Real-time photorealistic stereoscopic rendering of fire

Benjamin M. Rose*, David F. McAllister**
Department of Computer Science, N.C. State University, Raleigh, NC 27695

ABSTRACT

We propose a method for real-time photorealistic stereo rendering of the natural phenomenon of fire. Applications include the use of virtual reality in fire fighting, military training, and entertainment. Rendering fire in real-time presents a challenge because of the transparency and non-static fluid-like behavior of fire. It is well known that, in general, methods that are effective for monoscopic rendering are not necessarily easily extended to stereo rendering because monoscopic methods often do not provide the depth information necessary to produce the parallax required for binocular disparity in stereoscopic rendering. We investigate the existing techniques used for monoscopic rendering of fire and discuss their suitability for extension to real-time stereo rendering. Methods include the use of precomputed textures, dynamic generation of textures, and rendering models resulting from the approximation of solutions of fluid dynamics equations through the use of ray-tracing algorithms. We have found that in order to attain real-time frame rates, our method based on billboarding is effective. Slicing is used to simulate depth. Texture mapping or 2D images are mapped onto polygons and alpha blending is used to treat transparency. We can use video recordings or pre-rendered high-quality images of fire as textures to attain photorealistic stereo.

Keywords: rendering, stereo, real-time, fire, flame, billboarding

1. INTRODUCTION

Fire is a chemical reaction creating flames that are light emitting gas left more buoyant than air. Flames move in three dimensions in direction is that are difficult to predict. Parts of the flames may be transparent or semi-transparent. The color of light given by sections of flames depends on the temperature of the gas in that section, blue being hotter than yellow and yellow hotter than red. The temperature of the fire can be affected by many factors. For a fire in an open atmosphere, the more oxygen and hydrogen that is supplied, the hotter the fire will burn given that the source of fuel remains constant. If the fire were to be provided with a higher ratio of oxygen to fuel or fuel to oxygen, parts of the fuel or oxygen will not be part of the reaction, thus less energy is released: the fire will burn with less intensity and release more unburned particles in the form of smoke into the air.

Different fuels cause fire to behave in different ways. A steady stream of gaseous fuel will yield a fire with smooth curves comprising its silhouette as is indicated in Figure 1. A solid fuel, such as a block of wood, will produce a fire with more sporadic pointed flames jutting out in many directions, while flowing up. The key difference between the gaseous and solid fuels is that there is a force exerted by the stream of gaseous fuel. The section of gaseous fuel that is burning will quickly heat its neighboring gaseous fuel sections to their ignition temperatures. The result is a force being exerted on what appears to be fire, but is actually the force of the fuel itself.

In a gravity influenced atmosphere, the glowing gases will move upward due to diffusion of density. The surrounding air is denser than the gas. Gravity applies a downward force to denser gases, and the less dense gas must flow up and displace the denser air. This is the reason why flames are upward pointed when its fuel source is not acted on by an external force.

* BenRose3d@gmail.com
** David@cmonline.com

Figure 1. Fire with steady stream of gaseous fuel.

The density of the fuel before ignition can also influence the behavior of fire. A denser material with the same molecular structure will take longer to ignite than a less dense material.

Wind exerts a force on the heated, illuminating gas, and hence has a heavy influence over the motion of fire. Wind can also extinguish a fire. If the wind is strong enough to move and disperse the hot gas to preclude the already ignited gases from heating the neighboring gases or fuel, the fire will expire.

Lamorlette[1] points out that computer monitors do not have enough contrast to show fire for the semi-transparent, moving entity where only bright backgrounds would be visible through it and it is better to have a dark background.

Our primary concern here is the visual behavior of flames for applications in virtual reality. Fire moves at a rate that the human visual system is unable to collect images, so there is motion blur for the viewer. Flames vary in color and transparency, from flame to flame as well as within a flame. In the examples below we render a wood fire that is unaffected by wind. The solution presented herein can be applied to most types of fire and external environments.

2. MONOSCOPIC TECHNIQUES

2.1 Background

Fire was first modeled with particle systems by Reeves[5] in 1983. In Reeves' work particles would be generated and modified with every new frame in the animation: the location, acceleration, velocity, color and size of the particles. Reeves used two particle systems to produce a wall of fire and explosions on a planet in the movie *Star Trek II: The Wrath of Khan*[12]. Although simple to implement, the solution presented does not yield photorealistic images, a goal of this research.

Perlin[6] utilized his own randomization noise equations to produce turbulent fields, that when used in conjunction with a color-assignment function, could produce a fire-like image. Although Perlin noise functions can be used to produce images of fire in real-time, the images do not have sufficient realism.

Beaudoin[11] generates images of fire by creating a skeleton of individual flames using a velocity field, some starting from the surface of the burning object, some starting from a point off of the surface simulating how flames can break away from the fire. A three-dimensional model is created using the flame skeletons and numerical methods used to emulate the wide base and narrow top portion of flame. Rendering performance is not real-time and the resulting images are not photorealistic.

