Realistic and Controllable Fire Simulation

Philippe Beaudoin[a,b] Sébastien Paquet[a] Pierre Poulin[a]

[a]LIGUM
Département d'informatique et de recherche opérationnelle
Université de Montréal, Canada

[b]Matrox Graphics
Dorval, Canada

Abstract

We introduce a set of techniques that are used together to produce realistic-looking animations of burning objects. These include a new method for simulating spreading on polygonal meshes. A key component of our approach consists in using individual flames as primitives to animate and render the fire. This simplification enables rapid computation and gives more intuitive control over the simulation without compromising realism. It also scales well, making it possible to animate phenomena ranging from simple candle-like flames to complex, widespread fires.

Key words: Fire, simulation, natural phenomena, implicit surfaces, volume rendering, propagation on surfaces.

1 Introduction

Producing and controlling real natural phenomena for special effects realization can be challenging, costly, and sometimes dangerous, if not impossible. Computer simulations of such effects represent an interesting alternative as it can reduce or eliminate some of these obstacles. Fire being a very common natural phenomenon, its simulation can be expected to find uses in several contexts.

In this paper, we introduce a set of techniques that are used together to produce realistic-looking animations of burning objects. In addition to exhibiting convincing visual behavior, these techniques are efficient and provide easier control over the appearance of fire than previous simulation methods. This improvement is mainly due to our representation of fire which uses a relatively small set of flames instead of dealing with large numbers of particles.

1.1 Overview

Our work is motivated by the following two observations:

- Approaches to modeling fire that are based on differential equations can result in accurate animation [10], but to achieve this, they require the tracking of a large number of variables. In addition, the simulations they produce are difficult to control because their parameters are physical constants whose relationship to the desired visual effect is usually hard to figure out. Moreover, as the physics of fire is not fully understood [2], the values of many parameters have to be guessed.
- Most fire models [7, 8, 10, 11] are based on particle systems. A limitation of particle-based fire models is related to rendering. Although particles sets succeed in picturing the fuzzy, bumpy boundaries of clouds, they fail to show the characteristically crisp, sleek outlines of actual flames (unless an extremely large number of particles is used).

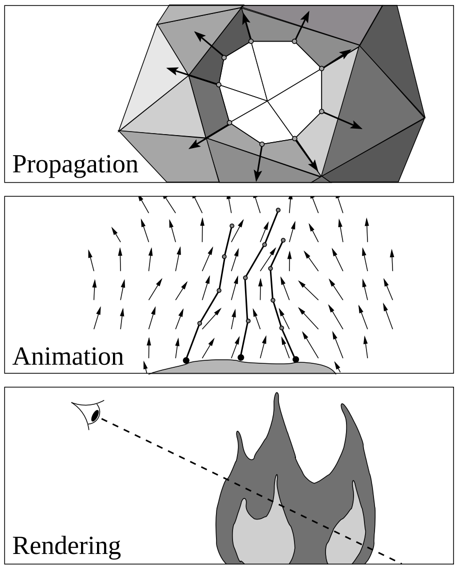


Figure 1: Overview of our fire model.

Our answer to these issues is a model that simplifies the physics involved in fire while retaining control over its important visual features. A key novel idea in our model is to represent fire as a set of *flames*, which are essentially deformable chains of vertices rooted on the surface. This representation captures well the behavior of simple flames while scaling up easily to picture intricate, turbulent fires. Another contribution of this work is the introduction of a new fire spreading algorithm that increases user control while maintaining realism.

Figure 1 outlines our approach. We divide the problem of simulating fire into the following three subproblems: fire propagation, flame animation, and rendering. *Propagation* refers to the spreading of fire over time, eventually engulfing the entire object. It is done by tracking the boundary between burning and not burning regions. *Flame genesis and animation* consist in placing flames on a surface and deforming them according to a space and time-dependent vector field in order to capture the visual dynamics of fire. Finally, *rendering* is achieved by defining implicit surfaces around these chains and displaying them using a volumetric technique.

This approach improves on previous methods for simulating fire in several respects. First, it produces more convincing images of isolated flames as well as burning objects. Second, the relative simplicity of our model results in reduced time and memory requirements compared to other approaches. Last but not least, our simplifications make the simulation more controllable, which means it is easier and takes less time to obtain desired effects.

