAMP` border behavior automatically enforces the *max* operation during the texture fetch. The inset boxes in Figure 13 show the effects of varying the q texture from monochrome to two-tone to three tone.

Note that we allow the ambient color K_a to vary in image space according to a user-specified texture map. This implements

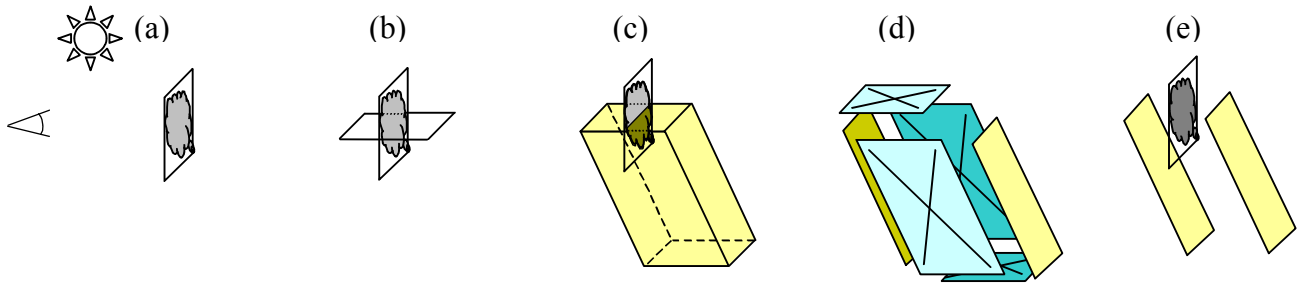

Figure 4. Construction of a billboard shadow volume. (a) Eye, light source, and billboard geometry. (b) With conceptual shadow casting proxy geometry and (c) the conventional shadow volume from that proxy. (d) Four shadow faces (marked with X's) are perpendicular to the image plane and cover zero pixels, so that only (e) three polygons need to be rendered: one billboard and two shadow faces.

gradient fills, which are popular among comic book artists. Gradients are different from the surface-normal based tone shading developed by Gooch et al. [1998; 1999], but similar in that it provides color variation in ambient-only areas.

4.2 Depth and Outlines

Nailboards offset the depth at each pixel of a billboard to create the appearance of 3D intersections with other objects. The recent introduction of a depth output register on graphics cards allows hardware implementation of nailboards.

In the same pixel shader that performs illumination, we add d to the depth at the pixel and the depth buffer test automatically produces correct intersections. A drawback of writing to the depth register is that it disables early-out depth tests, reducing the peak fill-rate on many architectures.

In Figure 3, the α-shapes are slightly larger than others. This creates the outline as a flange of dark pixels slightly behind the billboard. As desired, the outline at these flange pixels is opaque because α is one, colored black because the C is black, and farther from the camera than the particle itself because d is zero. Individual particles appear to merge together because the body of a nearby particle is closer to the camera than its neighbor's outline, so the particle bodies correctly occlude the outlines on the interior of a cloud, as shown in Figure 5.

We maintain a linear depth buffer. The conventional perspective projection matrix computes a hyperbolically interpolated depth value that distributes most depth precision close to the near plane. Were we to work with a traditional depth buffer, we would have to invert the hyperbolic values, offset them in camera space, and then invert again—a process that is both slow and reduces depth precision. With the linear depth buffer, offsetting is just addition.

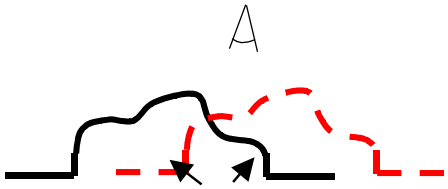

Figure 5. Top-view schematic of depth offsets for two particles. The depth test will conceal the outlines (flat flanges at the ends of each particle) on the interior of the cloud.

4.3 Shadowing and Self-shadowing

Thick smoke exhibits self-shadowing, a particularly striking effect when the smoke is between the viewer and the light source so that only the fringes of the smoke receive illumination.

