Meshes on Fire

Haeyoung Lee[†] Laehyun Kim[†] Mark Meyer[‡] Mathieu Desbrun[†]
[†]U. of So. Cal. - [‡]Caltech

Abstract.
We present a new method for the animation of fire on polyhedral surfaces. Using the notion of *discrete straightest geodesics*, we evolve fire fronts *directly on the surface* of arbitrarily complex objects. Animator control and motion complexity is achieved by driving the fire motion using multi-scale turbulent wind fields and geometric quantities. Our model also supports adaptivity of the fire fronts, multiple simultaneous fires, and merging of multiple fires. This new technique produces convincing simulations at interactive rates even on a low-end PC, greatly increasing the productivity of the animation design process.

1 Introduction

Fire is a common, yet mysterious entity used in widely varying areas such as entertainment and training. The physical use of fire can however be both costly and dangerous. Computer simulation of fire offers many benefits including increased control, reduced expense and danger, and reproducibility. However, most previous work has been focusing on fire rendering, while we propose to concentrate our effort on *fire propagation*.

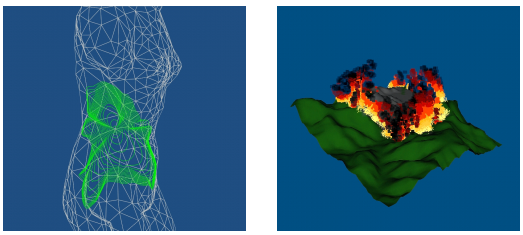
1.1 Previous Work

Fire simulation has been extensively researched in computer graphics [9, 10, 2, 6, 11, 4, 1]. The earliest computer graphics fire model was presented by Reeves [9]. The model used a large number of particles to animate a fire engulfing a planet. Although particles can easily represent fuzzy objects, representing well defined boundaries is more problematic and many particles are required to represent the fire.

Stam and Fiume [11] discretize the flammable object using a texture that represents fuel density, temperature, etc at every point on the object. A finite difference scheme is used to simulate the resulting fire. A turbulent wind field [10] is simulated and interacts with the fire to produce complex animations. The method is computationally expensive and requires a fine discretization to accurately represent the fire boundary.

Perry and Picard [6] and Beaudoin et al. [1] represent the boundary of the fire directly using several connected sample points. This results in far fewer particles being required to adequately animate the evolving fire boundary. Our model most closely resembles this work, with several important extensions including adaptivity, multiple simultaneous fires, and fire merging.

Our work also formalizes the use of geodesic flow to simulate the evolution of the fire boundary over a polyhedral object. Polthier et al. [7, 8] recently used geodesic flow to investigate surface properties. However, as our intended application area is different, several modifications were required to produce an appropriate geodesic flow algorithm, such as an appropriate handling of swallow tails created by conjugate vertices.



a b

Fig. 1. *(a) The* flame front *(in green) propagates on the surface according to the wind field and geodesic flow. (b) As the front progresses, flame particles [10] are dropped to visualize the fire.*

1.2 Overview

In this paper, we present a technique for modeling fires over polygonized objects (see figure 1). Our system represents a fire as an evolving front (also known as the *inception boundary*) and a system of particle-based flames. Since the front should only propagate on the object's surface, we use discrete straightest geodesics as defined in [7, 8] to evolve the fire front's motion directly on the surface itself. Using the formalism of geodesics we can guarantee that the front evolves correctly and always remains on any triangulated 2-manifold, regardless of its complexity.

To create rich, complex motion for the front, as well as to allow for general animation control, we drive the front motion using dynamic wind fields [10]. This multi scale technique allows the animator to describe large scale motions while a stochastic process creates the visually rich, small scale, turbulent motions.

As the front moves across the objects' surface it deposits particle-based *flames*. These flames evolve as standard fire particles [11] - flickering in the wind and even flying off the surface given a large enough wind field. These flames are rendered as blobs and leave a charred residue on the burnt surface.

Our system also incorporates advanced effects such as multiple, simultaneous flame fronts, merging of multiple fronts, fire ignition due to flying flame particles, and adaptivity for accurate fronts at minimal computational cost. Even with these complex features our system runs at interactive rates on a standard PC - extremely useful when designing a fire animation. Higher quality renderings [11, 1] can then be produced offline.

