Computational Aesthetics in Graphics, Visualization, and Imaging (2012)
D. Cunningham and D. House (Editors)

# A Technique for Art Direction of Physically Based Fire Simulation

A. Bangalore[1,2] and D. H. House[2]

[1]Dreamworks Animation SKG, USA
[2]Digital Production Arts, Clemson University, USA

**Figure 1:** *Wolf and lion shaped flames simulated and rendered using our method*

**Abstract**

*This paper presents a new approach to the art direction of individual flames in a physically based fire simulation. Fire, due to its warm colors and constant movement, often becomes the main attraction to the viewer's eye in a scene. Therefore, being able to control a fire simulation to obtain a desired look or shape is crucial if simulation is to be used to create a fire effect. Our technique provides control over this chaotic natural phenomenon at a fine level, enabling the artist to add character to flames and create highly stylized visuals. The fire system itself is a fully physics-based two-gas system, where flames are advected along convection currents generated by combustion. Our method provides artistic control of these convection currents, using a set of imported curves drawn by an artist. A full description of the implementation and performance of the fire system, and our control method is presented. The technique is illustrated with examples of highly stylized flame artwork rendered using our system.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling



DOI: 10.2312/COMPAESTH/COMPAESTH12/045-054

## 1. Introduction

Computer generated simulations of natural phenomena are widely used to produce visual effects in highly engaging movies and games. In general, efficient numerical methods are combined with physics-based dynamics to create general animation systems for visual effects. The computer generated effects produced by these systems would be very expensive if not impossible to create in real life.

Natural phenomena in visual effects are often required as environmental elements that interact with the characters or serve as a stage on which they perform. Rarely are these elements the "chief performers" in a scene. Therefore, having purely physically based controls for the simulation systems is often sufficient for their animation.

Fire, on the other hand, is an element that strongly attracts attention in a scene, rather than just behaving like a prop. One of the best examples of this would be the depiction of fire breathing dragons. Each dragon is characterized by the appearance of the fire it breathes, and thus the flames require careful art direction. For these purposes purely physics based controls, which typically provide only for the setting of various parameters and initial conditions, are not enough to provide the artistic control that is needed. Although there has been attention paid to the problem of tuning a simulation to produce a predictable outcome, for example see Bruckner and Möller [SB10], there has been no work that would provide the kind of control needed to produce highly stylized fire effects. Traditional production methods for influencing fire include subjecting a system of particles to wind forces, turbulence or other physics or noise based influences. While these methods can be very good at creating a realistic fire look, they require multiple iterations by an effects artist to get the desired result. Also, it is difficult to use these methods for fine artistic control of individual flames. The artist is reduced to tweaking a number of physics based parameters that tend to be unintuitive and unpredictable. Thus, it would be highly beneficial to provide control at the level of the shape and timing of individual flames, within a true physics-based fire simulation.

The focus of the work, described in this paper, has been on directly controlling convection currents within a physical fire model. The shape of the flame in a burn is controlled by convection currents induced by temperature gradients, which in turn are generated by combustion. Thus, controlling convection currents provides the secondary effect of controlling flame shape. The approach allows us to create "flame animations", as depicted in the wolf and lion examples shown in Figure 1. The input to our system is a set of space curves, drawn by an artist and imported into our system. The curves need not be fixed, but can themselves be animated via hand-controlled keyframing. This results in a very intuitive system, where the vision of the artist is precisely recreated. In addition, the idea of a flame as a paint stroke is incorporated into our control mechanism, which allows the flame to be shaped by varying the amount of fuel in different parts of the flame.

The system itself consists of the four main components shown in Figure 2: the *Fire System*, the *Fluid Solver*, the *Curve System*, and the *Volume Renderer*. These are explained in detail in Section 6. The *Fire System* is the central controller for the simulator, and accepts two sets of inputs from the user. The first is a set of physics based input parameters which includes the burn rate, the stoichiometric ratio, the oxygen density, amount of heat produced during combustion, the diffusion rate, and the source density of fuel. These parameters control the combustion reaction and the gas motion as described in the background section. The second is a set of B-spline curves that the user draws and then imports into the program. At each frame, the fire system computes the required data for the simulation step, using the fluid solver to solve the equations for fuel gas, heat and velocity. The resulting ignited fuel density in the grid is used as input to the volume renderer, which renders the frame.

