Directable, High-Resolution Simulation of Fire on the GPU

Christopher Horvath*
Industrial Light + Magic

Willi Geiger*
Industrial Light + Magic

Figure 1: *Three different frames from rendered animations of simulated fire. (Left) Fast moving fireball with sparks. (Middle) Twisting campfire with rotation. (Right) A heavy, dense, smoky wall of fire.*

Abstract

The simulation of believable, photorealistic fire is difficult because fire is highly detailed, fast-moving, and turbulent. Traditional grid-based simulation models require large grids and long simulation times to capture even the coarsest levels of detail. In this paper, we propose a novel combination of coarse particle grid simulation with very fine, view-oriented refinement simulations performed on a GPU. We also propose a simple, GPU-based volume rendering scheme. The resulting images of fire produced by the proposed techniques are extremely detailed and can be integrated seamlessly into film-resolution images.

Our refinement technique takes advantage of perceptive limitations and likely viewing behavior to split the refinement stage into separable, parallel tasks. Multiple independent GPUs are employed to rapidly refine final simulations for rendering, allowing for rapid artist turnaround time and very high resolutions.

Directability is achieved by allowing virtually any user-defined particle behavior as an input to the initial coarse simulation. The physical criteria enforced by the coarse stage are minimal and could be easily implemented using any of the wide variety of commercially available fluid simulation tools. The GPU techniques utilized by our refinement stage are simple and widely available on even consumer-grade GPUs, lowering the overall implementation cost of the proposed system.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling;

Keywords: fire, simulation, particles, GPU

*e-mail: {chorvath,wgeiger}@ilm.com

ACM Reference Format
Horvath, C., Geiger, W. 2009. Directable, High-Resolution Simulation of Fire on the GPU. *ACM Trans. Graph. 28*, 3, Article 41 (August 2009), 8 pages. DOI = 10.1145/1531326.1531347 http://doi.acm.org/10.1145/1531326.1531347.

1 Introduction

The visual characteristics which define fire present particular challenges for simulation. The flickering nature of fire derives from its very rapid time-evolution. Large gradients of temperature produce turbulent, vortical behavior in comparatively calm systems such as campfires or torches. Simulated fire must therefore have a very fine resolution in both space and time, consequently large-scale combustion simulations have been somewhat intractable for artist-driven, visual effects production applications.

Computer-generated fire for motion pictures has utilized a combination of techniques to date. A very popular technique is the use of filmed, extracted fire and smoke elements, rendered as camera-facing sprites. Such techniques take advantage of the fact that both fire and smoke have somewhat amorphous, soft, and rapidly evolving boundaries. Subsequently, the compositing of many such filmed elements can look believable. This technique has been used with great success in many feature films, most notably for the Balrog creature in "The Lord of the Rings" trilogy. Sprites need to be quite large to be visually effective, and naturally, filmed elements do not react to objects which were not present when they were filmed. Thus, sprite-based simulation of fire and smoke can suffer from a lack of fine detail, an inability to create fine tendrils beyond what may be captured in the elements themselves, and an overall softness. There also tends to be a lack of fluid continuity between the sprites themselves.

Recent work has utilized three-dimensional grids on which the Navier-Stokes equations are solved for velocity, density and temperature. These grids are usually augmented with advected or synthesized texture detail and then volume-rendered. Beautiful examples of this technique in recent films include Selma Blair's fire-engulfed character in the films, "Hellboy" and "Hellboy 2", and the slow-motion building explosion in "The Matrix: Reloaded". The advantages of this technique are that the resulting fire and smoke are able to fully react with other three dimensional objects, and the behavior of the simulation is more easily directable. Grid-based models suffer from limitations of both spatial and temporal resolution. A very large grid system might be $512 \times 512 \times 512$ grid points, and yet only capture half the level of detail needed to represent a film-resolution image of 2048 pixels or larger. Such a grid would be extremely resource consuming and time consuming to simulate and render, with an artist turnaround time of days rather than hours.

Our proposed technique utilizes components from both of the techniques described above. We begin with a traditional particle simula-

tion, defined by the artist, which may exhibit virtually any directed behavior. We require only that the simulation store with the particles a few scalar attributes representing fuel, mass and impulse. The first stage of our system is a coarse grid on which incompressibility is enforced and vorticity may be amplified. Particles in the simulation are allowed to move however the artist has directed them, and the coarse grid stage alters only the final positions and velocities of these particles at the end of each time step. The particle simulation is saved to disk at the end of this first coarse stage, with mass and momentum having been very roughly conserved. The second, refinement stage of our simulation is specific to a viewing projection. Within the frustum defined by a particular perspective projection (or a rectangular cube, if the viewing projection is orthographic), a number of evenly-spaced, two-dimensional "slices" are created onto which the particles from the course stage are projected. These slices represent independent, two dimensional grids on which a secondary Navier-Stokes simulation is performed in the GPU. Because the slices are only two dimensional and independent of each other, they can be extremely high resolution - our system typically utilizes slice resolutions of 2048×2048 or higher. Neighboring slices are kept continuous with each other by careful selection of particle projection kernels and careful application of properly anti-aliased driving forces. Finally, because the slices are computed independently, they can be computed in parallel across many separate GPUs, which drastically shortens artist turnaround time. Our implementation of the proposed system allowed for multiple takes of film-resolution fire to be produced by an artist in a day.

Our system does suffer limitations in the presence of very fast moving cameras and fire flows which are predominantly perpendicular to the viewing plane. While these effects can be handled to some extent, they do represent a limitation which we will discuss in more detail at the end of the paper.