The remainder of the paper is organized as follows. The basics of geodesics for both continuous and discrete surfaces are described in detail in section 2. Our fire propagation model is introduced in section 3. Results are presented in section 4 followed by conclusions in section 5.

2 Front Propagation on Meshes

Under the assumption of no external factors (i.e., no wind field, uniform fuel density, etc.) a fire front on a flat surface would propagate out equally in all directions, creating concentric fronts as shown in figure 2(a). Each point on the front travels in a (euclidean) straight line. Therefore, we can determine the next front by simply advancing each point on the front along its respective propagation line. To extend this technique to curved surfaces, we must extend the notion of straight lines from the euclidean plane onto the surface. In this section, we review and extend a technique originally defined by Polthier and Schmies [7].

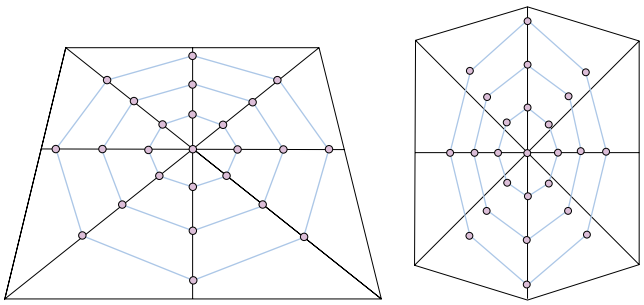


Fig. 2. *(a) Evolving a front on a flat plane. All points on the front move along euclidean straight lines. (b) Evolving a front on a polyhedral surface. All points on the front move along discrete straightest geodesics.*

2.1 Geodesics on Polyhedral Surfaces

Geodesic curves are the extension of euclidean straight lines to a surface and are defined for a smooth surface as:

Definition 1: Let S be a smooth 2-dimensional surface. A smooth curve $\gamma \subset S$ is a geodesic curve if any of the following equivalent properties hold:

- γ is a locally shortest curve
- γ is a straightest curve (has zero geodesic curvature)

Using definition 1, we can now solve for the evolution of a front moving on a smooth surface by simply moving all points on the front appropriately. Given a point $\mathbf{p}$ on the front and the direction (in the tangent plane of S) of front motion $\mathbf{d}(\mathbf{p})$ at $\mathbf{p}$, move $\mathbf{p}$ along the geodesic γ at $\mathbf{p}$ in the direction $\mathbf{d}(\mathbf{p})$.

Although definition 1 allows one to update a front on a smooth surface, we often need to work with discrete polyhedral meshes. While geodesic curves on smooth surfaces are both straightest and locally shortest, on polyhedral meshes these two concepts *differ*. In fact, shortest geodesics on polyhedral meshes are not necessarily unique while the straightest geodesic given a point and direction on the polyhedral mesh is uniquely defined. As our original motivation was to determine how to move straight in a given direction on a polyhedral mesh, we require only the definition of straightest geodesics on polyhedral meshes, as introduced in [7]:

Definition 2: Let M be a polyhedral 2-dimensional surface. A curve $\gamma \subset M$ is a straightest geodesic curve if for every point $\mathbf{p} \subset \gamma$ the left and right curve angles, θ_l and θ_r are equal - where θ_l and θ_r measure the angle to the left or right of the curve at $\mathbf{p}$ *within the surface*.

Definition 2 then allows one to advance a front on a polyhedral mesh by uniquely solving the following initial value problem:

Discrete straightest geodesic initial value problem: Given a polyhedral mesh M, a point $\mathbf{p} \subset M$, and a polyhedral tangent vector $\mathbf{d}$ at $\mathbf{p}$, there exists a unique straightest geodesic $\gamma \subset M$ solving the initial value problem:

$$\begin{aligned} \gamma(0) &= \mathbf{p} \\ \gamma'(0) &= \mathbf{d} \end{aligned}$$

where the set of all *polyhedral tangent vectors* at a point $\mathbf{p} \subset M$ is defined to be all vectors within the faces of M adjacent to $\mathbf{p}$.

