INTERACTIVE SIMULATION OF FIRE, BURN AND DECOMPOSITION

A Dissertation

by

ZEKI MELEK

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2007

Major Subject: Computer Science

INTERACTIVE SIMULATION OF FIRE, BURN AND DECOMPOSITION

A Dissertation

by

ZEKI MELEK

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	John Keyser
Committee Members,	Ergun Akleman
	Nancy Amato
	Ricardo Gutierrez-Osuna
	Donald H. House
Head of Department,	Valerie E. Taylor

December 2007

Major Subject: Computer Science

ABSTRACT

Interactive Simulation of Fire,
Burn and Decomposition. (December 2007)
Zeki Melek, B.S., Bogazici University;
M.S., Bogazici University
Chair of Advisory Committee: Dr. John Keyser

This work presents an approach to effectively integrate into one unified modular fire simulation framework the major processes related to fire, namely: a burning process, chemical combustion, heat distribution, decomposition and deformation of burning solids, and rigid body simulation of the residue. Simulators for every stage are described, and the modular structure enables switching to different simulators if more accuracy or more interactivity is desired. A "Stable Fluids" based three gas system is used to model the combustion process, and the heat generated during the combustion is used to drive the flow of the hot air. Objects, if exposed to enough heat, ignite and start burning. The decomposition of the burning object is modeled as a level set method, driven by the pyrolysis process, where the burning object releases combustible gases. Secondary deformation effects, such as bending burning matches and crumpling burning paper, are modeled as a proxy based deformation.

Physically based simulation, done at interactive rates, enables the user to efficiently test different setups, as well as interact and change the conditions during the simulation. The graphics card is used to generate additional frames for real-time visualization.

This work further proposes a method for controlling and directing high resolution simulations. An interactive coarse resolution simulation is provided to the user as a

"preview" to control and achieve the desired simulation behavior. A higher resolution "final" simulation that creates all the fine scale behavior is matched to the preview simulation such that the preview and final simulations behave in a similar manner.

In this dissertation, we highlighted a gap within the CG community for the simulation of fire. There has not previously been a physically based yet interactive simulation for fire. This dissertation describes a unified simulation framework for physically based simulation of fire and burning. Our results show that our implementation can model fire, objects catching fire, burning objects, decomposition of burning objects, and additional secondary deformations. The results are plausible even at interactive frame rates, and controllable.

ACKNOWLEDGMENTS

I had the perfect opportunity to work with John Keyser, the best advisor one could wish for these long years doing research. I would like to thank Ergun Akleman for introducing me to Texas A&M and for our discussions. I would also like to thank the rest of my committee. All of them contributed ideas, directions, and improvements for my dissertation.

I would like to thank Lale Akarun, my advisor during my M.S., for introducing me to computer graphics and motivating me to do research.

I had the luck of working with two brilliant officemates over the years: Derek Overby, who helped me during the first years of my research, and Cem Yuksel, who helped me and kept me motivated during the final struggle to finish this research. I would like to thank Didem for her support and help in keeping me partially sane.

NOMENCLATURE

α_a	dissipation rate for gas a (could be combustible fuel gas or smoke)
$\cdot$	dot product
Δt	Timestep of the simulation
Δx	Grid size
∇	partial differential operator $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$
∇^2	second order partial differential operator $\nabla \cdot \nabla = (\frac{\partial^2}{\partial x^2}, \frac{\partial^2}{\partial y^2}, \frac{\partial^2}{\partial z^2})$
$\overline{v}'(\vec{x}, t)$	Adjusted final data at a time t, and a position $\vec{x}$
$\overline{v}(\vec{x}, t)$	Final data at a time t, and a position $\vec{x}$
ϕ	Implicit distance field (level set), divergence
ρ	density
$\times$	vector cross product
υ	kinematic viscosity of the fluid
$\vec{\nu}$	Velocity field of a moving interface
$\vec{\nu}_N$	Normal component of the velocity field of a moving interface
$\vec{\omega}$	vorticity
$\vec{D}$	Direction of decomposition
$\vec{f}$	external forces