2 Previous Work

Particles have been used for fire simulation since [Reeves 1983]. Notable examples include [Stam and Fiume 1995; Nishita and Dobashi 2001]. [Lamorlette and Foster 2002] extended the particle approach to model individual flames with chains of particles defining a curve. Although particles offer a good control mechanism to the animator, it is difficult to get realistic fine detail and coherent structures much larger than the individual particles.

Other approaches have used numerical solution of the Navier-Stokes equations on a 3-dimensional grid, such as [Foster and Metaxas 1997]. [Stam 1999] introduced an unconditionally stable form of this solution. Other authors [Fedkiw et al. 2001; Selle et al. 2005] have augmented this model with additional vorticity to compensate for the excessive diffusion at each time step. [Nguyen et al. 2002] used this framework to simulate fire by using a levelset to track the flame reaction zone through the simulation grid. With these methods it can be hard to model the highly turbulent detail of real fire, so [Hong et al. 2007] used third-order hyperbolic equations for the levelset evolution to create interesting wrinkled features on the surface. The disadvantage with systems based on a fixed grid is that the size of the smallest detail is limited by the grid resolution, and so it becomes prohibitively expensive to create large-scale effects with photo-realistic detail.

Several methods have combined the particle and grid simulation approaches by tracking particles through a grid on which the Navier-Stokes equations can be solved. This approach was used by [Zhu and Bridson 2005] to create animations of sand. Our coarse particle simulation stage uses a similar technique.

[Rasmussen et al. 2003] created animations of large-scale combustion phenomena that avoided many of the limitations discussed above by combining various techniques in a view-dependent way. Their method uses high resolution 2-dimensional simulations as a starting point. A 3-dimensional velocity field is created from one or more of these 2-dimensional "slices" by interpolated extrusion or revolution in combination with a high-resolution tiling Kolmogorov field. Particles are then advected passively through the 3-dimensional velocity field and rendered volumetrically by rasterisation on to a view-aligned frustum grid. This combination of 2-dimensional and 3-dimensional techniques allows the use of detailed underlying simulations, but the method is limited to phenomena that can be decomposed into one or more 2-dimensional slices, and the problems of rendering coherent structures from individual particles remain. By contrast, our method uses a fully 3-dimensional particle simulation to define the overall motion, and multiple view-aligned high resolution grid simulations to generate coherent structure and fine detail.

Other researchers have modelled high-speed combustion processes using compressible flow, such as [Neff and Fiume 1999; Yngve et al. 2000; Feldman et al. 2003]. We have not concerned ourselves with these phenomena, as we are primarily interested in modelling lower-speed combustion such as fire and flame, where the incompressible equations are sufficient.

3 Input Particle System

The coarse particle simulations used as input to the fire simulation and rendering system need to have a visually plausible fluid-like motion. The simulations must also be fully directable, easy to control, and fast enough to allow many iterations to be run by artists. We want to use all the standard particle simulation tools that artists are familiar with, but add a final stage to each timestep that allows selected fluid characteristics to be approximated and applied to the particles in a controllable fashion.

The coarse grid step quickly and efficiently solves for an approximate non-divergent velocity for the particles, while simultaneously adding a selectable amount of viscosity. The vorticity of the particle system can also be selectively amplified at multiple scales of detail. The system needs only a relatively small number of particles to control the subsequent refinement - the simulations shown in this paper had particle counts ranging from $20,000$ to $100,000$ particles per frame. Additionally, the grid sizes for the coarse grid step are small, usually no larger than $50 \times 50 \times 50$. Fine detail will be added in the GPU-based refinement stage, so this coarse grid step only needs to capture broad, directed motion. The emphasis is on rapid artist turnaround, rather than detail.

Our particles are defined by vector attributes position $\vec{P}_i$, and velocity $\vec{V}_i$, as well as scalar attributes radius R_i, fuel K_i, mass M_i, impulse J_i, and age A_i. The fuel, mass and impulse attributes are intended to be normalized values indicating how much each particle contributes to the fuel, density and momentum stages of the secondary fire refinement, respectively. A value of 0 represents no contribution, and a value of 1 represents full contribution. An easy way of creating obstacles or disturbances in the refined fire is to create "impulse-only" particles that have non-zero values for impulse, which means they will influence the velocities in the subsequent refinement simulation, but zero values for fuel and mass, meaning no fuel or density will be added by those particles.

We assume that a standard time integration update step is performed on the particles in which new particles are emitted and old ones destroyed, forces are gathered from fields and collisions, user-defined update rules are executed, and lastly a new velocity, $\vec{V}_i^{\star}$ is computed and integrated to produce a new position, $\vec{P}_i^{\star}$.

Our coarse grid is inserted into the update step between the calculation of the new velocity and the calculation of the new position. The

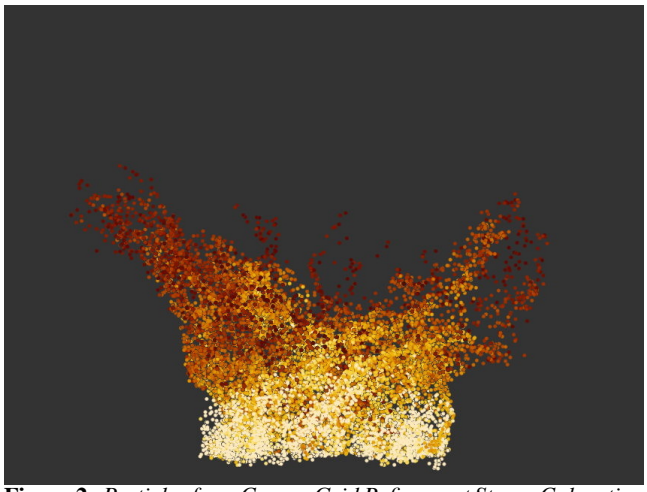

Figure 2: *Particles from Coarse Grid Refinement Stage. Coloration is a ramp between yellow and red corresponding to high and low values of particles attributes fuel, density and impulse.*

coarse grid step treats the new velocity as hypothetical and solves for a new, non-divergent velocity.