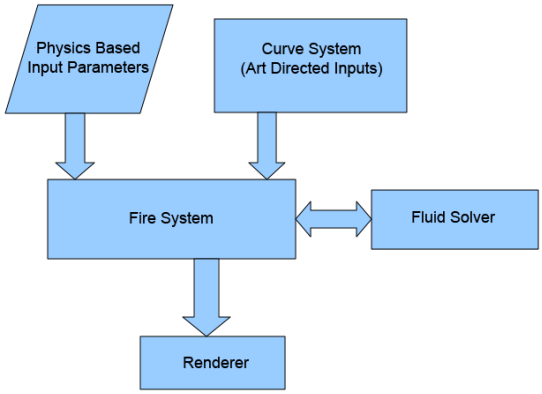


**Figure 2:** *Flowchart of the Fire Simulator*

## 2. Background

A flame is the visible, light emitting, gaseous part of a fire. Fire is the result of combustion, which is an exothermic reaction between a fuel and oxidant producing carbon dioxide, water and energy. As shown in Figure 3 fuel, preheated by ignition, when mixed with an oxidizing gas starts the process, creating a reaction zone at the boundary of the flame (flame front). The heated fuel at the center of the flame is broken down into smaller molecules and radicals due to the lack of oxygen. This is very luminous and gives flames their distinctive yellow color. As the reaction zone is approached, the increasing amount of oxidizing gas allows the chemical combustion reaction to occur. Combustion continues until the stoichiometric contour (flame front) is passed and the reaction is completed. The heat produced creates convection currents which control the shape of the flame [MK02].

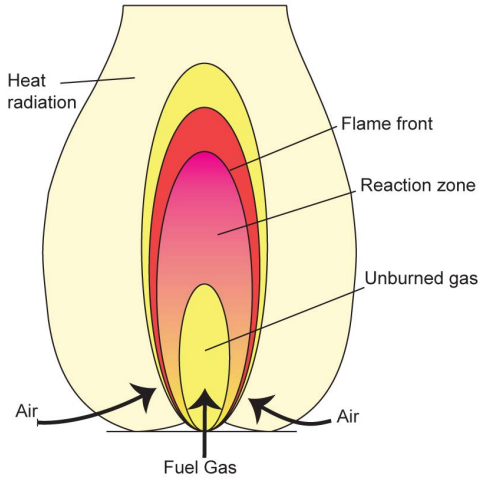


**Figure 3:** *A Simple Flame Model, redrawn with permission from Melek and Keyser. [MK02]*

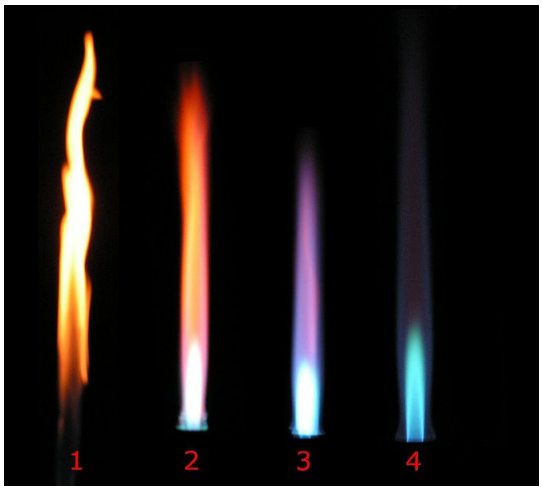


**Figure 4:** *Different Colored flames based on oxygen supply [wik]*

Flame propagation is explained by two phenomena: *heat conduction* and *diffusion*. In heat conduction, heat flows from the flame front to the inner cone, the area containing the unburned mixture of fuel and air. When the unburned mixture is heated to its ignition temperature, it combusts in the flame front, and heat from that reaction again flows to the inner cone, thus creating a cycle of self-propagation. In diffusion, a similar cycle begins when reactive molecules produced in the flame front diffuse into the inner cone and ignite the mixture [bri]. Diffusion or *laminar* flames are created when pure fuel gas from the fuel source mixes with the oxidizing gas through the mixture zone. If the fuel is mixed with oxidizing gas before ignition, a premixed flame is produced. The focus of our work is on diffusion flames.

In the most common type of flame, hydrocarbon flames, the most important factor determining color is oxygen supply and the extent of fuel-oxygen pre-mixing, which determines the rate of combustion and thus the temperature and reaction paths, thereby producing different color hues. Figure 4 shows four bunsen burner flames created by varying the mixture of fuel and oxygen. On the left, a rich fuel with no premixed oxygen produces a yellow sooty diffusion flame, on the right a lean, fully oxygen premixed, flame produces no soot and the flame color is produced by molecular radicals [wik].

## 3. Non-physically based fire models

The earliest reported fire model in computer graphics was by Reeves [Ree83]. This was based on particle-systems and some randomness. Reeves' method was used in the movie *Star Trek: The wrath of Khan* to create an expanding wall of fire. However, a large number of particles was required to hide the pointilistic nature of the technique and the fire had obvious grainyness.

