Real-time Procedural Volumetric Fire

Alfred R. Fuller* Hari Krishnan† Karim Mahrous‡ Bernd Hamann§ Kenneth I. Joy¶
Institute of Data Analysis and Visualization (IDAV) and Department of Computer Science,
University of California, Davis, Davis, CA 95616

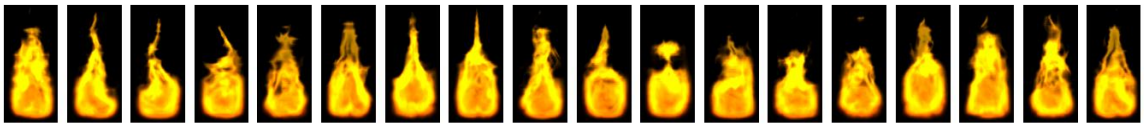

Figure 1: Procedural noise generation is used to animate the micro details of the fire to make every frame look unique.

Abstract

We present a method for generating procedural volumetric fire in real time. By combining curve-based volumetric free-form deformation, hardware-accelerated volumetric rendering and *Improved Perlin Noise* or *M-Noise* we are able to render a vibrant and uniquely animated volumetric fire that supports bi-directional environmental macro-level interactivity. Our system is easily customizable by content artists. The fire is animated both on the macro and micro levels. Macro changes are controlled either by a pre-scripted sequence of movements, or by a realistic particle simulation that takes into account movement, wind, high-energy particle dispersion and thermal buoyancy. Micro fire effects such as individual flame shape, location, and flicker are generated in a pixel shader using three- to four-dimensional Improved Perlin Noise or M-Noise (depending on hardware limitations and performance requirements). Our method supports efficient collision detection, which, when combined with a sufficiently intelligent particle simulation, enables real-time bi-directional interaction between the fire and its environment. The result is a three-dimensional procedural fire that is easily designed and animated by content artists, supports dynamic interaction, and can be rendered in real time.

CR Categories: I.3.1 [COMPUTER GRAPHICS]: Hardware Architecture—Graphics processors; I.3.3 [COMPUTER GRAPHICS]: Picture/Image Generation—Display algorithms; I.3.7 [COMPUTER GRAPHICS]: Three-Dimensional Graphics and Realism—Animation; I.3.7 [COMPUTER GRAPHICS]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture;

Keywords: Hardware Acceleration, Volume Rendering, Free-form Deformation, Fire Modeling

1 Introduction

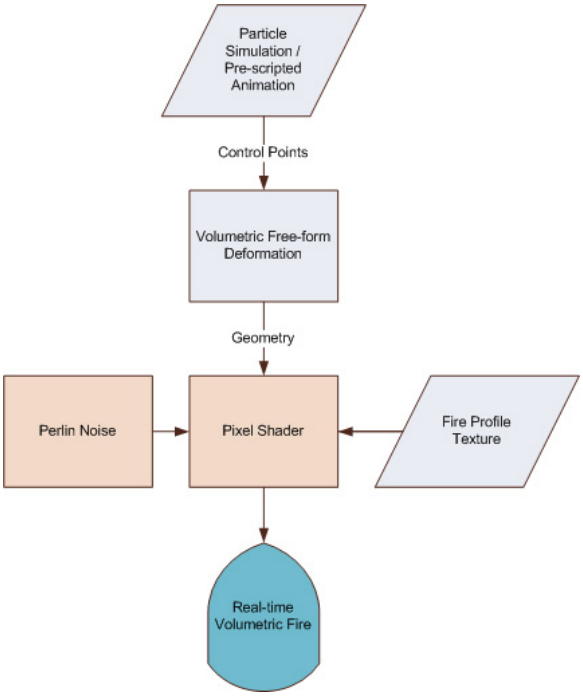

Figure 2: The pixel shader takes into account the artist-created fire profile texture (Section 2) and the geometry created by a volumetric free-form deformation driven by a particle simulation or animation script (Section 3); the shader renders it using Perlin noise (Section 4).

Interactive media applications strive to produce the most immersive experiences possible. An immersive experience is one in which the user actually feels that they are a part of the experience. This is done by creating an environment that is cohesive and consistent enough that the user accepts the illusion of its existence. Intermittent defects in a rendered environment can shatter the impression of

*e-mail:arfuller@ucdavis.edu
†e-mail:hkrishnan@ucdavis.edu
‡e-mail:kmmahrous@ucdavis.edu
§e-mail:bhamann@ucdavis.edu
¶e-mail:joy@cs.ucdavis.edu

reality for the user and destroy the immersion. In many of today's interactive media applications, real-time fire is a major source for interaction defects and visual artifacts that dispell the illusion of a given environment.