3.1 Coarse Grid Step

The Coarse Grid Step is a slight variation of the PIC/FLIP technique described in [Zhu and Bridson 2005]. Particle velocities are transferred onto a three-dimensional grid and incompressibilty is enforced on the grid to compute new, non-divergent grid velocities. Optionally, the vorticity of the non-divergent grid velocities can then be measured and amplified using "vorticity confinement", as described in [Fedkiw et al. 2001; Steinhoff and Underhill 1994]. The difference between the final grid velocities and the original grid velocities is transferred back onto the particles. Simulated viscosity is also selectively added by controlling how the velocities are transferred from the grid back onto the particles. For greater detail, we refer readers to the detailed description in [Zhu and Bridson 2005].

We augment the PIC/FLIP technique by using a simple wavelet decomposition on the velocity grid to produce multiple levels of detail. This allows us to amplify vorticity at multiple scales as well as accelerating convergence of the iterative incompressibility solution. An introduction to wavelet decomposition can be found in [Chui 1992]. A grid of exactly half the spatial resolution is calculated by low-pass filtering the original grid. The velocity values from this lower resolution grid are then interpolated at the positions of the cells of the original grid and subtracted from it, so the original grid then contains just the high frequency components of the velocity field that are not represented by the lower resolution grid. This process is repeated until we end up with a pyramid of velocity grids, each representing successively coarser scales of detail. The original velocity grid is exactly recoverable by recursively summing each detail level with the upsampled, coarser level of detail beneath it.

Incompressibility is enforced at each detail level, starting at the lowest and working upwards. At each level, after incompressibility is enforced and we have new velocities for that level, we can measure and amplify vorticity using vorticity confinement. The amount of vorticity amplification at each detail level is an artist-driven parameter, representing the amount of additional curling they want to introduce at small, medium, and large scales. Vorticity amplification is, by definition, divergence-free, so it does not affect the non-divergence of the new velocity field. Because each detail level is stored as only the high-frequency variation from the level beneath it, the non-divergence and vorticity amplification automatically travel up the pyramid of successive levels of detail.

There are many published works which describe efficient techniques for solving the non-divergence equation

$$\nabla \cdot \vec{U}^{\star} = 0 \tag{1}$$

where $\vec{U}^{\star} = (u, v, w)$ is the grid velocity in three dimensions. We use a standard "artificial pressure" Poisson solver, with simple Jacobi iterations towards convergence. More robust techniques exist, such as the Preconditioned Conjugate Gradient method. However, as we are only interested in a coarse solution and our multi-level approach minimizes the number of iterations we need for good convergence, we chose the simplest approach. For implementation details, we refer the reader to the detailed discussions in [Golub and Loan 1989; Foster and Fedkiw 2001]. Algorithm 1 summarizes the Coarse Grid Step algorithm.

Algorithm 1 Coarse Grid Step, computes approximately non-divergent velocity $\vec{V}_i^{\star}$ from hypothetical velocity $\vec{V}_i^{\star\star}$

1: Create three dimensional axis-aligned uniform grid encompassing particles
2: Calculate hypothetical new particle velocities from scene forces, collisions, and any other user-defined simulation sculpting.
3: Project hypothetical new particle velocities onto grid
4: Compute wavelet decomposition of grid into multiple detail levels
5: **for** each detail level, from the coarsest to the finest **do**
6: Incorporate upsampled new velocities from coarser detail level below
7: Loosely enforce incompressibility to get roughly divergence-free velocity
8: Optionally measure and amplify vorticity based on artist specification
9: **end for**
10: Project change in velocity back to particles with artist-specified artificial viscosity, as in [Zhu and Bridson 2005]

The Coarse Particle/Grid Step is typically iterated by artists without going to the secondary refinement stage until the desired coarse motion has been achieved. The particles are cached to disk before the refinement stage. Figure 2 shows one frame of the particles from a Coarse Particle/Grid simulation.

4 View-Specific Fire Refinement

At the end of the coarse particle/grid step, we have a fully computed particle set containing values for position, velocity, radius, fuel, mass, impulse and age. This coarse particle set has velocities that are roughly non-divergent, as well as large-scale vortices and viscosity. However, this coarse particle simulation contains very little fine detail and no uniquely fire-like behavior. The high-resolution details and fire-like physics are the responsibility of the view-specific refinement pass. Broadly, the refinement pass consists of creating a set of independent image planes parallel to a camera viewing plane, projecting attributes from the Coarse Grid Step particles onto these planes as forces and fuel, and then using the GPU to solve the two dimensional, incompressible Navier-Stokes equations on each image plane independently. Before getting into the specific details of this refinement, it is necessary to briefly examine why projecting the Coarse Grid particles onto independent simulation image planes is at all effective.