Perlin noise based systems [Per85, Per02], using animated spatially and temporally coherent noise textures, are commonly used to enhance the look of particle systems to create pyroclastic and lava-like effects. This approach, depends heavily on randomness and tuning parameters, and thus is not appropriate when a high degree of control is desired.

Beaudin et al. [BPP01], introduced a set of techniques used together to produce realistic looking animations of burning objects. They used *flames* as primitives instead of particles to model the fire. Their flames are essentially deformable chains or skeletons of vertices rooted on the surface of an object. Flame genesis and animation consist of 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. Implicit surfaces are generated around these chains and displayed using a volumetric technique. The concept of flame skeletons helps in microscopic control of the flames but the calculation of the vector field to modify particles is difficult for artistically driven flames. The implicit surfaces required for visualization are complex and difficult to generate for artistic cases.

Lamorlette and Foster [LF02] describe a non-physics based fire model with focus on controlling the fire using parametric space curves. These curves evolve over time according to a combination of physics-based, procedural and hand-defined wind fields. A cylindrical profile is used to build an implicit surface representing the oxidation region and particles are sampled close to this region using a volumetric falloff function. Two levels of noise are added to provide turbulent detail. This system can be used for animating

a variety of natural fire effects and provides complete control over the large-scale behavior.

The non-physical models described above use some kind of noise to add the turbulent detail that occurs as a result of the combustion reaction. These noise functions are a drawback when very fine level control is desired for highly stylized fire. In this paper we use a spline based control system similar to the one described by Lamorlette and Foster. However we use these control curves to control convection currents in a physical simulation to achieve fine control of the flame.

## 4. Physically Based Fire Simulation

A fire can be thought of as fluid simulation modeling two interacting fluids, and heat. The fluids are a fuel gas, and an oxidizing gas, and the heat affects the fluid's density, thus inducing convection currents. The resulting complex behavior of the fluid mixture can be described mathematically by the Navier-Stokes equations. Thus, a solver for this equation is an integral part of a physically based fire simulator. Various fire simulation models, starting from this approach, have been proposed in the literature, notably those by Nguyen et al. [DN02], and Melek and Keyser [MK02]. The work reported here uses the latter as a foundation for art directed fire.

Fluid simulation methods are of two types, Eulerian, which discretize space by observing the fluid through gridded data and Lagrangian which discretize the fluid itself into smaller pieces each with its own set of parameters. A pure Lagrangian method like Smoothed Particle Hydrodynamics [MCG03] is useful for modeling chaotic behavior like splashes,etc but produces discontinuity in the fluid, which we do not desire. Thus, in our work we use the Semi-Lagrangian Stable Fluid solver, as described by Stam [Sta99], where advection in the fluid is computed using a Lagrangian approach, but all forces are handled in an Eulerian framework.

### 4.1. Stable Fluids

In compact form, the incompressible Navier-Stokes equations governing fluid flow are

$$\frac{\partial u}{\partial t} = f - (u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu\nabla^2 u, \quad (1)$$

$$\nabla \cdot u = 0. \quad (2)$$

Equation 1 represents the conservation of momentum in the fluid and equation 2 represents the constraint that the fluid be incompressible, requiring a divergence free flow. In the equations $u$ is the velocity field, $p$ is a pressure field, $\nu$ is the kinematic viscosity of the fluid, $\rho$ is the density and $f$ is an external body force field, such as gravity. The second term of equation 1 accounts for the advection of the velocity field on itself. The third term accounts for the forces due to pressure gradients, and the fourth is the diffusion term, which accounts for viscous forces.

Advancement of these equations for each time step is achieved by updating a candidate velocity field in three main steps (full details are given in [Sta99]):

1. Advection : This is done using a Lagrangian approach, tracing each position in the velocity field backwards one time step and updating the current velocity with the velocity found by backtracing. This preserves fluid momentum. In our fire solver, temperature (heat), as well as velocity, is also advected in this way.
2. Diffusion : This step adds a force due to fluid viscosity $\nu$, tending to align adjacent velocity streamlines. It is computed using an implicit Euler integration step based on finite differences over the fluid computation grid.
3. Projection : This step enforces incompressibility of the flow by making it divergence free. This effectively reproduces the effect of pressure on the fluid, and is responsible for the production of vortices in the flow. A Poisson equation is solved to obtain a pressure field, and the resulting pressure gradient is applied to the velocity field to project the current field onto the nearest (in the least squared error sense) divergence free velocity field.

The fire model described in the next section uses this fluid solver to compute the velocity, fuel and temperature fields at every time step of simulation. The fire model also has additional routines to account for the combustion reaction that occurs.