The rest of the paper is organized as follows. In Section 1.2, we discuss previous work in the area of fire simulation. Section 2 describes our technique for evolving a fire front on the surface of an object. The following two sections explain how to create and animate flames and detail our method for rendering fire. Finally, we present and discuss our results, summarize the achievements and limitations of our techniques, and give directions for further investigations.

1.2 Related Work

A number of publications have been devoted to computer simulation of fire. Among the early models, Reeves [8] introduces particle systems as a modeling, animating, and rendering primitive. It allows the simulation of several natural phenomena, including the spreading and rendering of fire. However, this approach produces effects more akin to a series of small explosions than to genuine flames. Perlin [6] models fire using a noise function. The turbulent motion is incorporated by means of a fractal perturbation. This allows for somewhat realistic two-dimensional fire, but constrains the viewpoint to a single position and does not allow the fire to propagate. Inakage [4] uses a physical model of light emission and transmission around a combustion focus. The volume rendering techniques he uses yield convincing images of a single flame. However this work does not deal with multiple, animated flames.

Chiba *et al.* [1] and Takahashi *et al.* [11] describe methods for propagating, animating, and rendering fire. They divide space into a number of cells, each of which has an associated temperature and contains a certain quantity of fuel. Cell temperatures are evolved through a simulation where each cell receives heat from its neighbors. A cell ignites when its temperature exceeds a certain threshold. Fire itself is modeled using a system of independent particles which are influenced by a vector field. Images are produced by rendering a 3D primitive around each particle trajectory over a time step. Unless very large numbers of particles are used, this method results in bumpy flame contours.

Perry and Picard [7] simulate the evolution of a fire front on a polygonal mesh starting from an initial ignition point. The front is represented by a set of particles, and new particles are added to it as it expands. However the technique they describe does not give any guarantee that these new particles will lie on the mesh, which is an important requirement for properly depicting fire propagation. The particles that make up the front deposit fire sources along their path. These sources emit fire particles that can be affected by a wind field. Rendering is done using viewer-facing Gouraud-shaded hexagons whose aspect ratio depends on the fire particle's age. This produces flames with a generally blurry outline.

Stam and Fiume [10] cover the flammable object with a map that associates a fuel density, a temperature, and a lit/unlit flag with each point on its surface. A finite difference numerical method is then applied to evolve the system in time. This approach is somewhat costly and does not offer very flexible control to the user. Fire animation is done with turbulent wind fields [9]. The authors warn that their model "has a large set of interdependent parameters which are not necessarily easy to manipulate". Rendering is based on the *warped blobs* technique, which defines blobs around particles and casts rays along warped trajectories, computing a line integral along the way to obtain the color. Similarly to the previous method, the lack of well-defined contours prevents this technique from producing believable individual flames.

2 Fire Propagation

As was mentioned previously, it is possible to model fire propagation by numerically solving the differential equations that govern the evolution of temperature, pressure, and velocity of the air surrounding the burning object. We avoid the cost and complication of such simulation by observing that the main visual feature in fire propagation is the growth of the burning zone. This expansion is mainly driven by a few locally defined parameters: fuel density, oxygen supply, wind, and surface orientation relative to gravity. By tracking the boundaries between the parts of the object that are burning and those which have not yet been reached by the fire, it is possible to capture the essential characteristics of fire propagation at a relatively low computational cost. Our method for simulating fire propagation is inspired by that presented by Perry and Picard [7]. We improve on this work by ensuring that computed boundaries will always be continuous and lie on the burning object's surface, which is a necessary condition for the rest of the method to work properly.

2.1 Representing the Boundaries

A boundary is represented by a closed curve on the surface of the object. In this work the object is a closed triangular mesh and we use a broken line to represent the boundary, as Figure 2 illustrates. This line will always satisfy the following property:

A1. If two consecutive vertices are on different faces, then one of these vertices lies on an edge shared by those faces.

This prevents the broken line from leaving the surface.

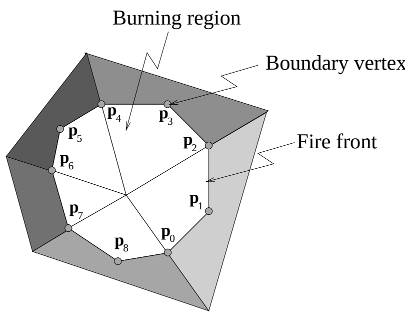


Figure 2: Broken-line representation of the fire front.