We optimize and extend stencil shadow volumes [Akenine-Möller and Haines 1999] to work with billboards. For each particle billboard, we create the shadow volume that would be cast by a square in the plane with normal perpendicular to the view vector. This volume comprises only one front- and one back-facing polygon, which exactly overlap in image space (technically, there are *six* faces on the volume, but the other four are perpendicular to the view vector and need not be rendered as shown in Figure 4).

Unmodified, these shadow volumes would intersect other billboards and planes at straight lines. To create a wavy intersection, we offset the depth value of each billboard pixel and shadow-volume pixel using the pre-rendered depth map texture.

Figure 6 is a visualization of a nailboard shadow volume. The diagonal stripe is the front face of the volume, which is a single quadrilateral. Note the detailed intersection with the ground plane because of the depth offset. The shadow volume is normally invisible; we artificially shaded it to show the effective curvature.

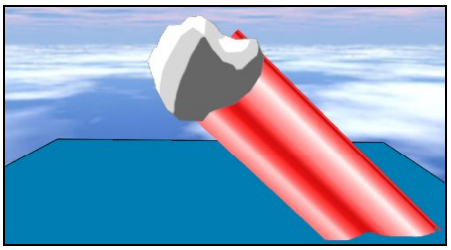

Figure 6. Visualization of a nailboard shadow volume.

4.4 Fixed-Function Variant

Most PC graphics cards now support the programmable features on which our algorithm relies, but hand-held game consoles and cell phones support only fixed-function graphics. The following rendering simplification runs on non-programmable OpenGL. It is limited to a single directional light and uniform ambient term.

The renderer maintains a mesh representation of one smoke puff. At the beginning of each frame, it renders a single texture map of that mesh with two-tone shading in four orientations. This texture map will become the billboard texture (effectively, a pre-shaded version of C from Figure 3).

When rendering this texture, the cartoon shading is computed by a well-known trick using the texture unit. We place the light vector L in the first row of the texture coordinate transformation matrix with `glLoadMatrix`, and then use `glTexGen` to set each mesh texture coordinate equal to the surface normal N. When OpenGL transforms the texture coordinate, the first component of the transformed coordinate will be the dot product, $x = N \bullet L$. We convert this texture coordinate into a quantized color by binding the texture unit to the same 1D texture q that is used in the programmable hardware implementation of our algorithm (described in Section 3.1).

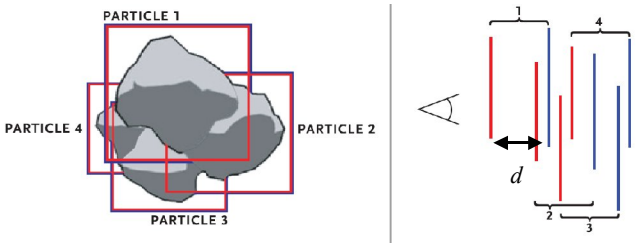

Figure 7. *left:* Front view of fixed function smoke. *Right:* Side-view schematic showing the two billboards per particle.

To produce black silhouettes and some internal contour lines for shape we add a second billboard behind each particle as shown in Figure 7. The second billboard is slightly larger, shaded entirely black, and offset along the view vector by a small distance, *d*. The thickness of the outline increases with the relative size of the black billboard. The number of internal contours drawn decreases as *d* increases; a reasonable value for *d* is the world-space width of the billboard.

5 Results

We tested our simulator and renderer by reproducing the motion and styles observed in art. We reproduced the appearance of cartoon smoke in a variety of artistic styles and created a complex animation sequence based on a film.

The top row of Figure 8 builds a rendering style to mimic style, a hand-drawn image [Talbot 1999] by enabling each effect in turn. The leftmost image (a) has outlines only, similar to Selle et al. [2004]. The center image (b) adds shading. Note that we use a gradient ambient term that is light purple at the bottom of the image and fades to dark blue at the top. The addition of lighting better reveals the 3D structure of the cloud but also introduces too much detail. The rightmost image (c) contains self-shadowing. Because the cloud is partly back-lit this removes unwanted, distracting detail and gives the cloud a more menacing look. The light is in the same position in images (b) and (c). The bottom row of Figure 8 compares the rendered cloud (d) to a hand-drawn cloud (e) in the context of a hand-drawn scene. In order to mimic the shape of the hand-drawn cloud, we disabled simulation and manually created a particle set of approximately the correct shape. The images are not identical due to our slightly inaccurate reconstruction of the scene, however the rendered image successfully mimics the original style elements.