### 4.2. Fire Model

Heat is also transported with the flow of the air, and affects the buoyancy forces and changes the flow accordingly. Sufficient heat, and an appropriate mixture of the air and fuel gas, creates a combustion reaction, releasing more heat into the system. This creates convection currents which give the flame its shape. Later, we will explain how we art direct flames by controlling the convection currents. Here, we outline the fire simulation model that we use.

Melek and Keyser [MK02], and Melek [Mel08] present a physically accurate model of fire, which we have incorporated into our system. They make several assumptions and simplifications that we carry over into our implementation. These are, quoting loosely from Melek and Keyser [MK02] pp. 6-7:

- The fuel gas is uniform. That is, the fuel is either burned or unburned, and that any combustible byproducts behave exactly the same as the original fuel, and that the amount of heat produced is a function of just the amount of fuel burned.
- The compression of the gases as a result of explosive combustion is ignored. Since basically stable flames are simulated, it is unlikely that this compression will have a significant visual effect on the solution. The total amount of

gas (fuel, oxidizer, and exhaust) in a unit volume is assumed to be constant.
- Heat and temperature are treated interchangeably.
- The pressure effects of gases at different temperatures are not modeled directly. However, the most significant effects (buoyancy of hot air, and spread of heat through the fluid) are modeled.

#### 4.2.1. Gas motion in the fire model

The velocity field computed by the fluid solver described by Stam [Sta99] is used to advect fuel gas, and heat. Smoke could also be advected, but we do not presently incorporate smoke into our flame model. Melek and Keyser [MK02] give the equations for the motion of fuel gas and heat as

$$\frac{\partial g}{\partial t} = -(u \cdot \nabla)g - \alpha_g g + S_g, \tag{3}$$

$$\frac{\partial T}{\partial t} = -(u \cdot \nabla)T - \alpha_T T + K_T \nabla^2 T. \tag{4}$$

Here $K_T$ is a diffusion constant, $\alpha_x$ is a dissipation rate and $S_g$ is the fuel gas source term. Thus, the characteristics of the motion of fuel and heat can differ significantly, though they are carried by the motion of the same fluid system. The external force acting on any one cell in an $(x,y,z)$ coordinate frame, where $y$ is the vertical direction, is

$$F = f_g g_g \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} + f_T(T - T_{amb}) \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{5}$$

where $f_g$ and $f_T$ are positive constants controlling the force components based on gravity and temperature respectively, and $T_{amb}$ is the room temperature. Hot air will thus rise, and cold air fall, creating convection currents necessary to give the correct flame shape. The fuel gas will tend to fall, though this effect is usually not very noticeable (i.e. $f_g$ should be fairly small).

#### 4.2.2. Combustion

Melek and Keyser [MK02] also give the equations for combustion in a cell, where $T > T_b$, as

$$C = r\min(d_A, bd_g),$$
$$\frac{\partial d_g}{\partial t} = -\frac{C}{b},$$
$$\frac{\partial T}{\partial t} = T_0 C.$$

Here $d_x$ is the density, $r$ is the burning rate ($0 < r \leq 1$), $b$ is the stoichiometric mixture (the amount of air required to burn one unit of fuel), $T_b$ is the lower flammability temperature threshold for burning to occur, and $T_0$ is the output heat from the reaction. The burning rate is the percentage of the gas that can be burned in a second.The oxygen density is defined as

$$d_A = D - d_g$$

where $D$ is the total amount of gas in each cell, which is constant and $d_g$ is the density of fuel gas in that cell. If there is no fuel gas in a cell then it is filled with oxygen, and if there is no oxygen it is filled with fuel gas. The fact that air is not all oxygen can be easily accounted for by adjusting the stoichiometric mixture appropriately. The heat output coming from a combustion cell can be sufficient to start a reaction in a neighboring cell. Or it might not be sufficient and the combustion reaction cannot continue, extinguishing the flame. Practically, the combustion inside the cell is controlled with four parameters: $T_b$, $b$, $T_0$, and $r$, controlling the temperature above which ignition occurs, the oxygen requirement for combustion, the intensity of the reaction, and the burn rate.

## 5. Volume Rendering

The heated fuel at the center of a flame is broken down into smaller molecules and radicals due to the lack of oxygen. This is luminous and gives the characteristic yellow color to the flame. The reactions at the flame front release energy and some of this energy may be released in the wavelength of visible light based on the type of fuel and oxidizer used. Volumetric rendering methods are most suitable for rendering flames of this kind. We use a simplified version of the ray marching method described in the *Volumetric methods* course presented at SIGGRAPH 2010 [WZC*10].