Lamorlette1 approaches fire in a similar way to Beaudoin where flames are created independently, then merged together. The complexity of this method exceeds that proposed by Beaudoin but provides the animator as much control of the look and behavior of the fire as possible. The images are photorealistic, although they cannot be computed in real-time using standard commercial hardware.

Nguyen[2] implements incompressible Navier-Stokes equations and gives attention to the often ignored gas expansion due to combustion. Nguyen treats the expansion of gas at the moment of combustion by approximating the reaction zone with an implicit surface. Incompressible Navier-Stokes equations are applied to the fuel and the heated gases, independent of each other, then the two sets of data are revisited to ensure the conservation of mass in the system. Nguyen was able to generate fire for various fuel types, both solid and gaseous, at a rate of approximately 3 minutes a frame using a Pentium IV.

Stam[6,7] devised an approximation to the Navier-Stokes equations that allowed him to reduce computation times of simulation-based modeling and rendering of fire to "interactive" rates. However, he indicates a time of 2.7 seconds per frame to render a 9000 sample particle animation on a 700MHz Pentium III; not real-time. Stam's method produces photorealistic images of fire and could potentially be computed in real-time for stereographic rendering.

Our method is based on using photorealistic monoscopic images of fire and any of the previous three methods could be used to produce such images. Since our emphasis is to determine how to use such images to generate a realistic stereo rendering of a fire, for expediency we chose to capture images using video. Hence, our method for producing textures, while easy to implement, is limited to rendering fire that can be represented by applying image processing to video frames.

2.2 Textures

Two dimensional textures are images or patterns that are "pasted," or mapped to surfaces of objects. Textures are patterns that can be used to simulate surface properties without creating the geometry that would be necessary to produce the same appearance. Textures can be combined with other textures, lighting, and object properties to create what appear to be very complicated geometrical objects and environments. We will use two dimensional semi-transparent textures in conjunction with alpha blending to simulate the speed and photorealism of fire. We will apply these textures to flat surfaces called billboards to provide depth for stereo imaging.

A texture map is a specification of the relationship of points on a section of the target surface of an object to a section of the texture. Two and three dimensional textures can be created from scanned images such as photographs, video, drawings, etc. Procedural textures are textures whose properties are computed by an algorithm.

For scanned images, a rule or map is required to associate each pixel with the location of points on the surface of the object. In addition, a rule is required to use a property or properties of a pixel on the texture image to modify the property of interest associated with surface points. In order to modify color values by applying a texture, the illumination values, surface color and other previously mapped textures must be combined with the current texture to be mapped. This is known as *alpha blending*, which is the process of combining two colors based on a parameter known as an *alpha channel* associated with each color.

We use alpha blending to simulate transparency in rendering fire. There are many different ways to use alpha-blending, the most common being linear interpolation. If $0 \leq \alpha \leq 1$ is the value of the current texture color to be blended, C_1, and the underlying color present on the surface is C_2, then the alpha blended color that replaces C_2 is $C = \alpha C_1 + (1 - \alpha) C_2$. Hence if $\alpha = 0$, the color C_1 is to be completely transparent and $C = C_2$. Using this simple linear alpha-blending to blend semi-transparent fire textures positioned in front or behind each other results in fire that has a flat appearance with an abnormally large yellow core. We implemented a slightly more complex method of alpha blending, discussed in section 4.3.

The value of α is used along with the depth and color values of all points in the lines of sight from the viewpoint to determine the final pixel value displayed. Because the color values of an object behind a transparent object must be computed before computing the resulting color values looking through the transparent object, care must be taken that the order objects are added in the rendering pipeline are from back to front order.

As an object increases distance from the viewer, details are less visible, less information is required to represent the object and less resolution is required to represent the textures for the object. In addition, applying full resolution textures to a distant object will result in pixel position conflicts creating aliasing. To treat this problem, common graphics APIs often provide mipmapping or downsampling objects and textures to exploit the reduced information to enhance rendering speed and avoid the position conflicts. In our case, the result is better looking, antialiased fire. We have applied mipmapping using the OpenGL API for this research.

2.3 Fluid simulations

Fluid modeling techniques generate three-dimensional models using a subset of gaseous phenomena properties with an acceptable loss of precision. Real-time rendering of fluid simulation representations is difficult because of the calculation complexity required. Reducing the complexity of fire simulation formulas without sacrificing visual quality has been advanced by many researchers. Obtaining aesthetic, realistic results using simulation involves essentially three steps: building the data for the model, extracting and computing data relevant to the surface, and using that surface data in one of a number of ways in conjunction to other colors of the scene (when the fluid is transparent or semi-transparent) to calculate the correct colors in the scene. Each portion presents its own challenges, and there are many options and intricate steps in each. There is a large performance impact on the choices one makes when determining which implementations of these steps to perform.

The model data can be stored as vectors, points, values in a grid, or a combination. Many implementations have incorporated several aspects of fire that are not displayed; rather, these attributes of the fire are used to compute the properties of the next model in the succession to be used for rendering. Essential attributes that are not directly rendered include density, velocity, and viscosity.