Boundary vertices have two vector-valued attributes: position and velocity. The positions of vertices in the initial boundary coincide with the ignition point and their velocities are spread uniformly (in proper order) along a circle that lies within the plane of the initial face.

2.2 Displacing Vertices

In what follows, each boundary vertex will move on the triangular mesh according to its velocity. As long as the vertex remains on a face, this is not complicated to deal with. Denoting its position by $\mathbf{p_i}$ and its velocity by $\mathbf{v_i}$, we have $\mathbf{p}_i(t+\Delta t) = \mathbf{p}_i(t)+\mathbf{v}_i(t)\Delta t$. However a vertex may leave a face by crossing an edge, as in Figure 3; it is thus necessary to specify what its velocity will become when it travels onto the other face.

In this situation, we define the new velocity $\mathbf{v}'_i$ using

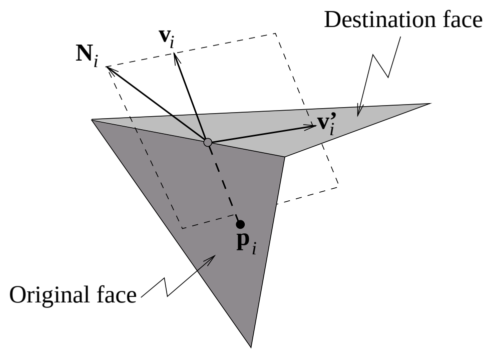


Figure 3: Transforming a velocity using Equation 1.

the following equation:

$$\mathbf{v}'_i = \eta(\mathbf{N}_i \times \mathbf{v}_i) \times \mathbf{N}'. \qquad (1)$$

In this equation, $\mathbf{v}_i$ is the original velocity, $\mathbf{N}_i$ is the linearly interpolated normal at the crossing point, and $\mathbf{N}'$ is the vector normal to the destination face. Coefficient η is chosen to preserve the magnitude of the velocity. This equation has a nice geometrical interpretation: vector $\mathbf{v}'_i$ is parallel to the intersection of the plane containing $\mathbf{v}_i$ and $\mathbf{N}_i$ with the plane of the destination face, as shown in Figure 3.

When used with closed manifold meshes with well-defined interpolated normals[1], Equation 1 has three interesting properties:

B1. The velocity does not change when the origin and destination faces are coplanar.

B2. Vector $\mathbf{v}'_i$ is nonzero and oriented away from the crossed edge.

B3. The continuity of a curve is preserved, i.e., no two neighboring points can be separated, even if they leave a face through different edges.

Property **B1** lets the propagation process behave correctly for an arbitrarily tesselated planar surface. Property **B2** ensures that no point of the front will get stuck near an edge, forever crossing it in successively opposite directions. Property **B3** , which was not enforced by the technique of Perry and Picard [7], is needed in the next subsection to enforce property **A1** of the front.

2.3 Evolving the Front

We wish to let the burning zone grow over time. This means that its boundary must expand with each time step.

[1] We say that an interpolated normal is *well-defined* if it has a positive dot product with its associated face normal(s).

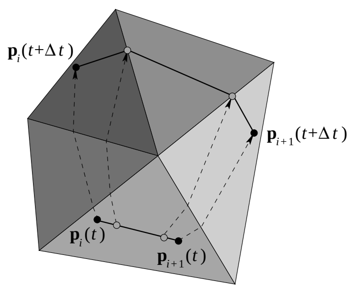


Figure 4: Enforcing property **A1**.

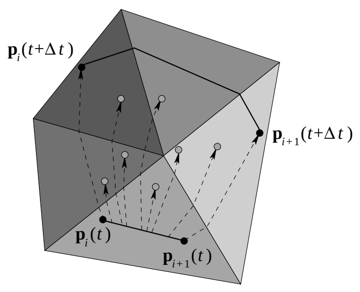


Figure 5: Sampling points on the surface.

This is achieved by the following two-step update procedure. First we let each vertex of the boundary move according to its velocity vector, updating its velocity along the way whenever an edge is crossed. Second, we ensure that the final boundary enforces property **A1** by introducing new vertices on edges where needed.