Starting a ray march from the position of the camera, through a position on the image array, a sampling position is iterated along the ray in small steps, until a predetermined maximum distance is reached. The color $C_a$ accumulated along the ray is initially set to 0, and the transmissivity $T$ is set to 1. These are updated at every step of the march according to the update equations:

$$\triangle T = e^{-\triangle s \rho(\mathbf{x})\kappa},$$
$$C_a^{n+1} = C_a^n + C_d(\mathbf{x})T^n(1 - \triangle T),$$
$$T^{n+1} = T^n \triangle T,$$

where $\mathbf{x}$ is the 3D position of the sampling point along the ray, $\rho(\mathbf{x})$ is the density, and $\kappa$ is the scattering coefficient.

## 6. Implementation

As we have indicated, our flame simulation system consists of four main components: the *Fire System*, the *Fluid Solver*, the *Curve System*, and the *Volume Renderer*. These are explained in detail below.

### 6.1. The Fluid Solver

The *Fluid Solver* solves the equations at every time step for the velocity field (equations 1 and 2), fuel gas (equation 3) and heat (equation 4). Stam's Semi-Lagrangian stable fluid solver [Sta99] is used as described in the background section. The fluid solver is implemented as a C++ object which

**Table 1:** *Fluid Solver Functions*

| Fluid Solver Functions | |
|---|---|
| `addSource` | Add fuel from computed sources |
| `setBoundaries` | Enforce boundary conditions |
| `linSolve` | Linear equation solver |
| `diffuse` | Diffusion operation |
| `advect` | Advection operation |
| `project` | Projection operation |
| `computeVelocity` | Solves eqns 1 and 2 |
| `computeDensity` | Solves eqns 3 and 4 |

the Fire System uses for the simulation. This was extended to three dimensions and converted to a C++ class with separate methods to solve for fuel gas, heat and velocity.

Table 1 shows the important functions in the fluid solver. The `computeVelocity` function is called for the velocity followed by the `computeDensity` function for the fuel gas and heat and at the beginning of the simulation time step. The `computeDensity` function first calls the `addSource` function to add fuel sources as computed by the fire system. Then the `diffuse` function performs the diffusion operation, and `advect` function performs the Semi-Lagrangian backtrace operation for advection. These two functions call the `linSolve` function to solve a system of equations and the `setBoundaries` function to enforce boundary conditions. The `computeVelocity` function has an additional step of making the field divergence free for which it calls the `project` function.

### 6.2. The Curve System

The *Curve System* handles all of the artistic input from the user. Its input is a set of curves drawn and animated by the user in *Autodesk Maya*. Although *Maya* was chosen because of its ease of animation, any other software package with animatable curve drawing capabilities could be used. Curve data is exported into a sequence of files by a *Python* script which generates one file for each frame of animation. The *Curve System* contains the number of curves the user is inputting, the array of curves input by the user, and a method to read the sequence of input files. The required data from the user for each curve is:

- The number of control points in each curve.
- The number of interpolating points in a curve.
- The order of the curve. (Must be less than the number of control points)
- A flag to indicate if the curve is a root source of fuel (active and ignited from the beginning of the simulation) or a branch source (ignites when heat from combustion reaches it).
- A scale in the range of 0 to 1 to control the amount of fuel emitted for each curve.
- A list of control points. (Must match the number specified before)

The fuel sources are perpetual emitters by default. Additional parameters can be added, if necessary, to control the lifetime of the flame. This can be easily implemented as the system keeps track of time as well as the frame number.

The curves are all implemented as B-splines, and C code for B-splines from the *Geometric Tools* open source library [Too] was used for curve interpolation. Additions were made to include the flag for the fuel source, the scale factor for fuel in each curve and to read the file input data at every frame.

### 6.3. The Fire System

The fire system is the central part of the program which is connected to all the other components. It houses data structures for the coarse grids containing the simulated quantities of velocity, fuel and heat and the directions of buoyancy and gravity at each cell. It contains methods to start simulation, calculate forces, sample and query fuel and velocity at any point in the grid and control combustion.

**Table 2:** *Fire System User Input Parameters*

| Fire System User Input Parameters | | |
|---|---|---|
| Name | Description | Value |
| numDivisions | Voxels per grid side | 80 |
| sideLength | Length of grid side | 14 |
| timeStep | Simulation time step | 0.04 |
| diffusionRate | Heat diffusion rate | 0.01 |
| threshTemp | Threshold combustion temperature | 0.0 |
| $f_g$ | Gravity scale factor | 0.1 |
| $f_T$ | Buoyancy scale factor | 0.2 |
| burnRate | Fuel exhausted per unit of oxygen | 1.0 |
| stoiMix | Oxygen in 1 unit of air | 1.0 |

The fire system has physics based parameters to control the oxygen density, stoichiometric ratio of air, the rate of oxidation and the amount of heat released during combustion. Table 2 presents a list of all the data contained in the fire system, and typical values that we used to produce the results shown in this paper.