$\vec{u}$	velocity of the fluid
$\vec{f_w}$	Vorticity confinement force
a_f	Combustible fuel gas
a_s	Exhaust gas, smoke
b	Stoichiometric mixture rate
$BC_{i,j,k}$	One if the cell i,j,k is a filled solid boundary cell
C	Combustion amount
d	Amount of decomposition
d_f	density of fuel gas
d_o	Oxygen density
d_s	density of smoke
$d^r_{i,j}$	Keyframed variable at time t_j, at position $\vec{x_i}$, and at scale r
f_g	Gravity force
f_T	Buoyancy force
G_r	Gaussian weight function with radius r
k_1	Strength of decomposition
k_2	Rate of fuel gas release
k_E	Diffusion constant for the solid-air interface
k_S	Diffusion constant for the solid

K_T	diffusion constant
$m(\vec{x}, t)$	Matching function
o_p	Number of octaves
p	pressure
$P(\vec{x})$	Turbulence noise at $\vec{x}$
r	Rate of burn ($0 < r \leq 1$)
r_s	Conversion rate from solid fuel to solid gas
S_a	Source for gas a (could be combustible fuel gas or smoke)
$S_{turbulance}$	Scale of turbulence
T	Heat
T_0	Output heat from the combustion reaction
T_{amb}	Room temperature
$T_{pyrolysis}$	Self-pyrolysis threshold for a solid
T_{sp}	Solid pyrolysis threshold (how volatile the solid is)
T_{thres}	Lower flammability temperature where burning can occur
V	Amount of solid fuel
$v(\vec{x}, t)$	Preview data at a time t, and a position $\vec{x}$

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

In the push for greater realism in computer graphics applications, complex physical simulations are playing a larger and larger role. From real-life experience, users are often familiar with the physical phenomena being simulated, and different simulations have varying success at replicating the details of a physical process. Such details can make the difference between a believable virtual world that draws the user in and a jarring environment that destroys any sense of presence.

Natural phenomena such as water, smoke and fire have been studied for centuries. Mathematical equations have been derived to define and predict their behavior. Physically based simulations are used to investigate their behavior in time using these equations, given an initial state and boundary conditions. As more computational power becomes available, these scientific simulations are used for increasingly more complex and accurate simulations.

There has recently been an increasing interest within the computer graphics community for simulation and visualization of natural phenomena. These visual simulations play an increasingly important role as a part of the story within the motion picture and game industries, and can be important cues in applications such as emergency personnel training. Generally, these applications require visually compelling but not strictly accurate models. However, the models need to be capable of capturing the visual characteristics of the phenomenon, which is usually not addressed in scientific simulations. Accuracy is defined as how well the simulation matches with the observation. In the case of visual simulation, we would like to match what we

---

This dissertation follows the style of *IEEE Transactions on Visualization and Computer Graphics*.