To perform this last step, we first identify two consecutive vertices which are no longer on the same face (if there are any). Using property **B3**, we know that the line segment which joined these two vertices at the previous time-step will be displaced into a new continuous curve. This curve crosses the edges where new vertices must be added (see Figure 4). To compute the intersection between this curve and the edges, we find which points on the original line are displaced directly onto an edge. These points are found efficiently using a binary search along the original line segment. Our boundary evolution algorithm may eventually produce a very sparse or very dense distribution of vertices along the front. To avoid this, we include another step where new points are inserted on boundary segments when two consecutive vertices get too far apart. As well, if it happens that vertices come too close together on the same triangle, we simply remove one of them.

2.4 Nonuniform Propagation Speeds

As was noted earlier, the evolution of a burning surface is driven by locally defined parameters. To account for these parameters during front propagation, we can alter the vertex velocities at the beginning of each time step. To make sure the front never shrinks, we only allow modification to the magnitude of these velocities. This magnitude can be computed as a function of the velocity direction, gravity, fuel density, and any other surface-defined parameter. Using this, for example, a front would propagate faster along an upward direction.

2.5 Generating Points on the Surface

The propagation technique explained above makes it possible to generate points on the surface enclosed by the front. Such points will appear at each time step inside the area newly swept by the front. For a given segment of the front, these points are generated by evolving a random point on the segment during a randomly chosen time between 0 and Δt. Figure 5 illustrates this process.

3 Flame Genesis and Animation

As was stated earlier, we represent fire as a set of flames. We picture a flame as a stream of incandescent gas which follows the air flow surrounding it. In our implementation, this stream is modeled as a chain of connected particles, which we call the skeleton. The first particle of the skeleton is the root of the flame and is attached to a point on the burning surface. The rest of the chain moves according to a turbulent, time-varying vector field which accounts for the dynamic behavior of the fire. This field is defined by the user and is meant to mimic the convection air flow which occurs when combustion gives off heat.

3.1 Planting Flames on the Surface

The first step needed to perform the animation is to place flames on the surface. To do this, we use the point generation technique of Section 2.5. Each of these points becomes the root of a new flame. The density of points generated on the surface can be adjusted to capture various effects visible in fire. Increasing the number of points generated within a given area will result in a more intense fire.

3.2 Defining the Air Velocity Field

In our model each flame skeleton's configuration at a time t will depend on the air velocity field $\mathbf{V}(\mathbf{x},t)$ at that moment. The function $\mathbf{V}(\mathbf{x},t)$ is designed to exhibit features specific to animated fire. It supports changes over space and time, allowing for local and temporary wind gusts, vortices, and other effects. To achieve this, we

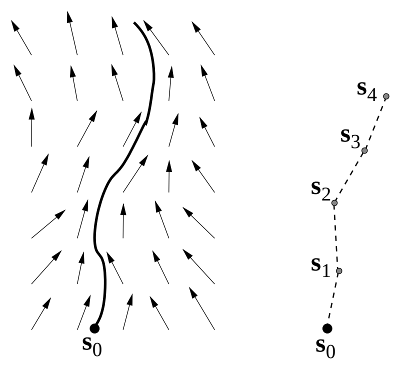


Figure 6: Flame skeleton in an air velocity vector field.

build $\mathbf{V}$ from various components:

$$\mathbf{V}(\mathbf{x},t) = \sum_{i=0}^{n} b_i(\mathbf{x},t)\,\mathbf{V}_i(\mathbf{x},t)\ . \tag{2}$$

In this equation, b_i is a blending function which defines the region of space and time over which component $\mathbf{V}_i$ will be effective. For better results, b_i should either be a constant or start from 0, smoothly evolve to 1 and return to 0 for each of its four parameters.

The $\mathbf{V}_i$ functions are based on user input and capture characteristics of an animated flame: low frequency wind blows, fast horizontal oscillation, vertical flickering, etc. Most importantly, a strong and steady component opposite to gravity is included to account for heat-induced convection. Noise functions [6] or predefined vector functions [3] can be useful in defining $\mathbf{V}_i$.

3.3 Defining the Flame Skeleton

Once the root of a flame has been placed at a point $\mathbf{s}_0$ on a surface, we create the skeleton which serves as a basis for the flame shape. This skeleton approximates a curve segment which begins at the root and is tangent at every point to the instantaneous velocity field $\mathbf{V}(\mathbf{x})$. In other words, this curve is the solution to the differential equation $\frac{d\mathbf{r}}{du} = \mathbf{V}(\mathbf{r})$, where u is the curve parameter, with the initial condition $\mathbf{r}(0) = \mathbf{s}_0$.