According to equation 5, only two constants are required to control the forces due to gravity and buoyancy in any cell of the system. The main idea in this thesis is to use curves to control the direction of the gravity and buoyancy vector at each cell. The control curve is drawn at the desired position, as shown on the left side in Figure 5, and tangents to the curve are calculated for sample points at regular intervals along the curve. These tangents are used as directions for gravity and buoyancy in the system. They are injected into the force direction grid using the algorithm discussed below.

A volumetric splatting algorithm [LH91] is used to inject force direction and fuel gas at any point in the 3D grid. If $\nu$ amount of the fuel has to be injected at a point $P(x,y,z)$,

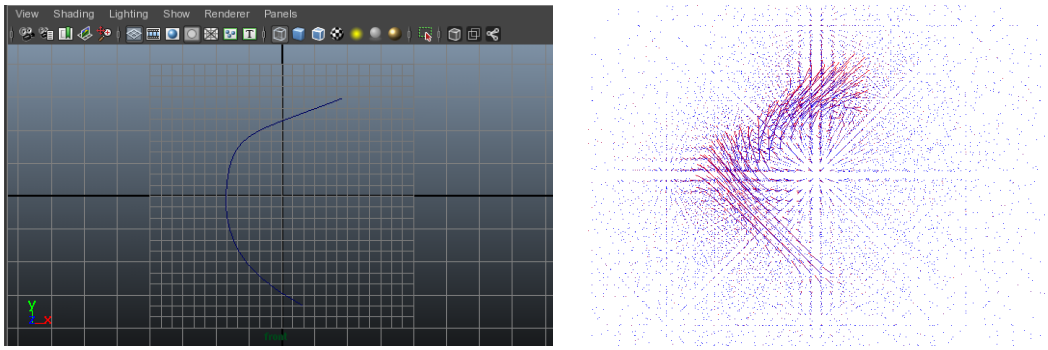


**Figure 5:** *A Maya Curve and the corresponding velocity field*

then let $a,b$ and $c$ represent the decimal part and $i,j$ and $k$ represent the integer part of $x,y$ and $z$ respectively (e.g. $i = floor(x)$ and $a = x - i$). Then by linear interpolation, for all $\alpha,\beta,\gamma \in \{0,1\}$,

$$\rho_{(i+\alpha,j+\beta,k+\gamma)} = \nu(1-a+\alpha)(1-b+\beta)(1-c+\gamma).$$

The force direction injected into the system based on the user-drawn curves directs the heat currents. The fluid solver ensures incompressible flows and populates the entire velocity grid as explained in the background section. This results in art directed advection with fluid behavior. The image on the right in Figure 5 shows the generated velocity based on the input curve to the left. The fuel gas and heat are then advected along this velocity field.

The sources of fuel are injected at the first point of each curve at the start of the simulation. The root fuel sources are ignited when the simulation begins. The combustion reaction begins and generates heat and consumes the fuel. The convection currents are directed along the control curves by the velocity field. The heated fuel is advected by the convecting heat currents and the flame is obtained. Branch curves are ignited only when sufficient heat from one of the root sources reaches the first control point.

However, due to the numerical dissipation in the fluid solver, and diffusion of heat in the grid, the flames start fading and the sharpness is lost when they are advected away from the sources. This behavior, although numerically correct, is a problem for creating stylized flames. This is fixed by injecting a small amount of fuel along the curve at each one of its interpolating points, when the advected heat reaches that point. The quantity of fuel injected at a curve point is made proportional to its distance from the first point of the curve. Mathematically, if $N$ represents the number of interpolating points (the resolution) of the B-spline, $(i_1,\ldots,i_n)$ and if $0 < \alpha \leq 1$ is a scale factor, then the injected density at a point is

$$\rho_i = \frac{N - \alpha i}{N-1}\rho_0$$

where $\rho_0$ is the fuel gas density at the first control point of the curve. $\alpha$ can be regulated to control the tapering of the flame and its length along the control curve. This miniscule amount of fuel ignites when the heat is sufficient and the additional convection currents generated stabilize the shape of the art directed flame.

### 6.4. Renderer

Once the new fuel gas and heat information are computed, the flame is imaged using a volume renderer, as described above. Ideally the color of a flame would depend on the mixing of oxygen with the fuel and the temperature. The luminous heated fuel has a yellow color and the reaction zone where the combustion happens has a reddish color. However, flame rendering is not the focus of this paper, so a simple approximation is used as the color of the flame. The quantity of fuel gas density in each voxel acts as a scaling factor for a base orange color ($R = 1.0$, $G = 0.3$, $B = 0.0$ was used for our examples), which can be changed by the user. When the fuel gas density is high the color is brighter, indicating the yellow luminous part of the flame. The consumption of fuel gas due to combustion decreases the fuel gas density yielding a darker shade of color at the edges of the flame. This color is then accumulated by the volume renderer to render the image.