Most fires generated in current interactive media applications are sprite- or billboard-based. These techniques use animated two-dimensional images to try to create the illusion of three-dimensional fires. In ideal circumstances, these fires, through the use of clever placement, color blending, and misdirection, can produce effects that maintain a high level of believability. Unfortunately, interactive media applications rarely foster ideal conditions. When the user is allowed to explore and interact with the environment the intrinsic and unavoidable artifacts associated with these methods become distracting. For example, when two-dimensional sprites intersect the three-dimensional environment, seams become apparent at the point of intersection, and the flat and lifeless nature of the fire becomes obvious, see Figure 3. Additionally, since these sprites are based on a predefined finite sequence of animation frames, the cyclic repetition in the animation is inevitably noticed. Again, clever placement can help eliminate these visual artifacts in controlled situations, however interactive media's intrinsic unpredictability requires a more robust solution. Furthermore, these methods do not support user-driven interaction, which is becoming increasingly important for modern interactive media applications.

[Remedy 2003]

Figure 3: The game "Max Payne 2" uses sprite-based fire extremely effectively. Unfortunately, even this example suffers from the intrinsic drawbacks of sprite-based methods. The seams seen in the bottom picture are undesirable artifacts.

Both visually stunning and physically accurate rendering solutions have been developed in computer graphics, but unfortunately these methods require offline rendering. The movie industry (not constrained by real-time requirements) has amassed an arsenal of extremely robust, believable but computationally intense solutions. The approach discussed in [Nguyen et al. 2002] accomplishes a high degree of realism through a sophisticated physical simulation that requires about three minutes per rendered frame when given simple physical environments. The method described in [Lamorlette and Foster 2002] uses a curve-based algorithm combined with an extensive particle simulation to create fire that can easily interact with complicated environments, and is used in the movie "Shrek." Our method builds on the basic idea of curve-based fire design and, through the use of recent innovations in graphics hardware, creates a visually appealing real-time fire visualization capable of macro-level bi-directional interactivity.

[Nguyen 2004]

Figure 4: In a recent technology demo, "Vulcan," an attempt was made to hide the cyclic repetition artifact by covering the top of the fire with smoke. Unfortunately, even in the computer-controlled camera sequence many animation artifacts are still visible when the fire beast swings its head and the smoke is momentarily cleared.

In Section 2, we discuss how a unit fire volume is derived directly from a content artist's input. In Section 3, we describe how to manipulate the macro shape and flow of the unit fire volume through a curve-based volumetric free-form deformation. Section 4 presents the technique used to render the volumetric fire and the noise-driven method for the procedural animation of the fire's micro details.

2 Fire Profile Texture

In interactive media, consistency is crucial to creating an immersive experience. To serve this end, our fire allows content artists to customize and control the fire's look and feel. Our method accomplishes this through the use of a fire profile texture as shown in Figure 5. This texture has no restrictions on size or shape and is easily created or modified in any image editing application. Through this texture, content artists can control the color, shape and intensity of the fire to fit any desired visual style. To create a blue fire, for example, one simply colors the fire profile texture blue. To create a column of fire instead of a flame, one simply draws a straight vertical line of fire instead of a curved flame. To create an intensely vibrant flame, one simply increases the values in the alpha channel of the texture.

From this two-dimensional representation we create the cylindrical unit fire volume shown in Figure 5. The fire profile texture is swept

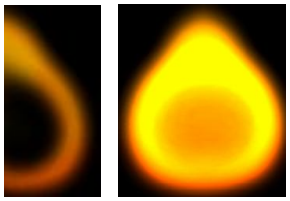

Figure 5: Left: Artist-created fire profile texture serving as the base for our volumetric rendering. It controls the shape, color and intensity of the volumetric fire. Right: Resulting unit fire volume created from the profile texture through the mapping defined by Equations (1) and (2).

around the z-axis through the mapping defined by:

$$\vec{uv} = (\sqrt{tex_x^2 + tex_y^2},\ tex_z) \quad (1)$$

$$color = FireProfileTexture(\vec{uv}) \quad (2)$$

Here, the two-dimensional vector $\vec{uv}$ is the positional vector sampled in fire profile texture for the three-dimensional location tex in the unit fire volume. The unit fire volume has coordinates in the range $[-1,1]$ for the horizontal axes x and y and $[0,1]$ for the vertical axis z.