The skeleton is a broken line approximation of function $\mathbf{r}$ (see Figure 6). If the $n+1$ vertices of the skeleton are denoted $\mathbf{s}_0, \dots, \mathbf{s}_n$ we define $\mathbf{s}_i = \mathbf{r}\big(l(t)\,i/n\big)$ where $l(t)$ is a factor which affects the total length of the skeleton and varies over the flame's lifetime as we describe further on (in Section 3.4).

In the general case, the differential equation cannot be solved analytically. We therefore use a simple Euler integration scheme. In practice, the precision of $\mathbf{r}(u)$ is not critical and we can use large integration steps without affecting the visual results. For this reason, we use as many steps as there are segments in the skeleton:

$$\mathbf{s}_i = \mathbf{s}_{i-1} + \frac{l(t)}{n}\,\mathbf{V}(\mathbf{s}_{i-1})\ . \tag{3}$$

This technique yields good results even with a small number of segments per skeleton. Results shown in this paper use $n = 4$.

The skeleton obtained by evaluating Equation 3 may penetrate the burning object, yielding undesirable results. This is due to the fact that the air velocity field is defined without reference to the geometry of the object and thus does not necessarily flow around it. To make up for this, we provisionally adjust the vector field by adding to it a component that is normal to the surface at the root. The magnitude of that component is chosen such that $\mathbf{s}_1$ is brought out of the object. When flames are small compared to the features of the burning object, this eliminates the majority of intersections between skeletons and objects; those which remain do not interfere significantly with the rendering. The end effect is typical of flames licking at an overhanging surface.

3.4 Growing and Shrinking the Flames

To allow for the eventual extinction of the fire, flames are assigned a life duration. Over its life span, the length factor $l(t)$ (Equation 3) of a skeleton can vary according to an arbitrary function. We use a clamped quadratic function of time starting and ending at zero, which prevents popping and results in believable ignition and extinction. The peak length of a flame is taken from a distribution centered around a user-defined mean length. This length and the flame's life duration can be made to depend on the point where the root was placed using per-vertex values or texture mapping.

3.5 Detached Flames

The technique described previously works well in the case of quiet fires, such as candle flames. However, highly turbulent fires often feature flames that take off from the surface and drift for a while before cooling down and vanishing. To incorporate this effect, we track detached flames whose root is free to move away from the surface. These flames act like the particles used in previous techniques and, as such, benefit from the same advantages.

A detached flame behaves exactly like a normal flame, with the difference that its root position $\mathbf{s}_0$ is updated at every time step. The root moves like an ordinary particle in the velocity field. At each time step the position $\mathbf{s}_0$, now a function of t, is updated using an Euler integration scheme:

$$\mathbf{s}_0(t+\Delta t) = \mathbf{s}_0(t) + \Delta t\,\mathbf{V}(\mathbf{s}_0(t),t)\ . \tag{4}$$

After the root has been moved, the rest of the skeleton is evaluated using Equation 3 where the length factor $l(t)$ varies over time in a manner similar to ordinary flames.

To decide whether to introduce a new detached flame, we evaluate $|\mathbf{V}|$ at each vertex of the skeleton. If at some point it exceeds a user-defined threshold then we insert a new detached flame, its length initially being set to zero. We then temporarily inhibit further detached flame emission for that skeleton. Other insertion mechanisms can be devised, leading to various visual effects like sudden flame bursts or brands.

4 Rendering and Modeling

Although the flame skeletons convey a lot of visual information and can be used effectively for previewing an animation (as shown in Figure 11), they are not sufficient for photorealistic rendering. To achieve higher quality rendering, it is necessary to "dress up" the skeletons. In actual fires, flames that are close enough smoothly blend together while distant flames remain separate. Modeling flames using implicit surfaces provides a convenient way to emulate this behavior.

We therefore begin by defining an implicit surface which describes the outline of a single flame. Next we adjust the function so that it behaves well in the context of a complete fire, where the contribution of many flames are summed up. We then model the color variations that are seen inside the fire. Finally, we render the final image using a ray tracing algorithm.