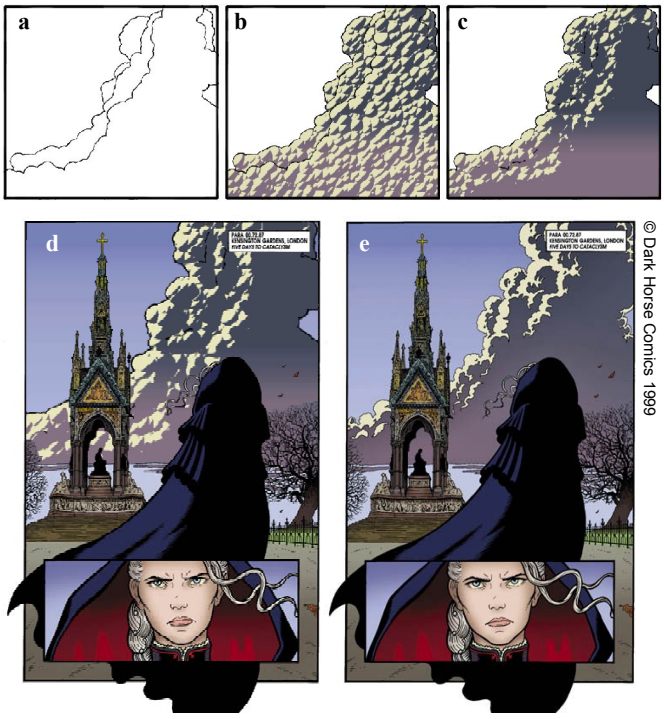

Figure 8. (a) Cloud with outlines only, (b) adding cel-shading, and (c) self-shadowing effects. (d) The previous image composited into a hand-drawn scene is comparable to (e) the artist's original.

Figure 9 is another example of reproducing the style of artist, in this case Roy Lichtenstein. We then used simulation to trace a missile's path with a particle generator attached to the missile tail. A small amount of turbulence and large amount of drag force created the desired irregular, cone shaped smoke trail. We disabled lighting and shadowing effects, leaving only outlines when rendering the final smoke trail. For comparison, the smoke in an approximation of the original scene. Figures 10, 11, and 13 show additional styles created with our method.

To further test the artist controls and performance, we recreated the opening battle scene from the live-action film "Behind Enemy Lines" (20th Century Fox, 2001) in a cartoon style. Figure 12 shows images from our NPR reinterpretation. As with other results, the video was captured from a real-time simulation and rendering. We then edited together scenes from several takes to match the editing style of the original film. The video is an example of re-creating some of the technical demands of a television animation production house or video game environment, although we are admittedly amateurs at the artistic aspects of such a process.

5.1 Performance

The system as a whole can process thousands of particles per frame at 30 fps, rendering scenes like Figure 11 with 50k particles in real time. Simulation is extremely fast because most objects do not interact. Rendering is also very efficient—current hardware has sufficient fill and vertex-processing rate to handle scenes ten times more complicated than we have shown. However, total system performance is limited by the speed of the PCI-express bus. This bottleneck occurs because it is necessary to expand each particle to a billboard and then transmit all of that geometry across the bus (4 vertices + 4 texture coordinates = 80 bytes/particle).

According to the DirectX 10 specification, the next generation of hardware will support new "geometry shaders" that can generate a billboard from a single point on the graphics card. We anticipate that this will remove bus bottleneck.