Our refinement pass takes advantage of several important observations regarding the perception of fire from a particular viewpoint. Fire is very fast-moving and turbulent, even in calm situations. Fine details are short-lived and tend to change quickly. Only the coarsest components of the fire flow generally remain present for more than

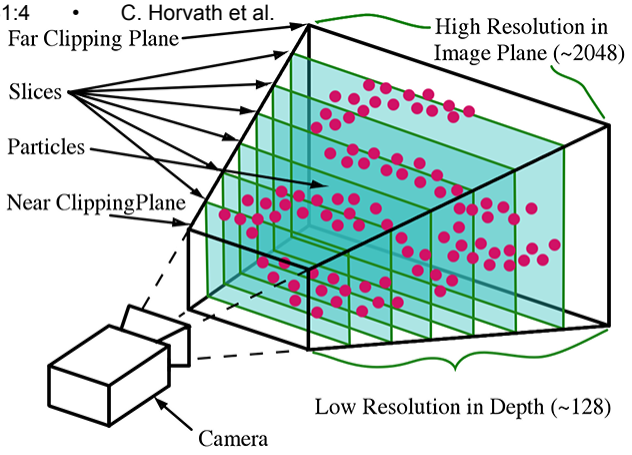

Figure 3: *Refinement simulation slices are aligned to the projection plane. They are spaced coarsely and evenly along the projection axis. Slices have very high resolution in the dimension parallel to the projection*

a fraction of a second. Additionally, the turbulent, fine details of fire do not have a large visual impact on the overall motion of the flame flow. From a particular viewpoint, fine variations of detail and movement which are perpendicular to the projection plane are not individually visible, and again, do not affect the large-scale flow very much. These observations help explain why camera-facing sprites do such a good job of representing fire and smoke, while having greater difficulty depicting phenomena like water splashes, where fine details like droplets tend to be persistent.

Based on these observations, we can surmise that as long as we globally preserve fluid-like behavior for large, coarse features in the fire, and as long as we have very high detail in the spatial dimensions which are parallel to the projection plane, we can ignore details perpendicular to the projection plane. Because these details do not significantly contribute to overall flow behavior, and because our underlying particle system already contains the coarse, global flow behavior, we do not need to concern ourselves with the high-resolution flow continuity in the direction parallel to the projection axis.

We can therefore create a series of high-resolution, two-dimensional simulation planes which are camera-facing and evenly spaced along the projection axis. As long as the spacing between these planar simulations does not exceed the Nyquist limit for sampling coarse features in the input particle flow, *we can simulate these two-dimensional planes completely independently*, without losing any important, visible features of the high-resolution fire. Continuity, momentum, and large scale features, even those which are primarily towards and away from camera, are maintained by the underlying particle system, which drives the refinement simulation. Thus, our refinement simulation takes the form of many large, independent, two-dimensional simulation "slices", evenly spaced from a near clipping plane to a far clipping plane within the viewing frustum. Figure 3 shows this configuration.

4.1 GPU Configuration

The two-dimensional refinement simulations are computed entirely on the graphics processing unit (GPU) using a combination of traditional geometry-based graphics processing operations and general purpose graphics processor computing, or GPGPU. By using specialized graphics hardware to perform our computations, we take advantage of the massive parallelisms inherent to GPU hardware and subsequently achieve large speed improvements over equivalent CPU implementations. This is an essential aspect of our system's design and usability. Modern GPUs are capable of as much as 50 times perfomance enhancement over multi-threaded, CPU-based algorithms, and this performance gain gives our system a fast enough turnaround to be iterative and useful to artists.

The GPU aspects of our system are built entirely in OpenGL 2.1, using the Open GL Shading Language, "GLSL".

[Harris 2003] presents a detailed analysis of GPU techniques for fluid simulation and the solution of the Navier Stokes equations. While that paper uses the "Cg" shading language rather than "GLSL", the techniques and problem formulation are structurally identical to our fire solver and form the basis of our system. The introduction of "slab operations", as well as various demonstrations of fluid solving techniques provide a robust simulation palette. We refer the reader to this thesis for all implementation details regarding the setup and operation of basic GPU fluid simulations.

4.2 Slice Refinement Overview

The refinement process for a single slice is a time-stepped, two-dimensional fluid simulation which iterates across a user-defined time-range, solving the incompressible Navier-Stokes equations. Our formulation of the equations and their solution follows [Harris 2003]. We differ from that work in the ordering of the major sub-steps in the update step, as well as with the addition of temperature, cooling, and thermal buoyancy. Our simulation state consists of identically-sized, rectangular planes of data:

Simulation Data Planes	
Inputs from Particles	Particle Fuel Particle Mass Particle Weighted Velocity
Simulation State	Density Temperature Texture Fuel Velocity Artificial Pressure

Each frame of refinement output produces two four-channel, floating-point images per slice. These images are the same spatial resolution as the slice simulation. The data is taken directly from the simulation state stored in the GPU, as described in table above. The first output slice image contains in its four channels, simDensity, simTemperature, textureDetail and fluidFuel, respectively. The second output slice image contains only two planes of data, simVelocityU and simVelocityV - the remaining two output channels are empty. The refinement slice images are used by the volume renderer to produce a final rendered fire image. See Figure 4 for examples of slice output images at a single frame.

Algorithm 2 provides an overview of the refinement simulation process. Each step will be described in detail. The time step is performed a fix number of sub-steps per frame, specified by the user. We do not explicitly calculate a CFL stability condition, and if the number of substeps is insufficient to resolve overly rapid motion, simulation discontinuities will occur. However, we have not found this to be a problem in practice - we default our system to 20 substeps per frame, and that seems to handle the normal range of velocities.