3 Volumetric Free-Form Deformation

To manipulate the shape of the unit fire volume, we use a curve-based volumetric free-form deformation that can be adjusted at runtime. For our implementation we define the curve function $C(t)$ to be an open uniform B-spline curve defined by:

$$t_i = \begin{cases} 0 & if\ (i <= degree) \\ \frac{i-degree}{n-degree} & if\ (degree < i < n) \\ 1 & otherwise \end{cases} \quad (3)$$

$$N_{i,0}(t) = \begin{cases} 1 & if\ (t_i \le t < t_{i+1}) \\ 0 & otherwise \end{cases} \quad (4)$$

$$N_{i,p}(t) = \frac{t-t_i}{t_{i+p}-t_i} N_{i,p-1}(t) + \frac{t_{i+p+1}-t}{t_{i+p+1}-t_{i+1}} N_{i+1,p-1}(t) \quad (5)$$

$$C(t) = \sum_{i=0}^{n-1} N_{i,degree}(t) \cdot P_i \quad (6)$$

Here, n is the number of control points, the t_i values are the so-called knot values, $N_{i,p}$ is the B-spline basis function and P_i is a control point, see [Bartels et al. 1987], [Cohen et al. 2001] and [Farin 2002]. Any well-chosen C(t) could be used. The deformation can be controlled by a pre-scripted sequence of control point movements, or by moving the control points with an interactive particle simulation. We have implemented the particle simulation described in [Lamorlette and Foster 2002], where the velocity of a particle at position x is defined by:

$$\frac{dx}{dt} = \vec{v} + \vec{w}(x,t) + \vec{d}(T) + \vec{c}(T,age) \quad (7)$$

$$c(T,age) = -\beta \cdot g \cdot (T_0 - T) \cdot age^2 \quad (8)$$

where:

- t is time and T is the temperature of the fire.
- $\vec{v}$ is the velocity of the fire.
- $\vec{w}()$ is the wind velocity as a function of position and time.
- $\vec{d}()$ is the velocity attributed to high energy dispersion as a function of temperature.
- $\vec{c}()$ is the velocity attributed to thermal buoyancy as a function of the temperature and particle age.
- β is the thermal expansion coefficient of and ideal gas ($= 1/T$), g is a gravitational constant, and T_0 is the environment's ambient temperature (usually about 300K).

In a fully interactive environment this simulation could also receive interaction forces from the player and the environment.

Figure 6 shows an example of an ideal curve-based volumetric free-form deformation. This the model employed by the method described in [Lamorlette and Foster 2002] and used in the movie "Shrek." This model is currently intractable on modern graphics hardware. A pixel shader operates on P‘ and returns P, and not the other way around as defined by $D(P)$. This problem is known as inverse parameterization. This is a well-recognized problem that cannot be solved analytically, and its numerical solution requires significant computation and exhibits many robustness problems. We can, however, easily solve for P‘ from P through the function $D(P)$ on the CPU. This means that we can deform a fixed number of points without any loss of detail, but cannot deform the entire volume, which has virtually infinite detail once noise is applied, in real time. Additionally, the pixel shader can only render to a location occupied by a triangle fragment; thus, triangles must be rendered that cover every point P‘ of the deformed fire. This presents a complication since the final pixel coverage of the rendered fire is not known a priori and the degree of frame-to-frame coherence can vary substantially.

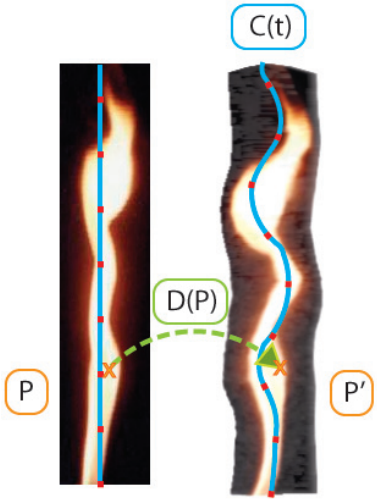

Figure 6: Mapping of coordinate space for curve-based volumetric free-form deformation:

- P is a point in texture coordinate space.
- P‘ is the deformed location of P in world space.
- $C(t)$ is a curve that defines the coordinate transformation.
- $D(p)$ is a non-linear deformation such that $P‘ = D(P)$.