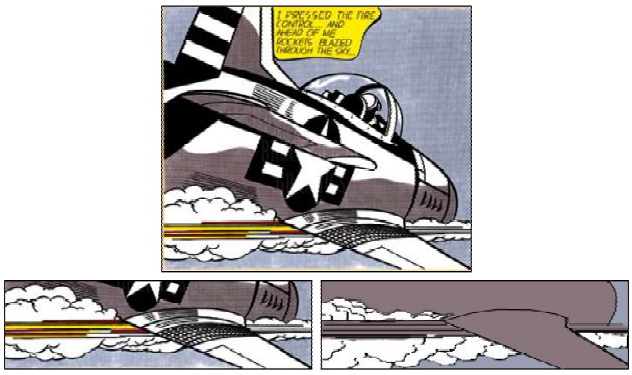

Figure 9. *Top:* Original artist illustration [Lichtenstein 1963]. *Bottom Left:* Smoke detail. *Bottom Right:* Simulated and rendered missile smoke in a modeled scene.

6 Discussion

We have described a new technique that combines and extends previous methods to create a fast cartoon smoke animation system. Selle et al. [2004] explored the visualization effect of rotating particles, which is compatible with our system, however we have not re-implemented it. It remains an open question how to effectively represent angular velocity of particles about axes far from the view vector, i.e., true 3D rotations with a 2D billboard.

Our system is limited to compressible media because it cannot efficiently preserve volume. The small number of billboard variations we use works well for smoke (particularly NPR smoke), but would likely appear very repetitive for more structured particles like asteroids or popcorn. Likewise, for large numbers of moving particles, viewers are unlikely to notice that the shadow from an individual particle does not exactly match the projected shape of that particle and is viewer dependent. Thus our nailboard shadows are not appropriate for arbitrary billboards that might be static and have easily recognized structure.

However, the nailboard shadows can be generalized in another way that stands as interesting future work. Low-polygon models with detailed bump maps are increasingly popular in video games. One drawback of such models is that their shadows are polygonal because they follow the underlying geometry; this artifact can be seen in the recent game *Doom III*. By distorting shadow volumes according to the already-present bump maps as we have done in this paper, it may be possible to create detailed and realistic shadow silhouettes.

7 Acknowledgements

We thank Colin Hartnett for editing and rigging scenes for the video in Figure 12. An NVIDIA Fellowship funds Morgan's research.

8 References

AKENINE-MÖLLER, T., AND HAINES, E. 2002. *Real-Time Rendering*, AK Peters, Ltd. 2nd Edition. Sec 6.12.3, 261—269

BUCHANAN, J., AND SOUSA, M. 2000. The edge buffer: a data structure for easy silhouette rendering. *Proc. of NPAR '00*, Annecy, Fr.39—42

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. *SIGGRAPH '01*, ACM Press, 15 –22

GOOCH, B., AND GOOCH, A. 2001. Non-photorealistic rendering. AK Peters, Natick, MA

GOOCH, B., GOOCH, A., SHIRLEY, P., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. *SIGGRAPH '98*. 447—452

GOOCH, B., SLOAN, P., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. 1999. Interactive technical illustration. *Proc. of I3D 1999*. Atlanta, GA, 31—38

LICHTENSTEIN, R. 1963. *left panel of* Whaam! Illustration. Magna on canvas. Tate Gallery, London

MARKOSIAN, L., KOWALSKI, M., GOLDSTEIN, D., TRYCHIN, S., AND HUGHES, J. 1997. Real-time nonphotorealistic rendering. *SIGGRAPH '97*, 415—420

MCGUIRE, M. 2006. A real-time, controllable simulator for plausible smoke. Tech Report, Brown Univ., Providence, RI.

MCGUIRE, M. AND HUGHES, J. 2004. Hardware-determined feature edges. In *Proc. of NPAR '04*. Annecy, Fr., 135—147

RASKAR, R., AND COHEN, M. 1999. Image precision silhouette edges. In *Proc. of I3D 1999*. 135—140

REEVES, W. T. 1983. Particle Systems—A Technique for Modeling a Class of Fuzzy Objects. *ACM Trans. Graph.*, 2:2, 91—108

SCHAUFLER, G. 1997. Nailboards: A rendering primitive for image caching in dynamic scenes. *Proc. of Eurographics*, Springer Wien, New York City, NY, 151—162