## 7. Results

In order to test the technique presented in the above sections, different cases of stylized flames were used. The tests ran on a computer running a linux operating system with an Intel Core 2 Quad (2.4 Ghz) processor with 4 GB of ram. The tests were all CPU based and no GPU acceleration was used. Autodesk Maya was used to draw and animate the control curves. No optimizations like bounding boxes were added to the renderer.

In the first case, the image of a set of flames shaped to form a flower shown in Figure 6, was recreated through art

directed simulation. Eight rendered frames from the simulation are shown in Figure 6. The flames grow from combustion and form the shape of the flower, then the convection currents that build up from the heat produced gradually distort the shape of the flower. The simulation was done on a 80x80x80 grid and the images were rendered at a resolution of 800x800 pixels. The simulation was in real time (approx. 0.03 seconds per frame) and the rendering took approximately 10 minutes a frame. Root curves were used at the tips of the petals and the stem. Branch curves were used for the leaves.

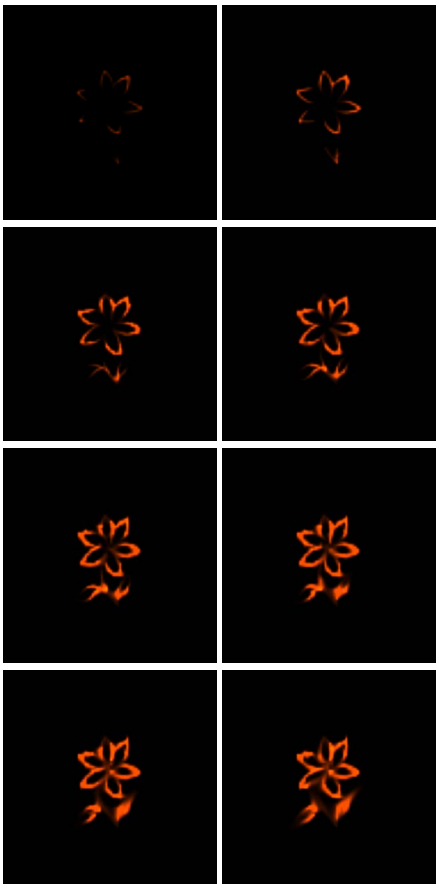

**Figure 6:** *Frames from "Fire Flower" animation*

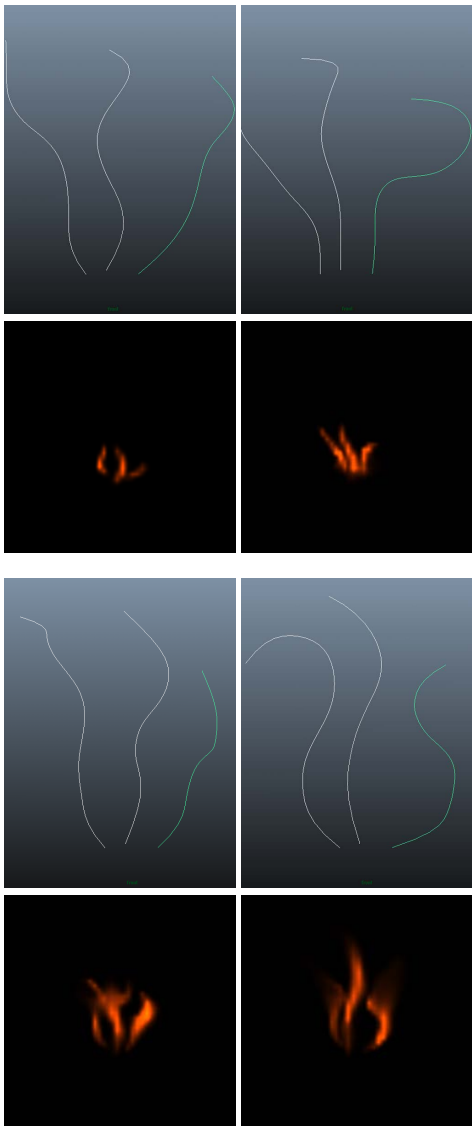

**Figure 7:** *Animated curves controlling a flame*

In the second case, three control curves were animated in Maya to create a stylized version of fire. Figure 7 shows four frames, out of one hundred, of the curves being animated as the fire grows, and the corresponding rendered frames. Clearly, the flames follow the curve animation, while exhibiting the fluid motion of fire. The simulation was done on a 40x40x40 grid and the images were rendered at a resolution of 500x500 pixels. The simulation was in real time (approx. 0.01 seconds per frame) and rendering took approximately five minutes per frame. All three curves are root curves.