4.2.1 Particle Projection

The particles are projected onto the refinement slices to create three planes of data: *Fuel*, which adds temperature to the system, *Mass*, which adds density, and finally *WeightedVelocity*, which is treated as an impulse, ultimately adding velocity. These particle data planes are used additively with the existing simulation, they do not replace any of the refinement data. The particles are drawn from a viewing camera in the standard OpenGL fashion. A vertex shader

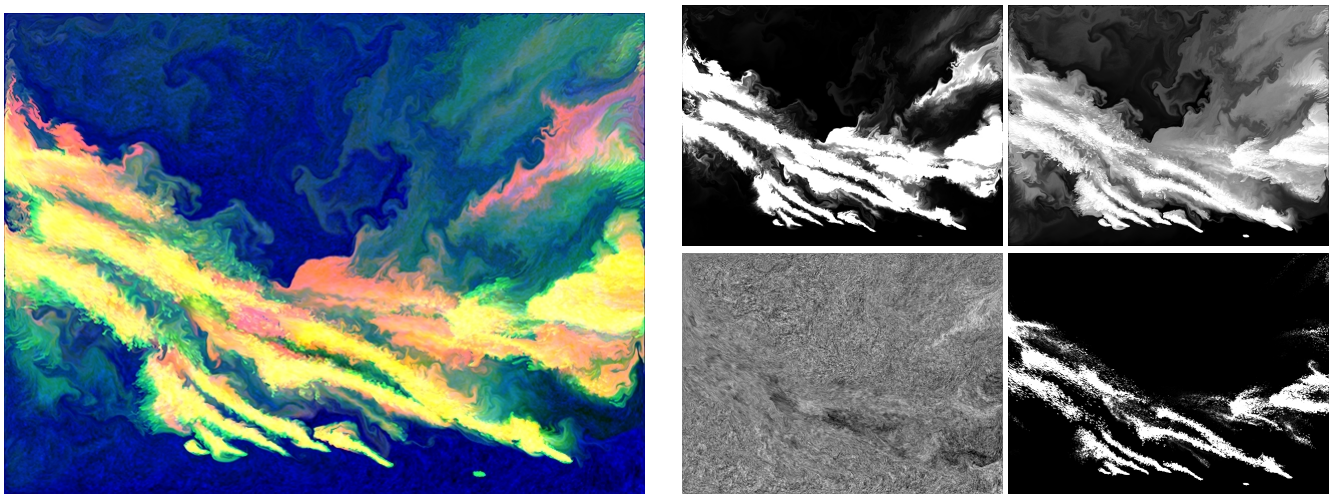

Figure 4: *Primary slice refinement output. (Left) Combined density, temperature and texture. (Right/UpperLeft) Density. (Right/UpperRight) Temperature. (Right/LowerLeft) Texture. (Right/LowerRight) Fuel.*

Algorithm 2 Fire Refinement Slice Simulation, Single Time Step

1: At each time step, project particles from viewing camera onto slice projection plane to produce forces, additional mass and fuel.
2: Cool simulation temperature, dissipiate simulation density, add new temperature and density from particles.
3: Enforce simulation edge boundary conditions.
4: Semi-Lagrangian advect fuel, density, temperature, texture, and velocity.
5: Add procedural texture to texture plane.
6: Damp velocity.
7: Compute vorticity confinement and add turbulence to velocity.
8: Add thermal buoyancy to velocity.
9: Enforce velocity incompressibility with Jacobi Pressure solver.
10: Save data into (density,temperature,texture,fuel) and (velocityU,velocityV) image files.

mimics the matrix transformations that would be performed by the standard hardware geometry pipeline, transforming the points from world space into raster space. For each particle at time t, we construct an interpolating hermite spline from the particle's position and velocity samples. The particle is drawn multiple times at random interpolating points along this interpolating spline, and the instantaneous velocity for each particle is derived at each drawing point from the spline itself. This jittered drawing creates smooth, curving projection of motion, and provides a user controllable ability to artificially elongate the projected input data in the direction of motion. This elongation smooths out the otherwise particulate nature of the controlling particle simulation.

As each particle is drawn, it uses its fuel, mass, impulse and radius attributes to control how it is drawn into the enabled output data planes containing Particle Mass, Particle Fuel, and Particle Velocity. Furthermore, each particle's contribution to the slice is weighted based on its perpendicular distance from the slice, using a gaussian kernel. If the perpendicular distance between particle i and this slice is d_i, and the uniform distance between slices is Δz, we create a gaussian weighting kernel with a support that covers at least 8 neighboring slices, like so:

$$w_i(d_i) = e^{(-d_i^2/(4\Delta z)^2)} \tag{2}$$

Having such a large support in the perpendicular direction ensures that the forcing variation between neighboring slices is very, very slight. By smoothly varying all of the inputs and all of the forcing in the perpendicular direction, we prevent the simulations in adjacent slices from diverging from each other too much.

After all the particles are drawn, the data planes contain what is essentially a weighted average of each of the projected particle inputs to the simulation. These input data planes can be modulated by a noise function or perturbed slightly to add additional detail.

4.2.2 Cooling, Dissipation, Heating, Density

The cooling, dissipation, heating and density step is the first of the "slab" operations, in which several two-dimensional data planes are operated upon to create an output data plane of exactly the same rectangular size. For most of these slab operations, the spatial mapping of input data points to output data points is almost exactly identity, which means the GPU can operate efficiently in a highly parallel way. Subsequently, performance for each of the remaining slab operations is optimal.