SELLE, A., MOHR, A., AND CHENNEY, S., 2004 Cartoon Rendering of Smoke Animations. *Proc. of NPAR '04*. Annecy, Fr., 57—60

SELLE, A., RASMUSSEN, N., FEDKIW, R. 2005. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph.*, 910—914

SIMS, K. 1990. Particle animation and rendering using data parallel computation. *SIGGRAPH '90*, 24:4, August, 405—413

TALBOT, B. 1999. Full-page comic panel from *Heart of Empire*. no. 1, Dark Horse Comics. p. 15

TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Trans. Graph.*, 22:3, 716—723

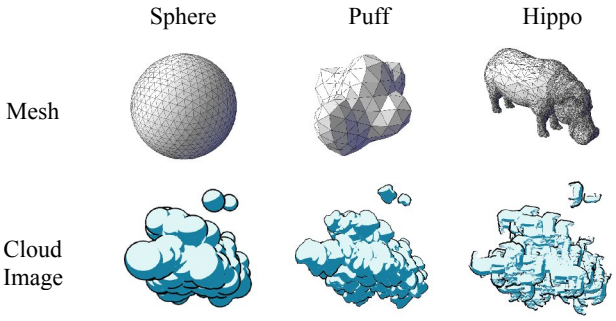

Figure 10. The billboard image used to render clouds is created from a mesh as a preprocessing step. *Left:* A sphere creates a bubbly cloud. *Center:* More complex meshes add small details. *Right*: Overly-detailed meshes create poor clouds.

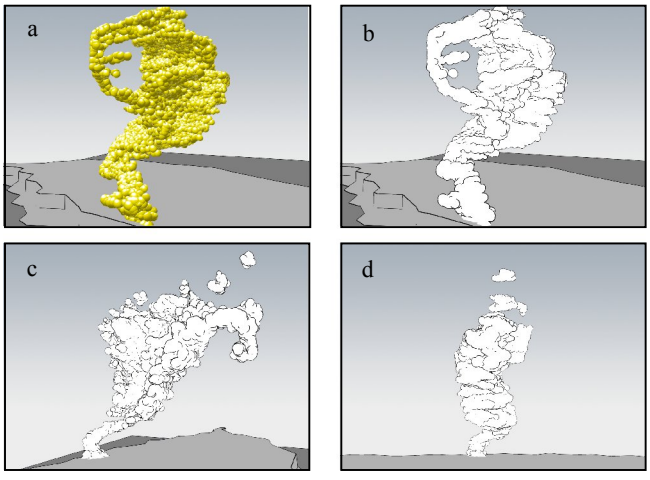

Figure 11. a) 50,000 particle cloud. b) Cartoon rendered at 30 fps using nailboards. c, d) The same cloud viewed from different angles.