The equations that describe fluid motion are called the Navier-Stokes equations. They are described by Stam[6] as "a precise mathematical model for most fluid flows occurring in Nature." Those working with Navier-Stokes equations have found them too complicated and computationally expensive to be sufficient for real time animation in computer graphics. Instead, people like Stam[6,7] have concentrated on creating adequate approximations of the Navier-Stokes equations that produce a pleasing and realistic but real-time alternative.

There are principally two types of fluid dynamic representations, Eulerian and Lagrangian. Both representations, or methods, involve estimating behavior of equations in different ways. Vesterlund[8] gives a good metaphor for helping the reader understand the differences in Eulerian and Lagrangian methods. Vesterlund describes the Eulerian method as putting a grid around and throughout a fluid, where the attributes of the liquid, such as velocity, were stored for every grid point. He describes the Lagrangian method as a system of corks floating on the surface of a liquid.

Jos Stam has contributed extensively in the fields of fluid dynamics and computer graphics. Stam's fluid solver involves two grids of the same size, one for density values and the other for velocity vectors. Both are given initial data, then updated in small time steps using his variations of the Navier-Stokes equations. Updating the density grid involves adding density to the grid, diffusing the density values throughout the grid, and density movement along velocity grid vectors. Adding density to the grid is straightforward, and is left to the artist using the system to determine how much and where. Diffusion affects each grid cell using its neighbors' density values, the diffusion rate, the time step, and the number of cells in the grid.

First, the time step, diffusion rate, and size of the grid are used to compute a variable to be used in every cell in the density grid. Second, the cells of the grid are all traversed and the density values are set based on the neighboring cells in a manner that avoids creating an unstable diffusion.

The computed value used to modify each cell's density value without proper care would change sporadically, going negative, or growing beyond realistic values. "Any technique that performs only explicit updates between adjacent cells will fail."[7] Stam incorporated an implicit time step approach using densities that when diffused in reversed would result in the densities at the previous time step. Once every cell's value has been adjusted, the boundary cells are modified once more to adjust for them being boundary cells.

The density in the density grid's cells must flow along with the velocity vectors in the velocity grid in order to produce fluid like motion. This is accomplished by using a technique similar to that used in the density diffusion portion, where the formula is traced backwards in time. In any given time step, each cell is computed in the following fashion. Starting in the center of the cell, the flow into that position is traversed towards its origin for one time step. The closest N cells to the final position are linearly interpolated and the resulting value is stored as the cell in question's density, where N is the same number of neighboring cells used in the density diffusion portion. The boarder cells are then modified again. Stam remarks, "the method is stable because it uses linear interpolation; the new values are never larger than the maximum density of the previous time step."

Stam's velocity solver is essentially the same as the density solver, with the exception that the velocity solver employs a method to conserve mass. It is important that the velocity leaving the cell equals that coming into it to generate good looking swirls for fluids. A Poisson equation is used to accomplish this task.

Stam's work is currently the only "interactive" simulation model available, yet there is no rendering information included. There still remains the challenge of moving from a simulated model of fire to a rendered image and the problem of rendering at animation speeds. This paper will explore solutions found in similar papers with realistic but not real time results. The data must be rendered correctly by first translating it into graphics primitives.

Traditionally volumetric rendering involved computationally traversing rays through pixels in the viewplane through the scene to get the appropriate color values for that pixel. There has been work on making this form of volumetric rendering more efficient, but it remains computationally expensive. A much faster alternative to traditional volumetric rendering using standard polygon oriented hardware is surface rendering.

Surface modeling partitions an object into polygons, or flat surfaces connected along edges. *Point splatting* is the technique of using ellipses with alpha transparency masks for rendering an object's surface in a scene. Point splatting using *surfels*[4,8] offers a more efficient alternative than using polygons. Surfels (surface elements) are essentially data objects that store all the relevant information required for rendering the object's surface. A surfel can hold any type of information, but usually contains the position, normal, radius and color associated with that surfel's point in space. The radius is used when calculating the alpha mask, gradually blending more towards the outer edge of the ellipses. Adding an acceptable shading scheme for the surfels specifically tailored for fire animations and performing ray tracing on the scene yields a smooth, attractive surface for the fire[8].

For surface rendering to be used, the surface must be extracted from the fire model, if it has not been already. Surfaces of fluids can be defined as implicit surfaces[8]. Implicit surfaces are surfaces that are defined by points that satisfy some conditional. As an example, if a simulation of fire has been run and a density grid obtained, the implicit surface of the fire could be considered where the density of the cell of the grid is not zero, and one of its neighboring cells' density values is. Implicit surfaces benefit from point splatting because they are comprised of points in 3D space.

Most fluid dynamic modeling solutions are realistic but are too computationally intensive for implementing real time[1,2,8], or they are tractable but do not yield realistic looking flames[10]. Stam's solution gives realistic results and may be able to achieve real time performance using modern hardware. We did not test this assertion.