Advancing a front on a polyhedral surface now amounts to advancing each point on the front along a unique straightest geodesic. This can be accomplished by a simple Euler integration. Given a point $\mathbf{p}(t_i)$, and a polyhedral tangent direction $\mathbf{d}(t_i)$ to follow, the next point $\mathbf{p}(t_i + \Delta t)$ is found as:

$$\mathbf{p}(t_i + \Delta t) = \mathbf{p}(t_i) + \mathbf{d}(t_i)\Delta t \tag{1}$$

For each pair of vertices $\mathbf{p}(t_i)$ and $\mathbf{p}(t_i + \Delta t)$ three cases must be considered (see figure 3):

- $\mathbf{p}(t_i)$ and $\mathbf{p}(t_i + \Delta t)$ lie on the same face - since the surface is locally flat, this case is trivial to handle. Tangent values for $\mathbf{p}(t_i)$ and $\mathbf{p}(t_i + \Delta t)$ are the same.
- $\mathbf{p}(t_i + \Delta t) - \mathbf{p}(t_i)$ first crosses an edge - using the intersection with the edge $\mathbf{q}$, we must rotate $\mathbf{d}(t_i)$ around the edge by the angle between the two normal vectors of two neighboring surfaces. Using this rotated vector as the new polyhedral tangent, we continue on towards $\mathbf{p}(t_i + \Delta t)$, possibly crossing more edges on the way.
- $\mathbf{p}(t_i + \Delta t) - \mathbf{p}(t_i)$ first crosses a vertex - we calculate the polyhedral tangent vector $\mathbf{d}(\mathbf{q})$ at the intersected point $\mathbf{q}$ such that the resulting curve through $\mathbf{p}(t_i)$, $\mathbf{q}$, and $\mathbf{q} + \mathbf{d}(\mathbf{q})$ will have equal left and right curve angles. Using this new tangent, we continue on from $\mathbf{q}$.

Every single particle update will be a succession of any of these three simple cases. Propagation on arbitrary meshes, even with a very irregular connectivity, is therefore handled robustly.

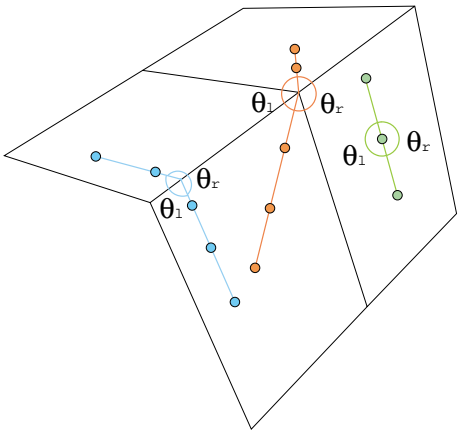


Fig. 3. *Three cases for integrating straightest geodesics: (green) both points on the same face, (blue) path between points intersects an edge, (red) path between points intersects a vertex. In each case, the two angles* θ_r *and* θ_l *are equal.*

Notice that evolving the front using discrete straightest geodesics guarantees a unique solution to the problem. This method correctly handles even the case when a point on the front evolves directly through a vertex. There is however a problem when the front passes through a hyperbolic point (a point where gaussian curvatue is negative) or a conjugate point (a point where the gaussian curvature is positive): a swallow tail effect

appears (see Figures 5 and 9(a)), after a conjugate point, creating a front interference. Our fire propagation model modifies the initial approach [7] to correctly deal with this case, as described next.

2.2 Continuous Fire Front

Although the original work [7] was designed to handle swallow tails, our fire propagation method should not contain these features. Since the fire burns the surface as it moves, front interferences (corresponding to double burning) should not occur. To remedy this problem, we first describe how we can keep the front ordered and uniformly sampled.

Adaptive, Ordered Front. To represent the continuous fire front, we maintain an ordered list of the front particle emitters allowing each particle access to its immediate neighbors. Often, as the front progresses, our original sampling becomes inadequate — especially after a hyperbolic vertex with negative gauss curvature, neighboring flame particles will diverge excessively.