Our method solves these problems by using the graphics hardware to approximate $D(P)$ on discrete intervals. By constructing a lattice around the deformed volume in world space and associating each point of the lattice with texture coordinates in the unit fire volume, the graphics hardware implicitly evaluates the mapping $D(P)$ when it interpolates the texture coordinates, see Figure 7. Unfortunately,

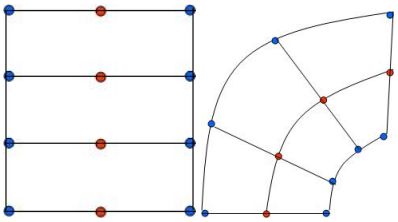

Figure 7: Left: Lattice in uniform texture space. Right: Lattice in deformed world space. The associated blue points are the same points from the perspective of the graphics card. This lattice mapping implicitly evaluates the deformation function $D(P)$.

since current graphics hardware only supports linear interpolation the discrete nature of this lattice creates discontinuities in the first derivative of the curve on the borders of each lattice section. However, by further refining the detail of the lattice we can approximate the continuous curve-based volumetric free-form deformation to an arbitrary degree of accuracy. The Nyquist theorem when used to define for the resolution of this lattice, requires us to use two lattice sections per pixel. However, the actual lattice resolution required is dependent on the curvature of $C(t)$, the final pixel coverage of the deformed volume, and the length of the fire. Generally, the lowest resolution with least visual artifacts is desirable. This resolution can range anywhere from a single section to several hundred sections. Fortunately, the performance impact of additional lattice sections is relatively small so the required lattice resolution can be over-estimated.

Additionally, this lattice also solves the pixel coverage problem fairly efficiently. By dividing the lattice into a finite set of cubes and only drawing the required number of triangles within these cubes, the coverage of the pixel shader is minimized, and we ensure that every section of the fire is rendered.

3.1 Lattice Construction

The lattice surrounding the fire must be flexible while maintaining appropriate texture coordinates for rendering. To do this, we take advantage of the $[0,1]$ domain of normalized B-spline curves. For each control point P_i in the center spline we sample the normal vector $\vec{N}_i$ and velocity vector $\vec{V}_i$ of the spline at $t = \frac{i}{n-1}$, $i = 0 \ldots n-1$, where n is the number of control points. This is the position of the curve that is most affected by that particular control point. We then generate four additional curves by placing control points on the corners of a square in the plane defined by P_i and $\vec{V}_i$ and centered at P_i. The curve can additionally define a "fourth dimension" of our central curve using the radius r of the fire so that the length of an edge of this square is $2r$. We align all n squares along their respective central curve normal $\vec{N}_i$ to reduce twisting. The central curve and the four corner curves are shown in Figure 9.

We exploit the fact that each B-spline is defined over the domain $[0,1]$ regardless of arc-length. By sampling the center curve by length c times, where c is the resolution of the lattice, we obtain c values of t. We use these values of t to solve Equation (6) for each corner curve. This step yields c sets of correlated positions on the four corner curves which define the lattice sections as seen in Figure 9. We also assign each point in our lattice the texture coordinate $tex_i = (x, i/(c-1), z)$, where x and z are 1 or -1, depending on which corner curve they are from, and i is the level of the current lattice. This approach effectively generates the texture coordinates for the volumetric free-form deformation. The result of using these