3. EXTENDING MONOSCOPIC TECHNIQUES

There are many issues involved in extending monoscopic techniques to stereo. A technique that appears feasible to implement in real time for a monoscopic environment can become too computationally expensive to implement in stereo. Or the monoscopic technique may not provide the necessary depth information to permit rendering of left and right eye views.

According to Adabala et. al.[10], to obtain acceptable visual representations of flame from a fluid simulation technique involving a grid, the animator or developer must be skilled at modifying non-intuitive controls. Beaudoin[11] points out that every solution that relies on a simulation of fire is often difficult to predict what effect on the fire a given value will have. For example, initial densities and initial velocities must be provided as must the velocity and density injectors.

Fluid simulations of fire can create three dimensional models of fire and needs only be modeled once per frame. Volumetric rendering must be computed for each of the two eyepoints. The fastest way to do this is by extracting the surface and point splatting surfels, computing colors, transparency, and intensity for every surfel, computationally expensive. Add to this the cost of modeling the fire as well as the other objects in a given scene, and simulations lose their appeal. For this reason, our solution focuses on extending monoscopic texture techniques to stereo.

Billboards are two dimensional (planar) surfaces. We have discovered that by preprocessing realistic monoscopic flame images and using several alpha blended billboards with images of flames pasted on them, a photorealistic stereo image of fire can be rendered in real-time.

4. BILLBOARDING

We describe our method using video capture, image processing, texturing and alpha blended billboards to generate stereo images of photorealistic fire.

4.1 Video Capture

Our method utilizes actual flames captured by a digital camcorder to produce photorealistic results. Filming was done at night with no artificial lighting so that the only color values in the final flame textures were produced from the flames. Care must be taken to ensure that the correct color values are being recorded by adjusting settings on the camera. The goal is to record so that an acceptable animated texture can be cropped from the resulting images. Care is required to avoid the appearance of over-exposure. One must ensure that the extents of the fire have been captured and that the camera is steady to simplify cropping during the editing process.

One must also avoid high intensity yellows or whites in the image, as color intensities are *additive* when using alpha-blending. We combine semi-transparent images to produce a flame with depth and the transparency causes intensity accumulation that can make the image too bright.

Images were captured at 30 frames per second. At this speed, 30 flame textures are sufficient to produce photorealistic real-time stereo fire. Camera settings were on default, point and shoot. The video was then imported into an iMovie project where individual frames were extracted and exported to images. We exported frames to images and saved them preserving the order that they appeared in the video. This produced frames similar to that shown in Figure 2. The next process is to crop and edit.

Figure 2. A frame taken from video footage of fire.

4.2 Image Editing

We used Macromedia Fireworks to crop to eliminate the fuel source so the fire could be ported to arbitrary scenes. This produces the image in Figure 3.

Figure 3. Image cropped from video frame (Figure 2).

Cropping severs the flame base from the fuel source and can result in apparent jumping and discontinuous movement. We used the line tool in Fireworks to add a line of black along the bottom of the images as seen in Figure 4. We then softened the lower edges of each flame image to minimize the jumping appearance by applying the smudge tool in Fireworks to blend the flames with the black line. See Figure 5. This altered the lower portion so the images of the fire so they are not completely flat.

Figure 4. Cropped image with added black line.

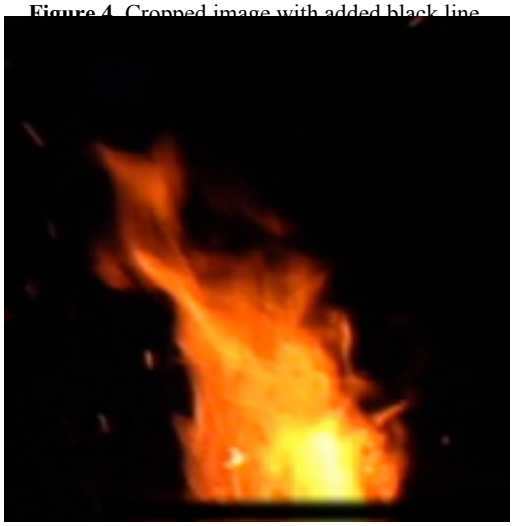

Figure 5. Cropped image with black line and softened base.

Cropping can also affects the acceptable range of choosing aspect ratios when mapping the texture to billboards during the animation. We discuss this in more detail later. The final step was to save all images in the same dimensions. We used gif images that measured 252 by 252 pixels.

4.3 Alpha blending

Using a linear alpha blending resulted in non-photorealistic flames that appeared flat and had abnormally large and unbroken yellow cores. See Figure 6.

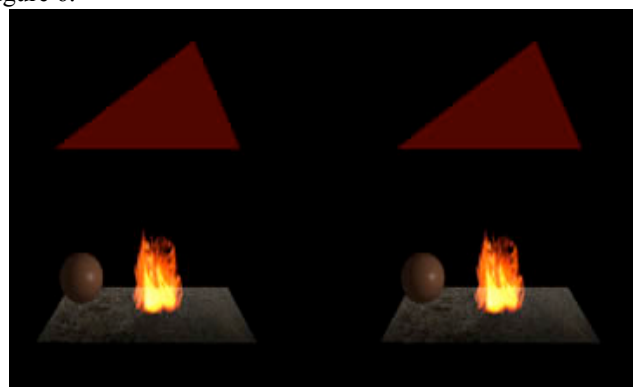

Figure 6. Our implementation using linear blending.