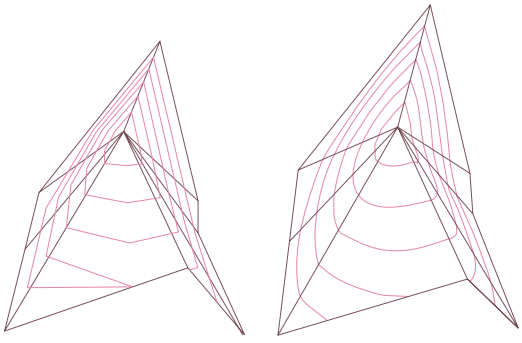
In order to provide a smooth description of the front, at each time step, new front particle emitters are deposited dynamically between the diverged particle emitters. If the distance between two consecutive particles becomes too large compared to the bounding box of the object (or any other geometry-driven criteria), we insert a new particle in between. To find the position and direction of the inserted particle, we tested two methods. In the first method, we start a new particle on the initial starting point of the front, with a tangent direction equal to the average of the 2 particles, and we compute its path up to the current time. However, after a hyperbolic vertex, this first method fails since there is always a "shadow" cone that will never be reached.

To suppress this issue, we use a second method that uses the current positions and tangents of the two consecutive particles apart and simply adds a particle in between. When the consecutive particles are either on the same face or on adjacent faces, the insertion is easy to do. In the rare cases (for a mesh extremely irregular and non uniform) when the particles are too far away, we first find the faces lying inside a small bounding box of the segment we want to split, then project the midpoint of the segment to the closest face in the bounding box. The new tangent vector is simply the average of the two consecutive tangent vectors. This technique also has the advantage of being significantly faster than the first method since it doesn't have to start from the original position and simulate to the current time.

Inversely, we destroy particles in highly sampled regions when they become too close. This adaptive refinement of the front as illustrated in Figure 4.

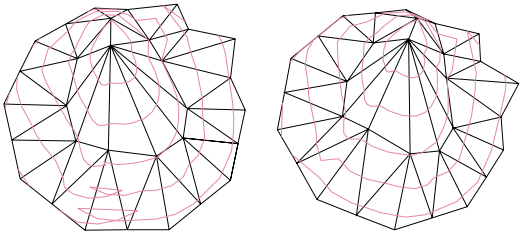
Using this adaptive sampling, the initial front can be sampled with a small number of points. This greatly reduces the memory and computational requirements for evolving smooth fronts on complex objects. As explained in the next paragraph, this ordering of the front also helps us to remove the undesirable interferences that may appear during propagation.

Suppressing Swallow Tails. Using this notion of an ordered front, we propose a simple technique to detect and handle these swallow tail effects. Since these tails develop large tangent discontinuities around the interference (see Figure 5(a)), we simply remove particles that have a large tangent difference with their immediate neighbor. As shown in Figure 5(b), this alleviates the interference problem, and leads to a smooth evolving front as desired.



a b

Fig. 4. *(a) No adaptive sampling: Many points must be created at the start in order to ensure an adequate discretization. (b) Adaptive sampling: Starting with a small number of points (eight) a smooth curve is created by adaptively refining where necessary.*



(a) (b)

Fig. 5. *Swallow tail effect: (a) after crossing a point, a front can develop a swallow-tail-like inteference. (b) By simply removing points with wildly different tangent directions, the front is fixed.*

2.3 Discussion

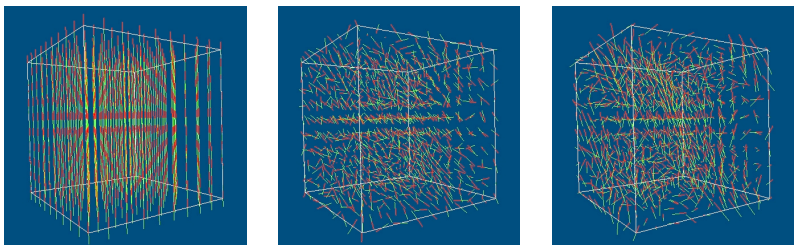
We have described in detail how to propagate fronts on a triangulated surface, regardless of mesh discretization. We have also described how every front particle follows a geodesic path at an arbitrary speed. Therefore, in the next section, we show how to use these different features to animate a realistic fire propagation on a 2-manifold.

3 Fire on Meshes

In this section we discuss in detail the process of animating fire propagation on meshes. In particular we discuss our wind field and front propagation models, effects of terrain slope, flame particles, avoiding double burning, and multiple fronts.