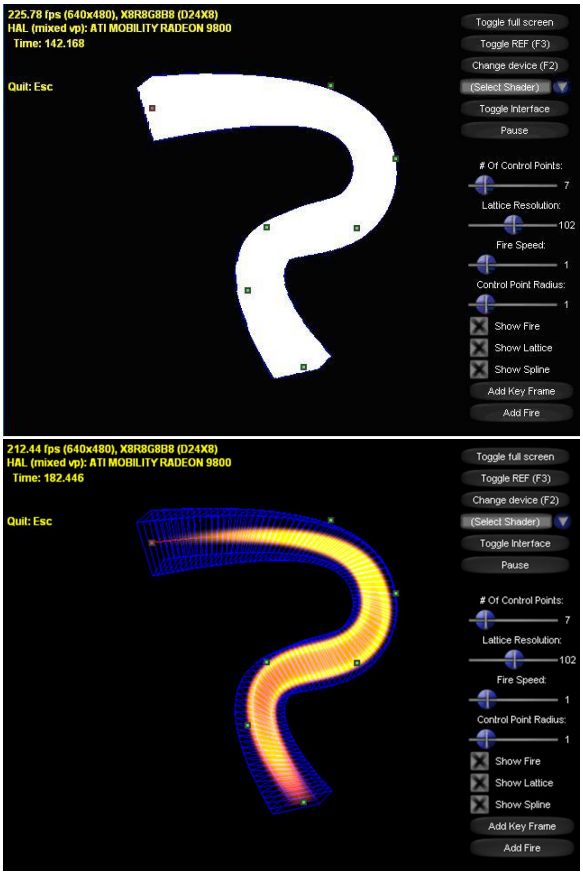

Figure 8: Curve-based volumetric free-form deformation. Top: The screen space in which the pixel shader is active, shown in white. Bottom: The result of deforming the unit fire volume from Figure 5.

texture coordinates in the mapping defined by Equations (1) and (2) is shown in Figure 8 and 9.

4 Rendering Volumetric Fire

4.1 Hardware-accelerated Volumetric Rendering

To render the volumetric fire, we use a technique similar to that described in [Cabral et al. 1994]. Every time the volume is rendered it is sliced into evenly spaced view-aligned slices as seen in Figure 10. Each slice is rendered through a pixel shader that uses the fire profile texture and additive blending to create a volume rendering that is equivalent to a ray tracer that uses evenly spaced samples. This can be done efficiently by implementing a highly optimized cube slicing algorithm and applying it to each segment of the lattice. Additive blending is used to handle translucency to account for the fire's emissive nature. Other types of blending can also be used to create additional effects such as using cartoon-like fire, see Figure 11. The slice spacing is selected based on performance requirements, and it can be adjusted on the fly. Larger spacing creates less slices and decreases rendering time.

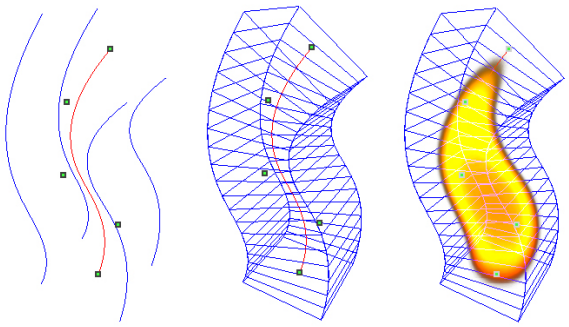

Figure 9: Left: Center curve with control points and four corner curves. Middle: Lattice constructed around the deformed volume using the four corner curves. Right: Rendered unit fire volume that is implicitly deformed by the graphics hardware.

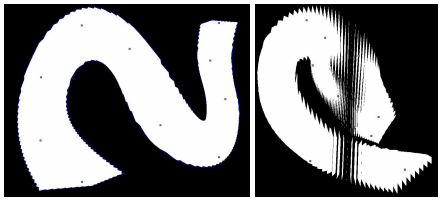

Figure 10: View-aligned slices are used to emulate ray tracing in hardware. Left: Head-on view. Right: Side view.

4.2 Noise

We have described how to manipulate the shape of the fire on a macro level. To actually create the micro-level details and animation that bring the fire to life we procedurally generate spatially consistent noise in the pixel shader. The ideal type of noise to use in this situation is Perlin Noise. There have been a number of attempts to adapt Perlin Noise for use on graphics hardware. The most notable of these are Improved Perlin Noise, described in [Green 2005], and M-Noise, described in [Olano 2005]. Improved Perlin Noise more closely approximates real Perlin Noise, while M-Noise is substantially more efficient. Three-dimensional Improved Perlin Noise uses six dependent texture look-ups while M-Noise only uses two independent look-ups. A single octave of M-Noise uses about 40 pixel shader instructions and can be compiled for pixel shader 2.0 hardware, while Improved Perlin Noise uses about 64 instructions per octave and can only be compiled for pixel shader 2.a or newer hardware. For most applications M-Noise is the better choice.