To overcome this effect of linear alpha blending, we applied a slightly more complex form of blending. We incorporate all colors and alpha values for every pixel of every billboard as well as blending factors for the original value and the new value to be blended, one each. We make use of OpenGL's glBlendFunc to get the desired appearance. Woo[12] presents a clear description of OpenGL's blending capabilities. The colors and alpha value of each texture are added to each other after the blending factors have been applied. Computations are made using textures in a back to front order using two sets of values (one from each texture) at a time, a source and destination. The destination values are those currently held by the pixel. The source values are those to be used in conjunction with the destination values to compute the new values for the pixel. Before this blending is applied, the color and alpha values are clamped to 1, meaning that they are forced into representing the percentage of possible intensity for their respective values. The source blending factor used by our method to modify the source values before summation is the destination value itself

subtracted from 1. The destination blending factor is simply 1. So if the source red value to be applied to the pixel were S_r, the destination red value were D_r, the resulting value for red would be $S_r (1-D_r) + D_r$. This process is continued on for every color value as well as the alpha value. Intuitively, we needed blending factors that would handle the transparency, while making it less likely that the color intensities reach their maximum. To ensure the transparency effect, the factors must not cause source or destination values to be replaced; rather they should allow both to contribute.

4.4 Animation

We are given texture frames 1- n in the video sequence that have been prepared as an animation sequence. In our example, n = 34. To obtain volume in the fire we partitioned the fire into several billboards, interleaved as shown in Figure 7. Each billboard will project a loop constructed from the flame animation captured above. Single texture frames are mapped to each billboard. Placing overlapping textures on the same plane results in artifacts in the form of pixel discoloration.

There are tradeoffs between the number and spacing of billboards. Too many billboards close together will produce a "blobby" appearance (a narrow histogram or a small variety of colors) while too few billboards will show as separate animations vs. a single continuous fire. For the example included here we used 6 billboards, arranged in 3 overlapped pairs $k = 1, 2, 3$ and having initial frame values as shown in Figure 7.

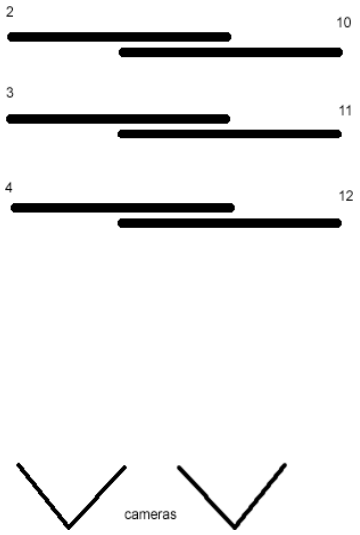

Figure 7. Billboard positions and initial frame indices.

Arranging the fire textures in pairs this way has multiple advantages. Overlapping helps give the illusion of large variation in the flames of a fire. The amount of overlap can be used to control the size of the "core" of the fire and resulting color.

In our example, Figure 7, there are 0.1 OpenGL units between the closer pair of textures. The portion of overlap for each pair in our example is 1/3. The pairs are spaced to enhance the illusion of volume in the fire. The distance between the texture pairs is 0.19 OpenGL units.

The selection of frame sequences for each billboard has a major effect on the character of the animation and contributes to the 3D effect of the fire. The goal is to create an animation of fire that has depth and coherence without distracting patterns. Sequencing the animation frames to obtain realistic results required considerable experimentation. Random sequences of frames were tried but did not contain sufficient consistency to appear realistic. We found that constructing subsequences of frames from the original video loop produced the best results. Different billboards are assigned

different initial frames and different step sizes or frame increments between frames. Frame increments, *m, n*, were varied for each billboard to enhance variation in fire behavior as described below.

We first choose two initial frame numbers for the furthest pair of billboards, e.g., for $k = 1$ the initial frames are 2 and 10 in Figure 7. The pair of billboards next closest to the viewer are then assigned succeeding frame values, e.g. for $k = 2$, initial frames are $2+1 = 3$ and $10 + 1 = 11$, and so on. We then choose increments (positive or negative) m_k, n_k for each billboard pair to be used to step through the frames in the animation sequence. For example, the frame sequence presented on the left billboard for $k = 1$ is $2, 2 + m_1, 2 + 2m_1, 2 + 3m_1, \ldots$ and we continue this process in a loop.

We tried incrementing the associated sequential texture index of every billboard by one (m_k, $n_k = 1$) in every time step. This, however, resulted in a fire that appeared to be moving away from the viewer. We then tried setting the frame increment for each billboard in the left column to 1 ($m_k = 1$) and the frame increment for the right billboard in the associated pair to –1 ($n_k = -1$). The result was an animation that moved in a clockwise circular direction about an axis in the center of the fire perpendicular to the ground. To remedy this, while maintaining realistic and interesting motion, we change n_k to –2 ($n_k = -2$) for each of the billboards on the right. We describe below how to extend our approach to grouping of billboards in groups of three vs. pairs. In the following section we discuss the effect of modifying billboard aspect ratios to change the appearance of a fire.