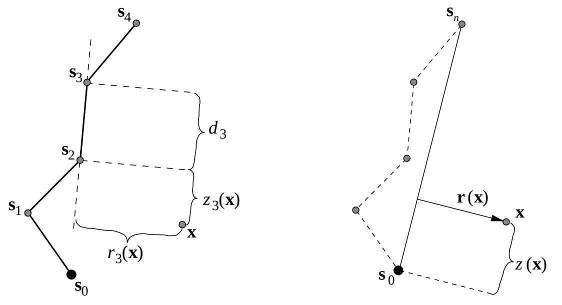
4.1 Basic Shape Equation

The function which we use for the outline of a single flame is initially defined using the physical analogy of the electrical potential induced by a uniformly charged rod. For the chain segment i joining vertices $\mathbf{s}_{i-1}$ and $\mathbf{s}_i$, this function is expressed as:

$$\begin{aligned} E_i(\mathbf{x}) &= \int_{p=0}^{d_i} \frac{1}{\sqrt{\left(p - z_i(\mathbf{x})\right)^2 + r_i(\mathbf{x})^2}} dp \\ &= \sinh^{-1}\left(\frac{z_i(\mathbf{x})}{r_i(\mathbf{x})}\right) - \sinh^{-1}\left(\frac{z_i(\mathbf{x}) - d_i}{r_i(\mathbf{x})}\right) , \end{aligned}$$

where $z_i(\mathbf{x})$ and $r_i(\mathbf{x})$ are cylindrical coordinates of point $\mathbf{x}$ relative to segment i (see Figure 7(a)) and where $d_i = \left|\mathbf{s}_i - \mathbf{s}_{i-1}\right|$ is the length of segment i. For a complete flame, we add the contributions from all the skeleton segments:

$$E(\mathbf{x}) = \sum_{i=1}^{n} E_i(\mathbf{x}) . \qquad (5)$$



(a) $z_i(\mathbf{x}), r_i(\mathbf{x})$. (b) $z(\mathbf{x}), \mathbf{r}(\mathbf{x})$

Figure 7: Cylindrical coordinate systems of skeletons.

4.2 Making the Shape Asymmetrical

The isosurface obtained using Equation 5 does not distinguish between the root and the top of the flame. However, real flames are bulged at the root and thin at the top. To produce this asymmetrical shape, we transform $\mathbf{x}$ in the previous equation according to a height-dependent radial contraction $\mathbf{x}'(\mathbf{x})$ which spans across the entire flame:

$$\mathbf{x}'(\mathbf{x}) = \mathbf{x} + \exp\left(\frac{2z(\mathbf{x})}{d} - 1\right) \mathbf{r}(\mathbf{x}) , \qquad (6)$$

where d is the length $\left|\mathbf{s}_n - \mathbf{s}_0\right|$. This function uses the cylindrical coordinate $z(\mathbf{x})$ relatively to the segment $(\mathbf{s}_0, \mathbf{s}_n)$. The vector $\mathbf{r}(\mathbf{x})$ joins that segment and the point $\mathbf{x}$ (see Figure 7(b)). When composed with E, the transformation ensures that the isosuface is tightened at the top while it remains bulgy at the root. This transformation was designed empirically so that isosurfaces of function $E\bigl(\mathbf{x}'(\mathbf{x})\bigr)$ would be suited for representing the shape of a single flame.

4.3 Defining a Complete Fire

Although the preceding function describes well a unique flame, it can create problems when trying to define a complete fire. Since function $E(\mathbf{x})$ falls off rather slowly with distance from the skeleton, two distant skeletons can influence each other's envelope. To remove these undesirable non-local effects while preserving the shape of a single flame, we amplify the falloff of E for values smaller than the user-defined isovalue v. The following empirically derived function takes care of that:

$$F(E) = \frac{v\ \left(e^E - 1\right)}{e^v - 1} . \qquad (7)$$

This function also has the advantage of reducing the bounding volume for a flame, diminishing the time

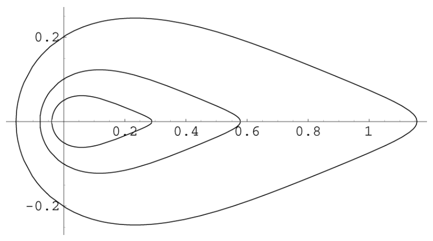


Figure 8: Isosurfaces with $v = 3$ for three straight skeletons with lengths 1, 0.5, and 0.25.