A turbulence function is employed to create a higher degree of realism. The turbulence function sums different octaves of noise to simulate natural fractal phenomena [Mandelbrot 1983], and it is defined by Equation (9). The *gain* and *lacunarity* depend on personal preference and control relative amplitude and frequency of each octave of noise. Common values are .5 and 2, respectively. Increasing the number of octaves refines the fire and increases rendering time. For applications requiring highly realistic fire simulations, three or more octaves are usually required. For applications that are meant to produce more stylized or cartoon-like enviornments, two or fewer octaves are usually sufficient.

To create the deformed cylindrical volume the pixel shader is given as input the three-dimensional texture coordinates in the unit fire volume defined by the volumetric free-form deformation. These coordinates are mapped to the two-dimensional fire profile texture coordinate vector $\vec{uv}$ as defined by Equations (1) and (2). To create

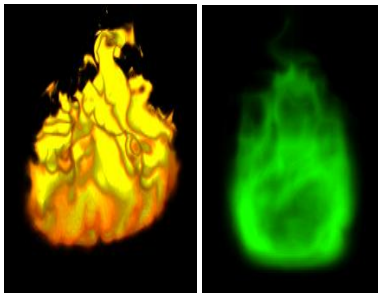

Figure 11: Left: Making the flame opaque results in fire that looks hand-drawn. Right: Tinting the flame quickly changes its appearance.

micro-level details we refine that mapping to include noise. Since Perlin Noise is smooth and spatially coherent, it can be employed to create small flames and fire plumes in the deformed unit fire volume by simply offsetting the vertical texture coordinate by noise as defined by Equation (13). We also enhance the mapping by taking into account several aspects of the deformation.

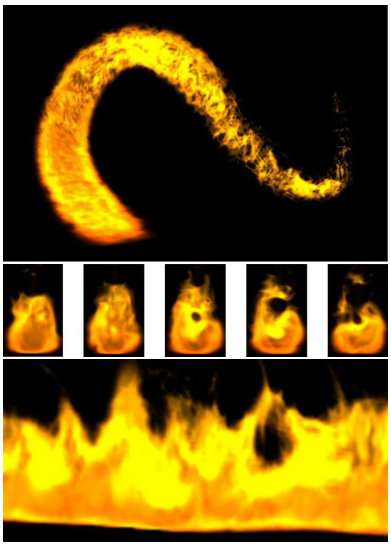

Figure 12: Noise is used to animate the detail of the fire. The frequency of the noise is scaled to world coordinates in the pixel shader to eliminate stretching and scaling artifacts. The middle image shows a procedurally generated animation sequence. The bottom picture shows the smooth spikes and plume created by offsetting the vertical texture coordinates with noise, see Equation (13).

When the fire is enlarged by either elongating the central curve or adjusting the radius associated with a control point, it should not stretch the effects of the noise. This would create visual artifacts across rendered frames. To address this problem we dynamically scale "noise space" according to Equation (10). Since the radius and length of the fire are known at any given point in world space we scale the horizontal and vertical axes of noise space by the radius and length, respectively.

The fourth dimension of noise space is time and it remains unscaled. Additionally, since various types of fire have different desired levels of detail (and since the mapping has no concept of world space scale) we multiply $noise_{scale}$ by an application-specific frequency. Each version of noise has its own inherent frequency, so we correct each noise function such that they have a base frequency

of one. This makes the $frequency$ factor in Equation (10) the spatial frequency of the micro-level detail in world space.

Each triangle fragment in the pixel shader has an associated position in noise space as defined by Equation (11). This position is found by scaling the unit fire volume texture coordinates by the $noise_{scale}$. The fourth dimension is set as $time$ to enable the use of four-dimensional noise. Additionally, in order to create a sense of upward flow we offset the vertical axis by $-time$. In order to make each fire unique, we add a different constant offset to the noise position for each instance of the fire. With this position in noise space we can apply the turbulence function defined by Equation (9). We multiply this value by a customizable kernel $f(tex_z)$ and add it to the vertical component of our texture look-up, see Equation (13). $f(tex_z)$ can be any kernel, but we have found that a stability factor times the square root of the height leads to visually pleasing results. Finally, we use Equation (14) to look up the color at the offsetted texture coordinates. When taking into account both the physical properties of the deformation and the desired effects of noise, the mapping is defined by these equations:

$$turb = \sum_{i=0}^{\#octaves} gain^i \cdot noise(position \cdot lacunarity^i) \quad (9)$$

$$noise_{scale} = (radius, radius, length, 1) \cdot frequency \quad (10)$$

$$noise_{pos} = (tex_x, tex_y, tex_z - time, time) \cdot noise_{scale} \quad (11)$$

$$u = \sqrt{tex_x^2 + tex_y^2} \quad (12)$$

$$v = tex_z + f(tex_z) \cdot turb(noise_{pos} + offset) \quad (13)$$

$$color = FireProfileTexture(u, v) \quad (14)$$

Results of this mapping are shown in Figure 12.