4.5 Aspect ratio manipulation and additive scaling

We experimented with different billboard aspect ratios to change the look and feel of the fire. 1:1.5 was the ideal aspect ratio for the billboards in our example presented here in Figure 8. Photorealism began to diminish when the aspect ratio exceeded 1:1 (Figure 9) or went below 1:2.5 (Figure 10).

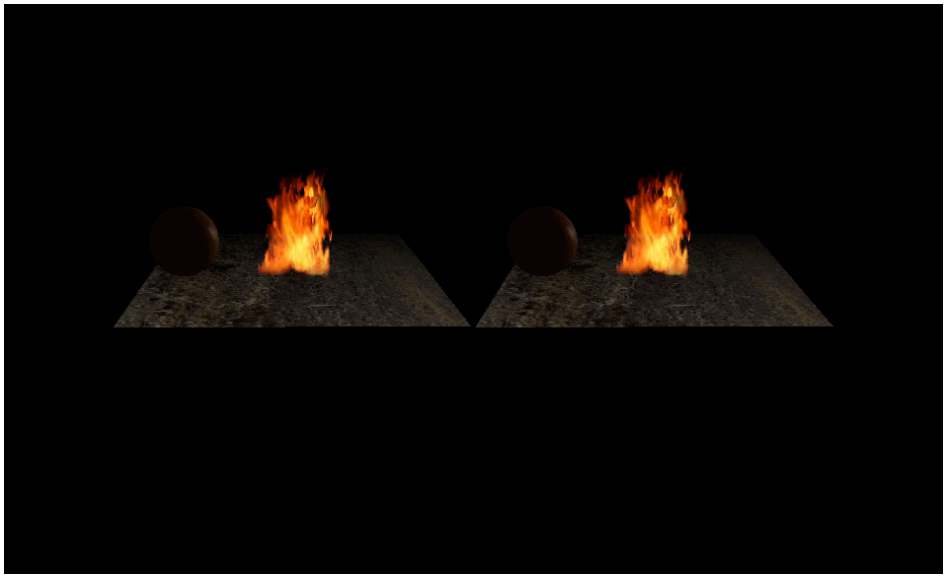

Figure 8. Single frame of billboards with a 1:1.5 aspect ratio for cross viewing.

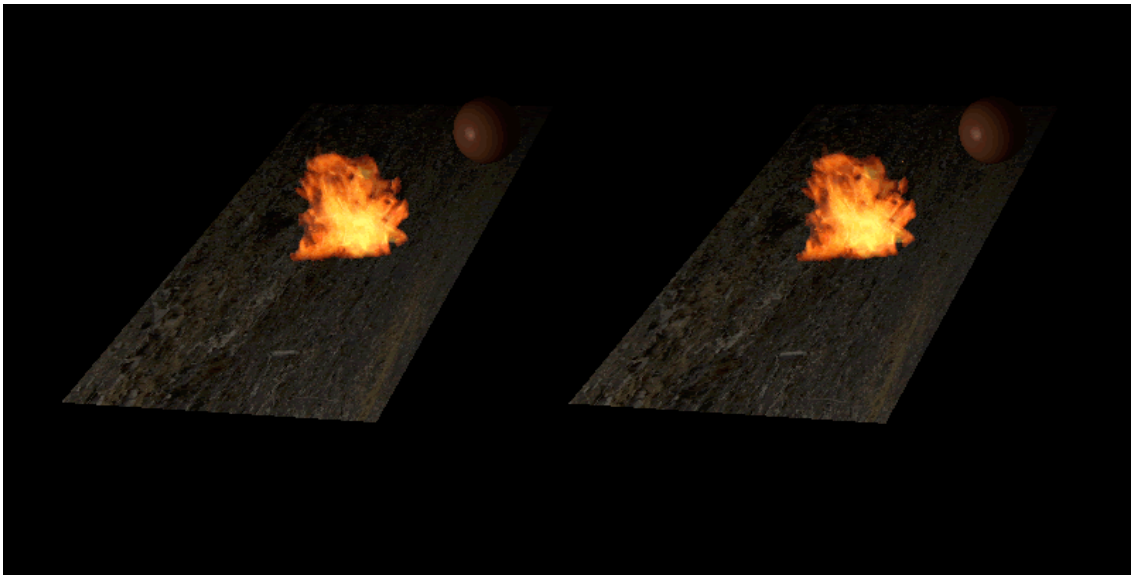

Figure 9. Single frame of billboards with a 1:1 aspect ratio for cross viewing.