needed for rendering an image (see Section 4.5). The following function I_s expresses the final contribution of skeleton s to the implicit function at point $\mathbf{x}$:

$$I_s(\mathbf{x}) = F\Big(E\big(\mathbf{x}'(\mathbf{x})\big)\Big) \ . \tag{8}$$

Function I is the value for a complete fire:

$$I(\mathbf{x}) = \sum_s I_s(\mathbf{x}) \ . \tag{9}$$

4.4 Various Color Layers

Until now, we have focused on modeling the flames' shape. Because temperature rises towards the center of a flame, radiation emitted by its different parts features different wavelength distributions. For a single flame, various colors appear in successive layers, each having a shape similar to the flame outline. Temperature mainly depends on distance from the base of the flame.

To model this, we compute Equation 9 using progressively smaller flame skeletons for each layer. These skeletons are obtained by removing a fixed proportion at the top of each chain. If the same iso-value v is used with these new implicit functions, we obtain a sequence of surfaces where each one encloses the next. For a single flame, the result is shown in Figure 8. When rendering, we assign different colors to the layers in order to obtain a variety of fire effects.

4.5 Computing the Surface

We now describe how the surfaces defined above may be computed. Given Equation 9 for a particular layer and a user-defined iso-value v, we use the marching cubes technique [5] to obtain the closed surface satisfying $I(\mathbf{x}) = v$. This technique requires evaluating I at each point of a dense regular grid, a process which can be time consuming. For a cubic grid of side length a with m flame skeletons, the time required is proportional to a^3m.

To make the computation of I independent of the grid's volume, we limit the contribution of each skeleton to its neighborhood. To this end we start from the base of each flame and perform a flood-fill type traversal of the grid vertices, stopping when I_s takes on a value that is considered negligible (our implementation uses $v/100$).

4.6 Rendering the Fire

Each layer defined in Section 4.4 is a volume which emits light of a particular color and intensity. Therefore, the color and intensity of light reaching the eye following a given path is a function of the path lengths inside each layer.

We assume that light scattering occurring inside the fire is not significant and thus consider only straight light paths. The lengths of these paths inside each layer can be computed easily by casting rays toward the polygons obtained by the marching cubes algorithm. We therefore render the scene using a ray tracing technique which considers the color contribution from each layer of fire. A scanline rendering method processing polygons from back to front could also be devised to produce the same results more efficiently.

5 Results

We ran our simulation on a number of polygonal meshes. Figure 9 illustrates front propagation on a moderately complex cow model (5804 triangles). Boundary vertices originate from a point above the cow's leg and spread progressively over the model surface. Figure 11 shows the evolution of flame skeletons on a flammable sphere. Still images from fully rendered animations are presented in Figures 10 and 12. The complete animations can be found on our web site associated with this paper from *www.iro.umontreal.ca/labs/infographie/papers*.

The simulations ran on a 400 MHz Intel Celeron with 256 Mb of memory. For meshes with a few hundred triangles, our propagation technique runs at around 15 Hz. Skeleton animation with a hundred skeletons is done at a rate averaging 4 Hz. Rendering with the same number of skeletons is more time consuming: our program outputs 4 frames per minute. As was mentioned previously, this could possibly be improved by replacing ray tracing with a scanline rendering technique. The most important memory requirement of the method was polygon storage for rendering purposes, which remained below 20 Mb.

6 Discussion

Our aim in this work was to obtain a controllable, realistic-looking, and fast fire simulation method. To this end, we have introduced a new propagation technique and have shown how flames could be used effectively as animation and rendering primitives. Our method allows for quick previewing of an animation and features intuitive parameters, which speeds up the iterative process toward obtaining the desired effects. Moreover, it enables the user to simulate simple as well as complex fires.

A limitation of our spreading model is that fire cannot reach objects that are disconnected from a burning object. This could be modeled by starting new propagation processes from points located above existing flames. Other directions for future work include taking into account the illumination of objects by the fire itself, actually destroying or altering the surface of burning objects, and incorporating smoke into the animations.

7 Acknowledgements

We acknowledge financial support from FCAR and NSERC.

8 References

[1] N. Chiba, S. Ohkawa, K. Muraoka, and M. Miura. Two-dimensional visual simulation of flames, smoke and the spread of fire. *The Journal of Visualization and Computer Animation*, 5(1):37–54, January–March 1994.