3.1 Wind Field Model

Using the front propagation and straightest geodesics described in the previous section we can propagate a fire front over our polyhedral model. However, using solely the geodesic flow to evolve the front results in extremely uniform and simple motion. This is due to the fact that we evolve the front using a constant geodesic velocity while the speed of fire propagation should depend on several complex factors including wind, fuel density, and terrain slope. Additionally, we currently have no mechanism to allow for animator control over the fire.



(a) (b) (c)

Fig. 6. *Wind field: (a) large scale (upward direction), (b) small scale , (c) sum of the two scales.*

To add both animator control and visual complexity to the front propagation, we drive the front velocity using a multi scale turbulent wind field model proposed by Stam et al.[10]. The wind field models wind velocity as a sum of two terms: a large scale term, $\mathbf{w}_l(\mathbf{x},t)$, used to describe global wind motion, and a small scale term, $\mathbf{w}_s(\mathbf{x},t)$, used to describe local turbulence (see Figure 6):

$$\mathbf{w}(\mathbf{x},t) = \mathbf{w}_l(\mathbf{x},t) + \mathbf{w}_s(\mathbf{x},t) \tag{2}$$

The large scale field can be specified by the animator or built by combining several wind field primitives. The small scale field is created using stochastic techniques to model turbulence. By separating the field this way, the animator has high level control and is not burdened with the specification of small, complex motions such as turbulence.

In order to reduce runtime computation the wind field is precomputed and stored in a 4D grid. During the simulation, the wind velocity at any point and time can be obtained by a 4D linear interpolation of the nearest grid points. Since the wind field can be constructed to have periodic boundaries, the grid can be tiled in both space and time. Additional wind fields can be combined with the precomputed field at runtime to allow for interactive wind field modifications (eg. mouse interaction).

3.2 Front Propagation

Given the wind field and the surface, we must now evolve the front in a plausible, visually complex way. We represent the front as a set of sample points, $\mathbf{p}$, each with an associated direction of propagation, $\mathbf{d}$, as explained in Section 2. The velocity of propagation for one of these points is given by:

$$\mathbf{v}(t) = \mathbf{d}(t)\,(1 + k_{wind}\,\mathbf{d}(t) \cdot \mathbf{w}(\mathbf{p},t)) \tag{3}$$

where k_{wind} is a coefficient specifying how strongly the wind influences the fire velocity and the term $(1 + k_{wind}\,\mathbf{d}(t) \cdot \mathbf{w}(\mathbf{p},t))$ is clamped to zero when it is less than zero. Note that this formulation reduces to the uniform geodesic flow in the absence of external wind. Using this velocity, we update the particle position using the geodesic flow technique of section 2 updating the tangential $\mathbf{v}(t)$ and $\mathbf{d}(t)$ when crossing edges and vertices.

We initialize the fire by placing a sampled front on the surface with the initial directions of the samples in the direction normal to the front. It is also possible to allow the user to specify the starting position of a fire by just clicking somewhere on the mesh. Given this starting position, we simply place several front samples at this location each with a different initial radial direction. The system then evolves the fire naturally and automatically.

3.3 Effects of Terrain Slope

In the absence of external wind the fire front should propagate up a vertical wall more quickly than across a horizontal floor. This is due to the upward convection of the air near the combustion area. We can simulate this by adding a wind-like term that opposes gravity. The velocity equation is then:

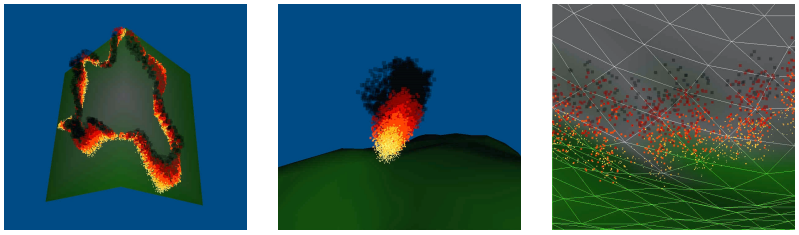
$$\mathbf{v}(t) = \mathbf{d}(t)\,(1 + k_{wind}\,\mathbf{d}(t) \cdot \mathbf{w}(\mathbf{p},t) - k_{slope}\,\mathbf{d}(t) \cdot \mathbf{g}) \tag{4}$$

where k_{slope} is a coefficient specifying how strongly the slope affects the velocity and $\mathbf{g}$ is the gravity vector.