Figure 1, at the start of the paper, shows rendered images of flames art directed to form wolf and lion shapes. 38 control curves were used to create the wolf image, and 45 control curves were used to create the lion image. Figure 8 shows a rendered image of flames directed on a set of input curves representing a phoenix bird. Twenty-one control curves were used to create it. The simulation was done on a 120x120x120 grid and the images were rendered at a resolution of 800x800 pixels. The simulation took approximately 2 seconds per frame and rendering took approximately 10 minutes per frame.

## 8. Conclusion

We have presented a technique to enable art direction of a physically based fire simulation. This frees the artist from using non-intuitive physical control methods like wind fields, noise and turbulence. Using the technique presented, the artist can draw and animate control curves in a familiar software package and then export them into the simulator. The simulator runs fast at relatively low resolutions ($\leq 100$ voxels per side) to enable real time visualization using OpenGL. Thus, the artist validate and adjust behavior and movement before doing the final flame rendering. We have demonstrated the technique to create single frame renders as well as animations for highly stylized flames, which are very difficult to produce using traditional physics based control methods. The results show that the simulated flames precisely follow the art direction and the resulting images retain an artistic feel.

There are several possible extensions and improvements that could be added to the model. For example, the volume renderer could be made more physically accurate by considering fuel-oxygen ratio, temperature and the combustion reaction at the flame front for color computation. The fluid solver could be upgraded to a solver with less numerical dissipation than Stam's stable fluid solver [Sta99]. For example, a solver using advection approaches like the BFECC [KLLR05] or the Modified MacCormack [Akw05] method could greatly increase the turbulent detail in the flame.

## References


[Akw05] AKWABOA S.: A modified maccormack's explicit time marching scheme for solving the conservation equations. In *Proceedings of American Physical Society, 58th Annual Meeting of the Division of Fluid Dynamics* (2005). 9

[BPP01] BEAUDIN P., PARQUET S., POULIN P.: Realistic and controllable fire simulation. In *Proceedings of Graphics Interface 2001* (2001), pp. 159–166. 3

[bri] BRITANNICA.COM: Flame. http://www.britannica.com/EBchecked/topic/209358/flame. 3

[DN02] D.Q. NGUYEN R. FEDKIW H. J.: Physically based modeling and animation of fire. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 02) 21* (2002), 721–728. 4

[KLLR05] KIM B., LIU Y., LLAMAS I., ROSSIGNAC J.: Flowfixer: Using bfecc for fluid simulation. In *Proceedings of Eurographics Workshop on Natural Phenomena* (2005). 9

[LF02] LAMORLETTE A., FOSTER N.: Structural modeling of natural flames. In *Proceedings of SIGGRAPH 02* (2002), ACM. 3

[LH91] LAUR D., HANRAHAN P.: Hierarchical splatting: a progressive refinement algorithm for volume rendering. In *Proceedings of SIGGRAPH 1991* (1991), ACM. 6

[MCG03] MULLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation (2003)* (2003), ACM. 4

[Mel08] MELEK Z.: *Interactive simulation of fire, burn and decomposition*. PhD thesis, Texas A&M University, College Station, TX, 2008. 4

[MK02] MELEK Z., KEYSER J.: *Interactive Simulation of Fire*. Tech. Rep. TR 2002-7-1, Texas A&M Department of Computer Science and Engineering, 2002. 2, 3, 4, 5

[Per85] PERLIN K.: An image synthesizer. In *Computer Graphics (Proceedings of SIGGRAPH 85)* (1985), ACM, pp. 287–296. 3

[Per02] PERLIN K.: Improving noise. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 02)* (2002), vol. 21-3, ACM, pp. 681–682. 3

[Ree83] REEVES W.: Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics 2*, 2 (1983). 3

[SB10] S. BRUCKNER T. M.: Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1467–1475. 2

[Sta99] STAM J.: Stable fluids. In *Proceedings of SIGGRAPH 99, Computer Graphics Proceedings, Annual Conference series* (1999), ACM, pp. 121–128. 4, 5, 9

[Too] TOOLS G.:. www.geometrictools.com. 6

[wik] WIKIPEDIA: Flame. http://en.wikipedia.org/wiki/Flame. 3

[WZC*10] WRENNINGE M., ZAFAR N. B., CLIFFORD J., GRAHAM G., PENNEY D., KONTKANEN J., TESSENDORF J., CLINTON A.: Volumetric methods in visual effects. In *Proceedings of SIGGRAPH 2010* (2010), ACM. 5

**Figure 8:** *Phoenix on fire*