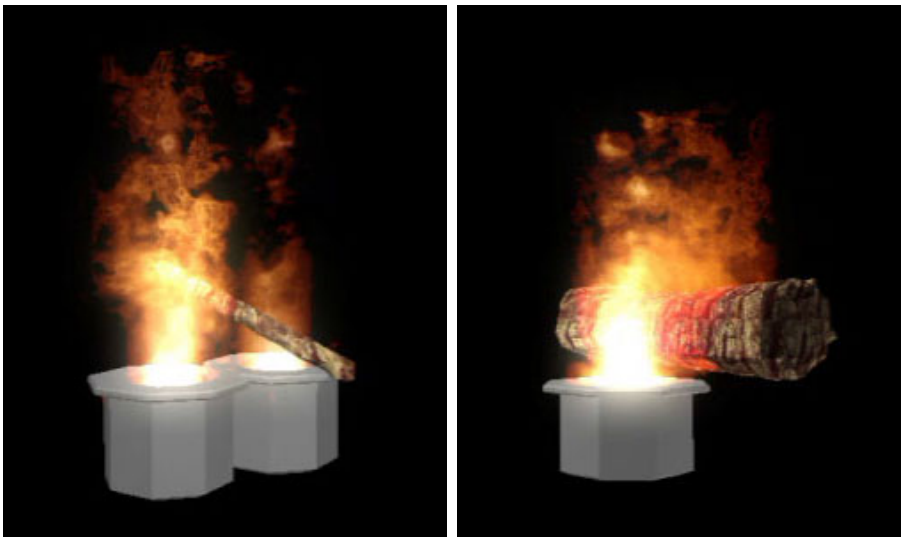

Fig. 1. Burning a match and a log.

see. To summarize, visual plausibility is the key requirement for visual simulations (Fig. 1).

Different phenomena might require completely different simulation models that are incompatible with each other. Combining different models and integrating them to work together is a hard problem (Fig. 2). Expanding current simulation systems to include additional models is not an easy task either. For these reasons, current simulation systems have only focused on parts of the actual phenomena. If we look at state of the art fire simulation research, there are different models for the combustion reaction, fire spread (different models at different scales, such as fire spread on a house vs forest), heat generated from the fire, and visualization.

Visual simulations are in some cases required to be interactive and easy to control/choreograph. Interactivity and accuracy are usually counter to each other in simulating natural phenomena because of the complex physics involved (accuracy and performance trade-off). Interactive methods are usually very simplified/adhoc systems compared to the accurate solutions, and they are not visually complex enough

to capture all the fine scale visual detail we encounter in natural events. As we stated earlier, scientific simulations do not necessarily make good visual simulations, and they are definitely not practical.

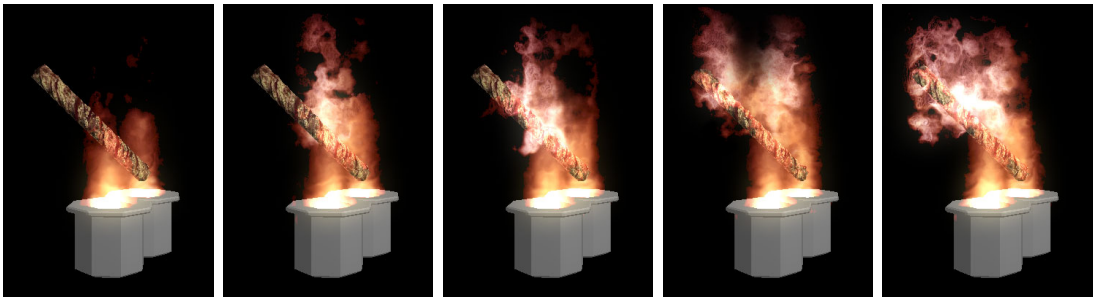

Fig. 2. Everything happening together. Flame, solid object catching fire, starting to burn and decomposing all happening together.

A. Simulating Natural Phenomena in Computer Graphics

Early solutions for simulating natural phenomena in computer graphics (CG) include randomized functions, fractals, and particle based solutions that generate plausible but not physically based results. On the other hand, the scientific simulation literature, especially computational fluid dynamic (CFD) literature, has generated many methods to create more and more accurate results, but with a huge computational complexity/cost. Recently, a number of advances have made realistic physically based modeling of smoke and other gaseous phenomena practical for CG requirements. These coarse-grid fluid-dynamic equation solvers are capable of capturing fine scale swirling motion of the air, with a very small computational cost.

Just as in simulation, complexity vs cost is an issue in visualization of the detail-rich behavior of natural phenomena such as fire. Very realistic results are achieved using photon map solutions combined with blackbody radiation, but the interactive results are far from convincing, so far. One exception we should note are computer

games, but often they achieve a realistic look by using adhoc nonphysically based techniques, prerendered/offline simulated images, or even recorded and preprocessed video.