[2] J. Chomiak. *Combustion. A Study in Theory, Fact and Application.* Abacus Press/Gordon and Breach Science Publishers, New York, 1990.

[3] T. L. Hilton and P. K. Egbert. Vector fields: an interactive tool for animation, modeling and simulation with physically based 3D particle systems and soft objects. *Computer Graphics Forum*, 13(3):329–338, 1994.

[4] M. Inakage. A simple model of flames. In *Proceedings of Computer Graphics International 1989*, pages 71–81, 1989.

[5] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH 1987 Conference Proceedings*, volume 21, pages 163–169, July 1987.

[6] K. Perlin. An image synthesizer. In *SIGGRAPH 1985 Conference Proceedings*, volume 19, pages 287–296, July 1985.

[7] C. H. Perry and R. W. Picard. Synthesizing flames and their spreading. In *Fifth Eurographics Workshop on Animation and Simulation*, pages 105–117, September 1994.

[8] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[9] J. Stam and E. Fiume. Turbulent wind fields for gaseous phenomena. In *SIGGRAPH 1993 Conference Proceedings*, volume 27, pages 369–376, 1993.

[10] J. Stam and E. Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *SIGGRAPH 1995 Conference Proceedings*, pages 129–136, August 1995.

[11] J. Takahashi, H. Takahashi, and N. Chiba. Image synthesis of flickering scenes including simulated flames. *IEICE Transactions on Information Systems*, E80-D(11):1102–1108, 1997.

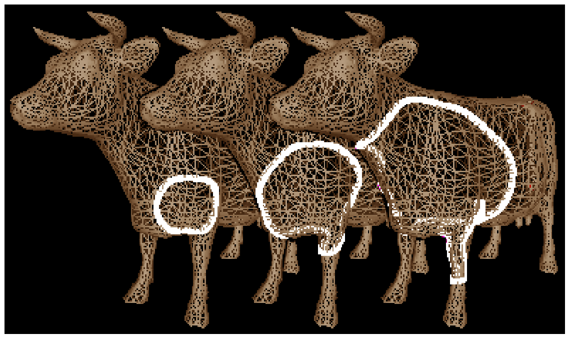


Figure 9: Front propagation.

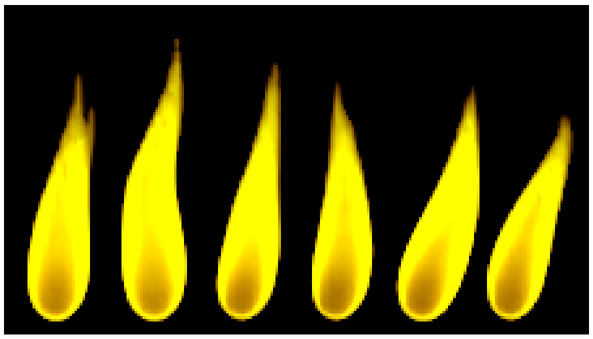


Figure 10: Animated candle flame.

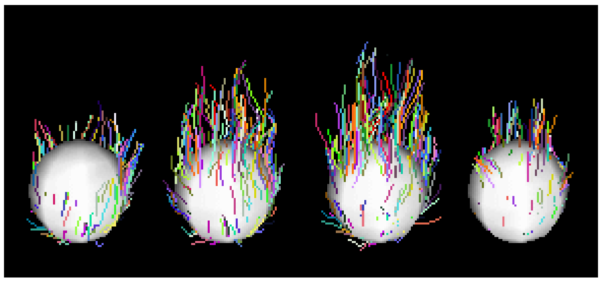


Figure 11: Flame skeletons on a burning sphere.

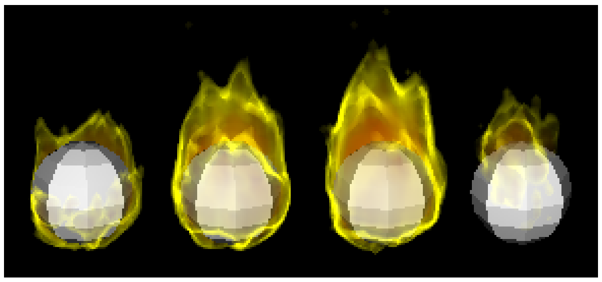


Figure 12: Burning sphere.