Using this new velocity formulation the front is updated as described in the previous section. Our current implementation uses only the wind field and terrain slope to modulate the front velocity. However, it is possible to modify the velocity function to account for other factors including fuel density and fuel thickness. The animator can for instance spray-paint directly on the surface fuel densities, to direct the fire propagation at her will. However, our simple model using only wind and slope factors already leads to visually complex and realistic propagation over a mesh without any user interaction.

3.4 Flame Particles

In order to visualize the fire, flame particle emitters are deposited on the surface as the the front evolves. These emitters are deposited in the small area swept out by the front samples (see Figure 7). They can have values determined by local surface properties such as temperature, fuel density, and front velocity. These emitters then emit flame particles according to their internal state. These flame particles are standard particles as defined in [11]. Their motion is determined by a combination of advection due to the wind field, diffusion and decay, to simulate the integration of the PDE defining fire motion. Additionally, the size, transparency and color (from red when hot to dark grey for the smoke) of each flame particle can vary over time. After a period of time the particles and emitters die off.



a b c

Fig. 7. *(a) Flame emitters are deposited by the moving front. (b) Flame particles are emitted to give shape to the fire out of the surface. (c) Changing the luminance or the color of the vertices depending on their distance to the front allows for a smooth transition between standard surface and burnt surface. Note that we only used a sparse field of emitters in this figure to clearly see the mesh surface.*

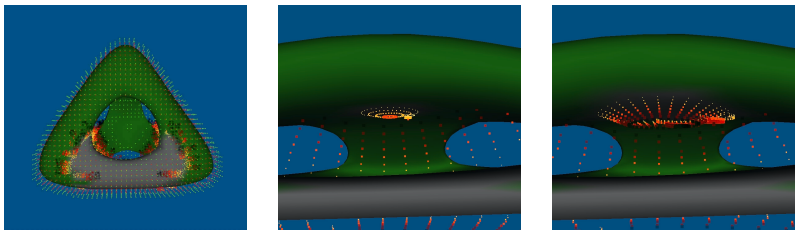
3.5 Blackening of Burnt Regions

Once a region has burnt down, we must change its color accordingly. In order to implement a smooth color change in the burnt area, the color of each vertex is darkened according to the distance to the fire front (to simulate the alteration due to the heat of

the flames). Each time a particle is advanced in a triangle (or across a triangle during its integration step), we increase the darkness of the vertices of the current triangle depending on the distance from the particle. To do so, we use a simple color lookup table to compute a luminance; if the current luminance of the vertex is higher than this new luminance, we change the color accordingly. When the three vertices of a triangle have been completely burnt, the whole triangle will be completely dark. If, however, only a corner has been burnt, the Gouraud shading used to display the mesh will provide a smooth ramping of the color. This effect can be seen in Figure 7(c), where we voluntarily reduced the number of flame emitters to clearly show the blackening.

3.6 Multiple Fronts

An important characteristic of fire propagation is that a flame can ignite another fire at a remote location. For instance, if a fire flame reaches an overhanging part of the object not yet burnt, a new fire front can develop at this point. In order to simulate this remote fire ignition, we should know how close to the surface each flame particle is. Computing the distance to the closest surface triangle at runtime would be prohibitive as the number of particles can be significant. Instead, we employed an implicit representation of the object's surface through a regular grid approximation. We used the closest point transform [5] to store the distance to the burning object on a coarse, regular grid, as a pre-process (Figure 8(a) shows the grid nodes around the object). We can then keep track of whether a particle penetrates the object by checking the interpolated distance from grid points near the particle. In addition, in order to quickly find the position where to start the new fire, an index of the closest vertex as well as the distance value will be assigned at each grid point before the simulation.