We use the cooling equation from [Nguyen et al. 2002], which is a fourth-order polynomial temperature function with a single user-defined cooling parameter. We have simplified the equation from the above paper by assuming that T_{air} is set to zero, which leads to the following relationship. If the temperature at a grid point is T, the cooling coefficient is C_{cool}, and the theoretical maximum system temperature is T_{max}, the cooling equation is defined as follows:

$$T^\star = T - \Delta t * C_{cool} * (T/T_{max})^4 \tag{3}$$

We decay (dissipate) the mass and fuel planes based on an exponential decay. Let the user-defined decay rates be γ_{mass} and γ_{fuel} for mass and fuel, respectively. Then the mass and fuel decay functions are:

$$Fuel^\star = Fuel * (1-\gamma_{fuel})^{\Delta t} \tag{4}$$

$$Mass^\star = Mass * (1-\gamma_{mass})^{\Delta t} \tag{5}$$

Fuel and Mass are added to the system from the normalized Particle input planes. We have a very simple model of fuel. Fuel entering the system from the particle input planes is assumed to be already reacted and enters the system at its maximum temperature. There is

no handling of the discontinuous region between unreacted and reacted fuel. We have not found this simplification to be an aesthetic problem. We do not allow the temperature plane to heat above the user-defined fuel combustion temperature, and we combine the particle input fuel with the simulation temperature plane using a maximum rather than an addition. Letting T_{fuel} be the user-supplied fuel combustion temperature, and remembering that the Particle Fuel planes are normalized to the range $[0,1]$ range, we update fuel:

$$Fuel^{\star\star} = maximum(Fuel^{\star}, ParticleFuel * T_{fuel}) \quad (6)$$

Mass (density) is added to the system similarly. There is a user-defined density rate, $K_{density}$, which is used to update mass from the Particle mass like this:

$$Mass^{\star\star} = maximum(Mass^{\star}, ParticleMass * K_{density}) \quad (7)$$

The reason that we use maximums and do not incorporate the time step Δt has to do with the artist-defined nature of the input particles. The Coarse Grid Step is sufficient to impart a base level of fluidity to the input simulation, but it does not guarantee any particular spacing or distribution of particles. It is therefore very likely that the user will create input simulations in which many particles bunch up in a particular area, which causes the Particle data inputs to have dense concentrations of mass and fuel in some areas. Updating via maximums is a quick way to prevent runaway mass deposits or hot-spots. There are more elegant ways of solving this problem, but we have not noticed any problems with our solution, and it performs consistently across a wide range of particle spatial concentrations.

4.2.3 Boundary Conditions

We do not explicitly handle collisions in the refinement simulation at all, we assume that any collisions are taken care of in the input particle simulation, or that the input particle simulation contains so-called, "impulse only particles" which have no fuel or mass but still affect simulation velocity. Therefore the boundary conditions are trivial and are only concerned with the edges of the simulation. We use Neumann boundary conditions, implemented exactly as described in [Harris 2003].

4.2.4 Semi-Lagrangian Advection

Semi-Lagrangian Advection is a technique decribed in [Stam 1999] which allows for fluid grid advection steps to be performed in an unconditionally stable manner for any size timestep. We follow the technique exactly as described in the paper to advect Temperature, Detail Texture, Fuel and Velocity.

4.2.5 Detail Texture Synthesis

It is useful to have a high-frequency pattern which evolves and flows along with the fluid. This texture pattern can be used at render time to add additional visual detail. It can also be incorporated directly into the simulation by modulating temperature and density. The bilinear resampling step in the advection phase filters out a small amount of high-frequency detail with each iteration, and over many iterations sharp details are quickly lost. Our texture synthesis scheme is very simple, as it is only intended to replace the highest frequencies of lost detail. The Texture plane is initialized to black, and at each time step a rapidly evolving fractal consisting of several octaves of time-evolving noise is added to the advected texture plane. Noise functions have an average value of zero across all spatial and temporal dimensions, so continually adding an evolving noise function to the persistent texture plane will not cause escalating values.

The detail which is lost due to successive advection steps is proportional to the spacing between pixels, because the bilinear resampling kernel has a support which is proportional to the pixel spacing. In the raster plane and over time we want to use a noise function which is rapidly changing and very highly detailed, so explicitly set the frequency of the finest octave of our noise in (x,y,t) to be the Nyquist limit of $(\frac{\Delta x}{2}, \frac{\Delta y}{2}, \frac{\Delta t}{2})$.

In the raster z direction (depth), we want to be somewhat more conservative, in order to preserve the continuity between adjacent slices. Therefore, along the projection axis we explicitly set the highest spatial frequency to half the Nyquist limit, $\frac{\Delta z}{4}$.

We therefore define our fractal noise to be one which has a maximum frequency, $\vec{\nu}_{max}$:

$$\vec{\nu}_{max} = (\frac{\Delta x}{2}, \frac{\Delta y}{2}, \frac{\Delta z}{4}, \frac{\Delta t}{2}) \quad (8)$$

For this and all other GPU-based noise generation, we use 4D Simplex Noise as described in [Olano 2005].

4.2.6 Velocity Damping

A user defined amount of damping is applied to the velocity prior to vorticity, turbulence and incompressibility calculations. For velocity $\vec{V}$, timestep Δt, and velocity damping coefficient K_{damp},

$$\vec{V}^{\star} = \vec{V} * (1 - K_{damp})^{\Delta t} \quad (9)$$

4.2.7 Vorticity and Turbulence

The Vorticity Confinement technique described in [Fedkiw et al. 2001] is actually a two-step process. First a scalar vorticity field Ψ is measured by taking the magnitude of the curl of the 2D velocity field. This vorticity is then multiplied by a user-supplied constant K_{vort}, which controls the amount of vorticity amplification, and then the curl of the new, amplified scalar field is added back to the velocity.