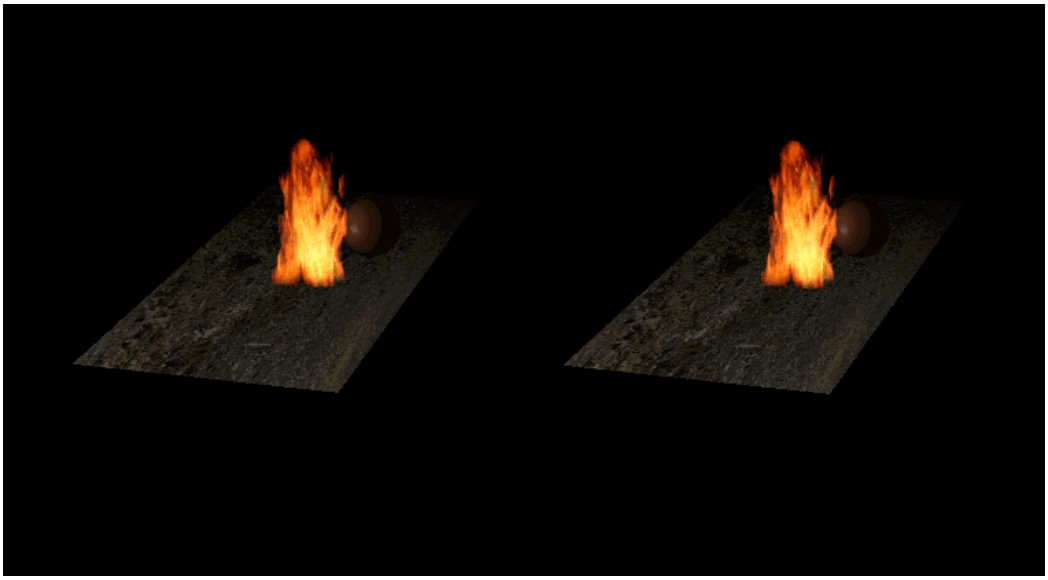

Figure 10. Single frame of billboards with a 1:2.5 aspect ratio for cross viewing.

While modifying aspect ratio can be used to scale a fire, one can also add new billboards, a method we call *additive scaling*. To create wider fires using the same textures without hyper-extending the billboard's aspect ratios for our example, we add more juxtaposed billboards and group them with existing billboards into triplets as shown in Figure 11. Juxtaposed billboards are in the same plane and the distance between their plane and the billboard behind is still 0.1 OpenGL units. The distance separating triplet groups in this example is the same as previous examples. Using the same number of flame textures as our other examples (n = 34), we create a completely different animation of stereo photorealistic fire images (Figure 12). We allow *l*, *m*, and *n* to denote frame increments in this example representing the leftmost, middle, and rightmost columns of billboards respectively. We use the same values for m_k and n_k ($m_k = 1$, $n_k = -2$) while setting the value for l_k as -2 ($l_k = -2$).

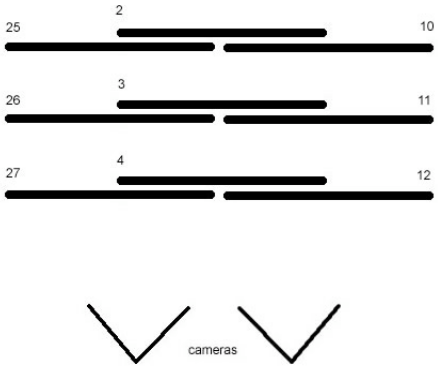

Figure 11. Widening a fire base by adding juxtaposed billboards.

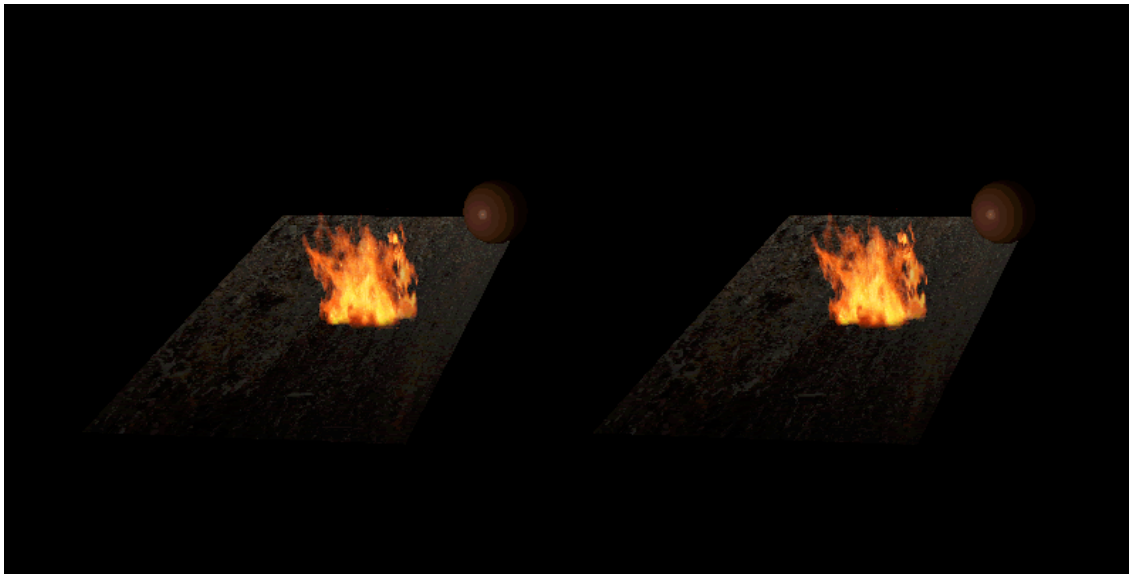

Figure 12. A single frame from the animation resulting from additive scaling.