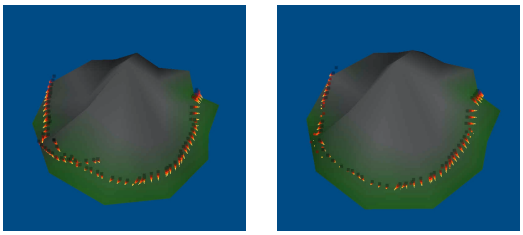
a b c

Fig. 8. *(a) The closest point transform generates the distance from each grid point to the closest surface as implicit representation. The color of the grid nodes indicates the proximity of the surface. (b) New fire starts at the intersection between a flame and a surface. (c) New flames start spreading from new origin. Note that each flame is represented by very few particles in order to illustrate the new ignition more clearly.*

Once an interpenetration is detected during the animation, we proceed as follows: we first move the flame particle out of the object (the flame will therefore "lick" the object's surface), and then, if this particle is still hot enough, we start another fire front at this very location on the surface. This new fire front is initialized and assigned a unique front ID (we use a simple integer counter that we increment each time a new front is created) for further distinction between fronts. This behavior is illustrated in Figure 8(b) and (c).

3.7 Fire Front Merging

Although we already solved the interference problem in the previous section (as depicted, this time for a real fire front, in Figure 9), there is another issue that needs to be dealt with: a fire front can/will eventually collide with itself or another fire front at some point during the propagation. The front must then to stop, since the area it is entering has already been burnt.



a b

Fig. 9. *(a) Swallow tail effect generated after a conjugate point. (b) The swallow tail is removed by simple particle deletions where an inversion of direction is detected (see Section 2).*

This case of "double burning", when two different fire fronts are to be merged into one fire front as shown in Figure 10(c)), can easily be avoided. Although it would be easy to keep a pointer on each triangle to the list of the front IDs currently propagating over it, and then find the exact intersection(s) within the triangle, we opted for a simpler procedure. Whenever a triangle becomes completely burnt, we simply declare the front particles on it as dead: they will no longer move or emit flame particles. As demonstrated in Figure 10, this is sufficient to deal with both self-intersection and collision of different fronts. Notice that this procedure does *not* change the fact that we still have an ordered description of every front: some parts are simply no longer moving, but they still form the contour of the whole front. If one was to set fires at different points on the mesh, the final contours of the fronts (once everything has burnt) would be the Voronoï regions of these starting points, if no wind and slope effects are taken into account.

3.8 Rendering

Our simulation is visualized in real time using the OpenGL library. We used glPoint to render our evolving particles. A particle can be displayed as a transparent disk using the GL_POINT_SMOOTH mode and the GL_BLEND mode of the OpenGL library, generating automatic billboards for the blobby particles. To ensure correct transparency, we have to order the points in decreasing depth value. The transparency of each particle is determined mainly by the distribution of mass, decay constant and size. The size is changed over time to account for diffusion [10]. The color of the flames changes with temperature which decreases with time, as does its mass. We assume that the color of a flame is a function of its distribution of mass and its size.

Obviously, our implementation uses only one of the many possible rendering techniques. Other more sophisticated renderings of flames [11, 1] could be used instead. However, our simple rendering method allows for real-time display, which is very convenient for rapid design of an animation.