5 Conclusions

We have presented a method for creating and rendering uniquely animated, visually impressive and fully three-dimensional volumetric fire in real time.[1] Our method is easily customizable by content artists to match application-specific needs. It is efficaciously animated though either a pre-scripted sequence of control point movements or an intelligent particle simulation. The performance of our volumetric fire can also be fine-tuned to meet a wide range of requirements.

The performance of our real-time volumetric fire method depends on occupied screen space, size of the fire, slice spacing, number of octaves of noise, lattice resolution, type of noise and graphics hardware. All these factors influence overall performance. Fortunately, almost all these factors can be adjusted to fit any demand. Table 1 shows performance results for a unit-radius fire with a slice spacing of 0.2, taking up approximately 90,000 pixels on a nVidia GeForce 7800 GT graphics card.

Pixel Shader Version	Octaves	Noise Type	Lattice Resolution	Frames per Second
N/A	0	No Fire	N/A	197
2.0	1	M-Noise	35	157
2.0	1	M-Noise	135	141
2.a	3	Improved Perlin	35	94
2.a	3	Improved Perlin	135	66
3.0	3 of 4D	Improved Perlin	35	45
3.0	3 of 4D	Improved Perlin	135	30

Table 1: Performance results for a unit-radius fire with a slice spacing of 0.2, taking up approximately 90,000 pixels on a nVidia GeForce 7800 GT graphics card.

6 Acknowledgements

This work was supported in part by Electronic Arts and the Institute for Data Analysis and Visualization (IDAV) at University of California, Davis. We would like to thank the members of the Visualization and Computer Graphics Research Group of IDAV.

[1] Demonstrations of our method are available at the following sites:
codec: http://idav.ucdavis.edu/%7Ealfnoodl/tscc.exe
video: http://idav.ucdavis.edu/%7Ealfnoodl/VolumetricFireDemo.avi
video: http://idav.ucdavis.edu/%7Ealfnoodl/vfire.avi

References

BARTELS, R., BEATTY, J., AND BARSKY, B. 1987. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann Publishers, San Francisco, CA, USA.

CABRAL, B., CAM, N., AND FORAN, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 Symposium on Volume Visualization*, ACM Press, New York, NY, USA, 91–98.

COHEN, E., RIESENFELD, R., AND ELBER, G. 2001. *Geometric Modeling with Splines: An Introduction*. A K Peters, Wellesley, Mass.

EBERLY, D. H. 2004. *Game Physics*. Series in Interactive 3D Technology. Morgan Kaufmann Publishers, San Francisco, CA, USA.

FARIN, G. 2002. *Curves and Surfaces for Computer Aided Geometric Design*, fifth ed. Academic Press, San Diego, CA, USA.

GREEN, S. 2005. Implementing improved perlin noise. In *GPU Gems 2*, M. Pharr, Ed. Addison-Wesley, March, ch. 26.

LAMORLETTE, A., AND FOSTER, N. 2002. Structural modeling of flames for a production environment. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, 729–735.

MANDELBROT, B. B. 1983. *The Fractal Geometry of Nature*, revised and enlarged ed. W.H. Freeman and Co., New York, NY, USA.

NGUYEN, D. Q., FEDKIW, R., AND JENSEN, H. W. 2002. Physically based modeling and animation of fire. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, 721–728.

NGUYEN, H. 2004. Fire in the "vulcan" demo. In *GPU Gems*, R. Fernando, Ed. Addison-Wesley, March, ch. 6.

OLANO, M. 2005. Modified noise for evaluation on graphics hardware. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM Press, New York, NY, USA, 105–110.

PERLIN, K. 1985. An image synthesizer. *Computer Graphics 19*, 3, 287–296.

REMEDY. 2003. *Max Payne 2: The Fall of Max Payne*. Rockstar Games.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, 151–160.