4.6 Billboard movement

As the viewer moves in a scene, the billboards must also move so that their normals are always pointed towards the viewer. Billboards are being rotated as a group around an axis centered on and orthogonal to the base of the fire (Figure 13).

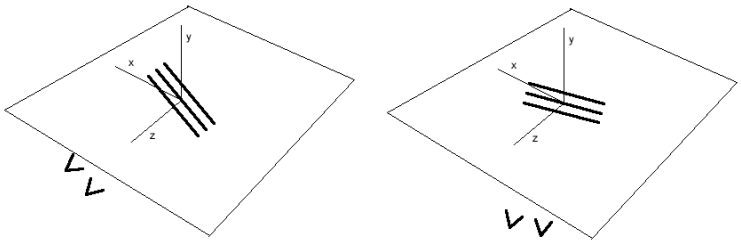

Figure 13. Billboard motion in response to viewer change in position.

Fire is unique in that the flames move and change so quickly that a viewer in motion is never aware of the group motion of the billboards. Billboards are restricted to be perpendicular to the base of the fire. Our method does not require the base of the fire to lie flat on the ground or surface. Viewer motion must be somewhat restricted; if the viewer looks down on the fire, the separate billboards used in the representation will be evident.

5. CONCLUSIONS AND FURTHER RESEARCH

We have presented a viable solution for rendering real-time photo-realistic fire in stereo. The solution stresses ease of image gathering and implementation. While the method requires preparation of texture images from each selected frame of a video at the beginning of the process, it allows for very rapid rendering using commercial graphics hardware. The example fire scene presented here in all its variations, using 34 flame textures stored in memory, executed at 60 fps on a 1.73GHz machine running XP Service Pack 2 with 512 MB of RAM. The graphics card was an NVIDIA GeForce Go 6200. The method can be easily extended to treat multiple fires burning in parallel and to simulate fire propagation. Simulation of a fire covering a large area will require study of how to randomize initial frames and frame step sizes to avoid the forming of recognizable cyclical patterns.

It may also be possible to combine our method with a fluid flow model with the use of careful image morphing. Other issues to be studied include the addition of smoke, audio, realistic illumination of objects by the fire, arbitrary viewing angles and applying transformations to the billboards to simulate external forces such as wind.

REFERENCES

1. A. Lamorlette, and N. Foster, "Structural Modeling of Flames for a Production Environment", *SIGGRAPH 29th annual conference proceedings*, vol. 21, issue 3, pp. 729-735, ACM Press, New York, NY, 2002.
2. D. Nguyen, R. Fedkiw, and H. Jensen, "Physically Based Modeling and Animation of Fire", *SIGGRAPH 29th annual conference proceedings,* vol. 21, issue 3, pp. 721–728, ACM Press, New York, NY, 2002.
3. K. Perlin, "An Image Synthesizer", *SIGGRAPH 12th annual conference proceedings* , vol. 19, issue 3, pp. 287–296, ACM Press, New York, NY, 1985.
4. H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels: Surface Elements as Rendering Primitives*", SIGGRAPH 27th annual conference proceedings*, pages 335–342, ACM Press / Addison-Wesley, 2000.
5. W. Reeves, "Particle Systems, - A technique for modeling a class of fuzzy objects", *ACM Transactions on Graphics,* vol. 2, issue 2, ACM Press, New York, NY,1983.
6. J. Stam, "Real-time Fluid Dynamics for Games", *Game Developers Conference Proceedings, 2003.*
7. J. Stam, "Interacting with Smoke and Fire in Real-Time", *Communications of the ACM*, Volume 43, Issue 7, 76-83, ACM Press, New York, NY, 2000.
8. M. Vesterlund, "Simulation and Rendering of a Viscous Fluid using Smoothed Particle Hydrodynamics", 2004.
9. M. Woo, et al. *OpenGL Programming Guide, 2nd ed.,* Addison-Wesley, Boston, MA, 1997.
10. N. Adabala and C. E. Hughes, "Gridless Controllable Fire", *Game Programming Gems 5,* K. Pallister (Ed.), pp. 539-549, Charles River Media, Hingham, MA, 2005.
11. P. Beaudoin, S. Paquet, and P. Poulin, "Realistic and Controllable Fire Simulation", *Graphics Interface 2001*, pp.159–166, Canadian Information Processing Society, Ottawa, Ontario, Canada, 2001.
12. PARAMOUNT, *Star Trek II: The Wrath of Khan* (film), June 1982.