In [Bridson et al. 2007], the authors describe adding the divergence-free curl of a scalar field to a velocity field as a way of procedurally creating turbulence effects. They are essentially doing exactly the same thing as vorticity confinement, except instead of measuring and amplifying vorticity to come up with the scalar field that is curled to get an impulse, they simply invent a field from scratch using noise functions.

We combine the vorticity confinement and curl-noise turbulence techniques into a single simulation step. We call the procedural turbulence scalar field Φ, and proceed as follows:

$$\Psi = \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} \quad (10)$$

$$\Phi = userDefinedScalarTurbulenceField() \quad (11)$$

$$K = \Phi + (K_{vort} * \Psi) \quad (12)$$

$$\vec{J} = curl(K) = (\frac{\partial K}{\partial y}, -\frac{\partial K}{\partial x}) \quad (13)$$

$$\vec{V}^{\star} = \vec{V} + \Delta t * \vec{J} \quad (14)$$

The user-defined noise function here is subject to the same Nyquist limits as the procedural functions described in section 4.2.5, and the same style of 4D Simplex noise functions are employed.

4.2.8 Thermal Buoyancy

In this step, the velocity is updated by adding an "upward" impulse which is proportional to the temperature T relative to ambient "room" temperature T_{room}, times a user-defined constant, K_{bouy}, and in the direction of a user supplied unit vector $\vec{V_{up}}$.

$$\vec{V}^{\star} = \vec{V} + (\Delta t * K_{buoy} * (T - T_{room})) * \vec{V_{up}} \quad (15)$$

4.2.9 Enforce Incompressibility

The incompressibility solution is a two step process, in which divergence is calculated based on the new velocity, and then an artificial pressure term is iteratively solved for using a Jacobi solver, using the previous time step's artificial pressure as an initial guess to accelerate convergence. The final velocities are corrected with the negative gradient of artificial pressure to be approximately divergence free. We follow exactly the procedure defined in [Harris 2003], and refer readers there for implementation details.

5 Fire Volume Rendering

After each slice has been independently computed throughout the simulation frame range and the results saved on disk for every frame of every slice, volume rendering is performed using a simple GPU-based renderer. Each of the slices is already camera oriented and already aligned to the frustum, so no projections need be set up. In fact, each slice, starting from the near clipping plane and iterating towards the far clipping plane, can be loaded, converted into light and opacity via a user-defined shader, and then discarded as the next slice is processed.

Converting the density, temperature, texture and fuel into light and opacity is, to some extent, a matter of aesthetic direction. In the examples in this paper, we calculate light from temperature using the Planck Blackbody Radiation function, after the temperature has been gained and biased by the user. We similarly compute opacity as an exponential function of density and the spacing between slices. The texture channel is selectively used to modulate temperature and density during this conversion, and we do not currently use the fuel information.

The velocity images are used in conjunction with the semi-lagrangian advection slab operation to distort the slice images for the purposes of creating motion-blur. We redraw the stack of slice images over and over again into an accumulation buffer, with different time settings each time. Because the rendering operation is very fast, this is not very expensive. Our render times for 2k images were typically about 30 seconds to load each of the slices from disk, and then between one and ten seconds to render and save.

Holdout mattes are incorporated during the near-to-far rendering traversal in order to allow the fire renders to be composited fully integrated into complex environments. The heat distortion effect seen in the rendered movies is created as a post process in a compositing application. We warp the the background outwards by a blurred version of the brightness of the fire render. This is a standard compositing technique for simulating heat distortion, and we have found it is indistinguishable from "correctly" volume tracing with refraction.

6 Examples

The images shown in 1 are the results of three different fire simulations of very different characteristics. Each was simulated with approximately 25,000 particles in the coarse grid simulation, for between 50 and 200 frames. None of the coarse grid simulations took more than 10 minutes to execute and save to disk. Each of the simulations was run with a refinement slice resolution of 2048×1556, and the examples were run with 32, 64, and 128 slices, respectively.

We used a farm of NVidia Quadro 5600 GPUs running from 2.2 GigaHertz AMD Dual-Core Opteron machines running a modified build of SUSE Linux, version 9.3.1. Each frame of each slice took, on average, 20 seconds to complete and save to disk across a busy production network. Without saving the output files, the simulation took an average of 10 seconds per frame, per slice. Distributed across 10 GPUs, the simulations generally took between 2 to 4 hours to complete, with renders taking an additional hour to two hours.

These three simulations differ mostly in the direction of the input particle systems and the settings for exposure and density gain in the final rendering stage. This is a testament to the ease of directability of the tool. We have found that the system is very usable with a default set of control parameters listed in the equations in this paper. Though we added velocity damping to our system to control excessive velocities, in practice we did not find we frequently needed to use it, hence the setting of K_{damp} to zero in these examples. It remains in our implementation for cases in which very rapid movement may require extra damping.

Parameters	Fireball	Campfire	FireWall
T_{fuel}	1700	1700	1700
C_{cool}	3000	3000	3000
$K_{density}$	20.0	20.0	20.0
K_{damp}	0.0	0.0	0.0
K_{vort}	40.5	150.0	0.135
K_{buoy}	0.1	0.15	0.1
γ_{mass}	0.08	0.25	0.25
γ_{fuel}	1.0	1.0	1.0

7 Conclusions and Future Work

Our system has been employed on three feature films so far, and we have been able to direct the fire to perform in a wide variety of ways, from fireballs to walls of angry flame. The artist learning curve has been very short, and our turnaround times for gigantic, high-resolution simulations have been on the order of hours rather than days.