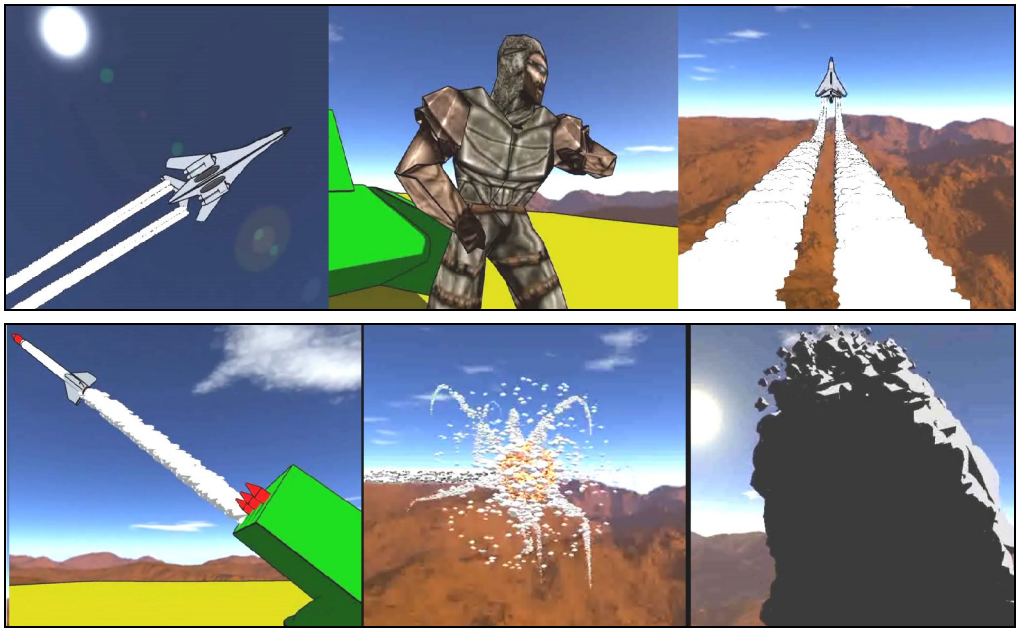

Figure 12. Frames from a long 3D animation sequence containing smoke and clouds simulated and rendered in real-time.

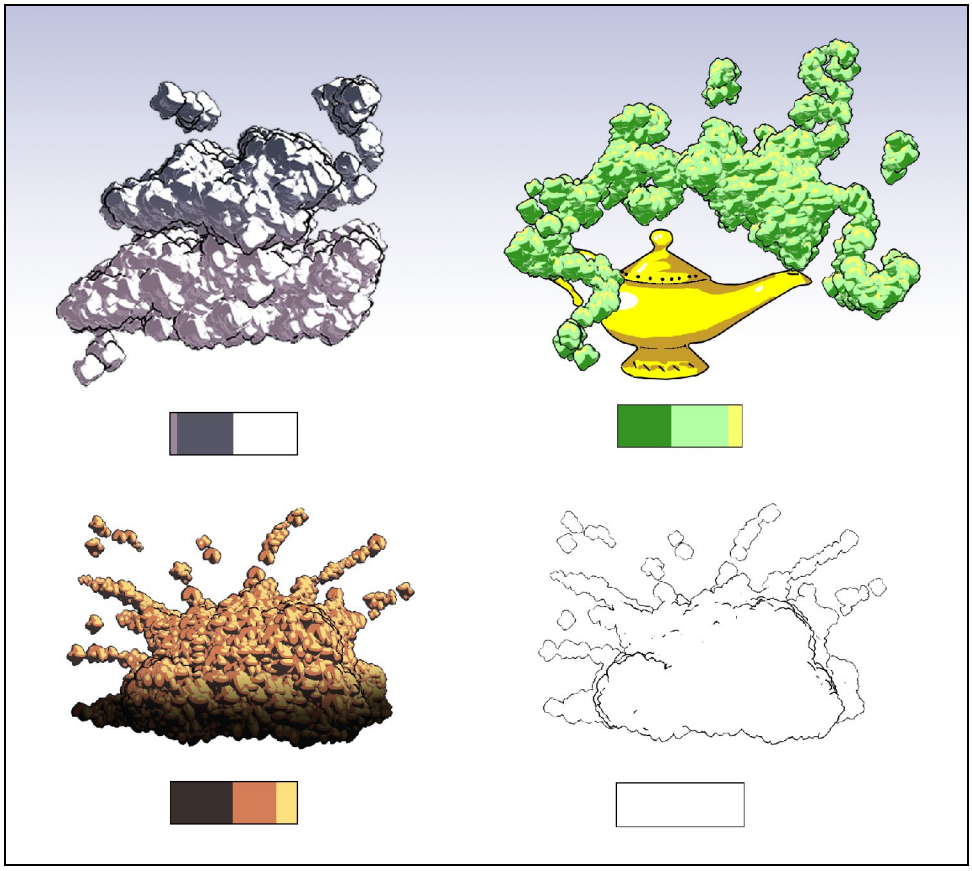

Figure 13.. a) Two-tone cloud with minor third tone for back-lighting. The inset color map is the q texture described in Section 4.1. b) Magic lamp with three-tone q shown. The bright yellow tone appears as a specular highlight but is actually viewer independent. c) Three tone-explosion with dark-to-light ambient (K_a) term. d) Identical explosion shape renderered with a solid color and outlines only.