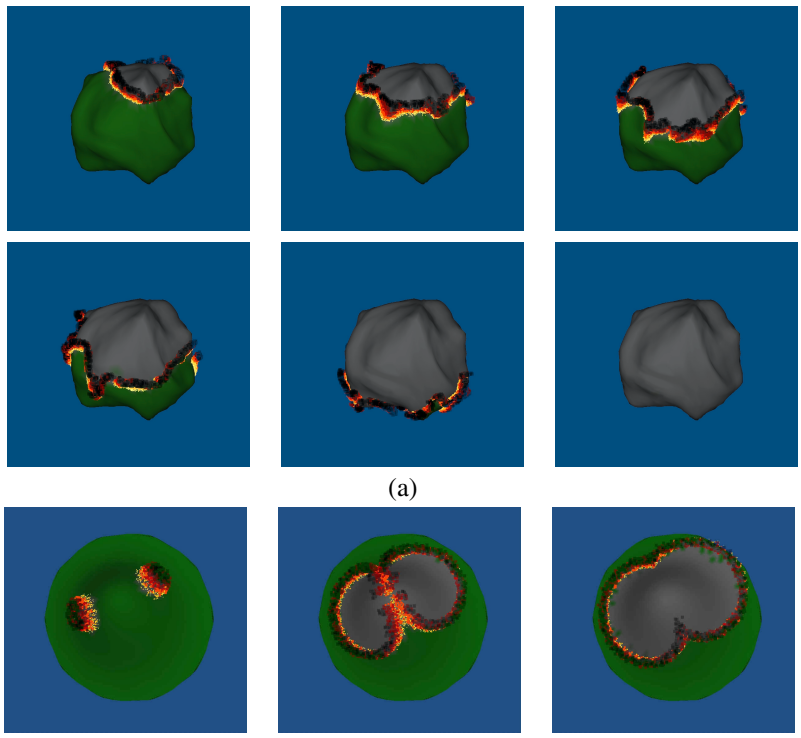


Fig. 10. *Grass fires: (a) Animation with wind and slope effects of a fire propagation on a mesh. (b) to (d): Fire fronts merging — when a front reaches a burnt region, it merges with the other fire front(s) appropriately. On all these snapshots taken during an interactive session, we can see the blackening of the mesh around the fire fronts.*

4 Results

The interactivity of our technique allowed us to test fire propagation on arbitrary meshes very quickly. Figure 10(a) illustrates a typical session, where the animator clicks on a point on the mesh to start near the top of the model: the flame front then propagates on the surface instantaneously. The shape, speed, and the direction of the front flames are influenced by wind, geodesic flow, and terrain slope, and the result has a natural look of real fire propagation. Blackening of burnt regions are also spreading as the front flames evolve. Other examples, like the heightfield on Figure 1(b) or the complex shape in Figure 8 demonstrates that we can handle any boundary or genus too, as long as the mesh is a 2-manifold.

5 Conclusion

In this paper, we developed a fire propagation technique designed for arbitrary triangle meshes. We define the fire propagation as a set of front particles following simple geodesics on the mesh, with a velocity depending on external wind and/or forces, slope, and other possible attributes. Contrary to previous methods, we define an lazy, adaptive front description, and handle conjugate points, front merging, and remote ignition

to mimic real fire behavior. We also offer control to the animator by describing the wind field as a sum of a small scale and a large scale field. We demonstrated that our approach leads to complex and visually realistic fire front propagation. We used [10] for rendering of the flame, but this technique is open to other good offline rendering techniques if necessary.

Merging geometric properties of the meshes and physical properties of the wind contributes to the realism of this natural phenomenon simulation. This combination of geodesics and random wind fields can also be applied directly to other complex simulations such as spreading liquids on meshes or for the visualization of heat distribution. Future work includes the handling of non-manifold meshes, which should be done by keeping track of bifurcations in the fire fronts happening at each non-manifold edge/vertex.

Acknowledgements

Many thanks to Eitan Grinspun for initial discussions and support. This work has been partially supported by the Integrated Media Systems Center, a NSF Engineering Research Center, cooperative agreement number EEC-9529152, and the NSF STC for Computer Graphics and Scientific Visualization (ASC-89-20219).

References

1. P. Beaudoin, S. Paquet, and P. Poulin. Realistic and controllable fire simulation. *Graphics Interface 2001*, 2001.
2. N. Chiba, S Ohkawa, K. Muraoka, and M Miura. Two-dimensional visual simulation of flames. *The Journal of Visualization and Computer Animation*, 1994.
3. A. C. Fernandez-Pello. Flame spread modeling. *Combustion Science and Technology*, 1983.
4. W. W. Hargrove. Simulating fire patterns in heterogeneous landscapes. *Ecological modeling 2000*, 2000.
5. Sean Mauch. Closest point transform. http://www.ama.caltech.edu/s̃eanm/software/cpt/cpt.html, 2000.
6. C. H. Perry and R. W. Picard. Synthesizing flames and their spreading. *Eurographics Workshop on Animation*, 1994.
7. K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. *Mathematical Visualization*, pages 135–150, 1998.
8. K. Polthier and M. Schmies. Geodesic flow on polyhedral surfaces. *Proceedings of Eurographics-IEEE Symposium on Scientific Visualization '99*, 1999.
9. W. T. Reeves. Particle systems-a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, pages 91–108, 1983.
10. Jos Stam and Eugene Fiume. Turbulent wind fields for gaseous phenomena. *Computer Graphics Proceedings, ACM SIGGRAPH*, pages 369–376, 1993.
11. Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. *Computer Graphics Proceedings, ACM SIGGRAPH*, 1995.
12. F. A. Williams. Mechanisms of fire spread. *Sixteenth Simpomsium on Combustion*, 1976.