One limitation of our current system is that collision objects are not explicitly handled during the refinement stage. Instead, collisions are handled during the coarse grid step as a standard part of a traditional particle simulation workflow. Because the refinement simulation is so strongly sculpted by the input particles, simply handling collisions during the particle simulation step is often sufficient. However, a collision boundary condition can be explicitly created during refinement through the creation of "impulse only" particles. These are input particles which have their mass and fuel attributes set to zero. They contribute a velocity force to the refinement simulation but do not create heat or smoke. By emitting these impulse-only particles densely from the surface of collision objects and adhering them there, interacting objects are translated into the refinement simulation effectively. These particles create a "full-slip" boundary with friction near the collision interface. We have found this to be easy to control. The system could be enhanced so that collision objects are rasterized onto the refinement planes along with the input particles.

An area of future improvement is the system's handling of rapidly moving cameras. Camera movement is currently incorporated by simply updating the camera projection through which the particles are projected at each time step. This means that the linear and angular momenta of the moving camera are inherited by the refined fire. For slow-moving cameras, this is not visible and is not a problem. However, for rapidly moving cameras it does present a problem. In our productions so far, we have worked around this problem by rendering from locked-off cameras and then projecting onto geometry in the scene, which is then rendered from the moving camera. This is a tedious and imperfect workaround.

An alternative solution, which we are interested in exploring further, is to incorporate a compensation for the camera's movement into the advection velocity during the Semi-Lagrangian Advection step. For all but the most wildly moving cameras, this would allow the physical state of the refined fire to be correctly transferred

from one frame of reference to the next. There would still be a lack of transfer between the slices, but we think this will significantly improve the moving-camera problem.

Finally, as GPU storage and speed rapidly increase, we will soon be able to perform the entire simulation with all the slices on a single machine. This will allow for a more complete communication between slices, while still maintaining the many aesthetic advantages of the view-oriented detail that characterizes our technique.

8 Acknowledgements

This work could not have been completed without the support and guidance of Tim Alexander and Robert Weaver. Michael Jamieson and Lindy DeQuattro bravely endured using the tool in production while it was still under construction, and the quality of the resulting images is largely due to their talent and dedication.

References

BRIDSON, R., HOURIHAN, J., AND NORDENSTAM, M. 2007. Curl-noise for procedural fluid flow. *ACM Trans. Graph. (SIGGRAPH Proc.) 26*, 1.

CHUI, C. K. 1992. *An Introduction to Wavelets*. Academic Press Professional, Inc.

FEDKIW, R., STAM, J., AND JENSEN, H. 2001. Visual simulation of smoke. In *Proc. of ACM SIGGRAPH 2001*, 15–22.

FELDMAN, B., O'BRIEN, J., AND ARIKAN, O. 2003. Animating suspended particle explosions. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 3, 708–715.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proc. of ACM SIGGRAPH 2001*, 23–30.

FOSTER, N., AND METAXAS, D. 1997. Controlling fluid animation. In *Comput. Graph. Int.*, 178–188.

GOLUB, G., AND LOAN, C. 1989. *Matrix Computations*. The John Hopkins University Press.

HARRIS, M. J. 2003. *Real-Time Cloud Simulation and Rendering*. PhD thesis, University of North Carolina at Chapel Hill.

HONG, J.-M., SHINAR, T., AND FEDKIW, R. 2007. Wrinkled flames and cellular patterns. *ACM Trans. Graph. (SIGGRAPH Proc.) 26*, 3, 1188–1198.

LAMORLETTE, A., AND FOSTER, N. 2002. Structural modeling of flames for a production environment. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 3, 729–735.

NEFF, M., AND FIUME, E. 1999. A visual model for blast waves and fracture. In *Proc. of Graph. Interface 1999*, 193–202.

NGUYEN, D., FEDKIW, R., AND JENSEN, H. 2002. Physically based modeling and animation of fire. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 721–728.

NISHITA, T., AND DOBASHI, Y. 2001. Modeling and rendering of various natural phenomena consisting of particles. In *Proc. of Comput. Graph. Int. 2001*, 149–156.

OLANO, M. 2005. Modified noise for evaluation on graphics hardware. In *Graphics Hardware (2005)*, 105–110.

RASMUSSEN, N., NGUYEN, D., GEIGER, W., AND FEDKIW, R. 2003. Smoke simulation for large scale phenomena. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 703–707.

REEVES, W. 1983. Particle systems - a technique for modeling a class of fuzzy objects. In *Comput. Graph. (Proc. of SIGGRAPH 83)*, vol. 17, 359–376.

SELLE, A., RASMUSSEN, N., AND FEDKIW, R. 2005. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph. (SIGGRAPH Proc.) 24*, 3, 910–914.

STAM, J., AND FIUME, E. 1995. Depicting fire and other gaseous phenomena using diffusion process. In *Proc. of SIGGRAPH 1995*, 129–136.

STAM, J. 1999. Stable fluids. In *Proc. of SIGGRAPH 99*, 121–128.

STEINHOFF, J., AND UNDERHILL, D. 1994. Modification of the Euler equations for "vorticity confinement": Application to the computation of interacting vortex rings. *Phys. of Fluids 6*, 8, 2738–2744.

YNGVE, G., O'BRIEN, J., AND HODGINS, J. 2000. Animating explosions. In *Proc. SIGGRAPH 2000*, vol. 19, 29–36.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph. (SIGGRAPH Proc.) 24*, 3, 965–972.