B. Interaction and Control

Controlling the simulation behavior is a hard problem. Traditionally in a physical simulation, this is achieved by setting the starting conditions. On a complex and lengthy simulation, this might be difficult, inefficient or even impossible to do, because of the sheer number of parameters controlling this complex physical phenomenon. One can also process the results of the simulation, than adjust the simulation at one desired editing point, and continue to simulate from the point of adjustment again. This refinement process could be lengthy and unpredictable since the desired end result is not guaranteed.

A general simulation framework, capable of handling various conditions accurately yet responsive enough to be used with limited preprocessing or stored data, can go beyond the boundaries of classical systems. In an application such as firemen training within the highly dynamic environment of a building fire, results of the decisions of the trainee should be immediately evaluated by the training application. Another case we should look at is artistic/creative applications, where the user can choreograph using simple tools that control the simulation behavior.

A desired solution would be to provide a *preview* simulation running interactively, modeling only the basic simulation behavior plausibly. This structure will enable the user to access and control the simulation behavior immediately and get the desired behavior or *starting* point quickly. Later, a more accurate fine scale simulation could be used for more accurate or pleasing results guaranteed to have similar

gross behavior.

C. Hypothesis

The hypothesis of this dissertation is as follows:

> *One can simulate different aspects of the burning process in a unified physically based framework by taking advantage of some simplifying assumptions and achieve visually plausible results at interactive rates on current hardware. More accurate simulations could be guided to have global behavior similar to the interactive results.*

Here is what I mean by:

- *visually plausible:* an accurate enough result, which looks as though it could happen for the given conditions.
- *burning simulation:* physically-based, and accounting for all the parts of the burning phenomenon, such as ignition, decomposition, bending and crumpling.
- *unified simulation framework:* all in one package.
- *interactive:* speeds fast enough on current hardware for users to choreograph by changing parameters and conditions and get the results of these changes immediately.
- *guide:* maintain the same global behavior.

D. Contributions

The objective of this work is to effectively integrate the major processes related to fire, namely a burning process, chemical combustion, heat distribution, decomposition and

deformation of burning solids, and rigid body simulation of the residue into one unified modular fire simulation framework. This has not been achieved before. Simulators for every stage will be provided, and the modular structure will enable switching to different simulators if more accuracy or more interactivity is desired. Although this work will not try to propose the best solution for every component, we will present suitable solutions for every aspect of the phenomena discussed above, and demonstrate the suitability of the simulation framework in addressing them.

This work tries to bridge the gap between interactive methods and physically based simulations, accurately simulating major aspects of the burning process from a physical basis, and creating visually convincing images, at interactive rates on current hardware.

Physically based simulation, performed at interactive rates, will enable the user to efficiently test different setups, as well as interact and change the conditions during the simulation. The artist, on the other hand, can use the exact same tools to choreograph a scene. An interactive coarse resolution simulation is provided to the user as a preview to control and achieve the desired simulation behavior. A higher resolution final simulation that provides the fine scale behavior is *matched* to the preview simulation such that the preview and final simulations behave in a similar manner.

Here one should note that our work is aimed at computer graphics. Most of the methods discussed in the next chapters converge to the actual solution in the limit case (where the grid sizes and time steps go to zero), but that would not happen in practice, and is not even necessary for visually acceptable results. Unlike scientific simulation, results that do not necessarily match theoretical or experimental data exactly are perfectly acceptable in computer graphics. Algorithm simplicity, unified physically based simulation, and the interactive user experience are the major aspects

of this work.

This dissertation will address

- *A unified simulation framework* addressing many different phenomena related to combustion and the burning process:
  - *Combustion process*
  - *Hot air motion*
  - *Solid objects interacting with flames/air motion*
  - *Heat distribution and exchange*
  - *Solid objects catching fire (even when not even touching the flame)*
  - *Flame spread on burning objects*
  - *Decomposition of burning objects*
  - *Deformation of burning objects (bending)*
  - *Burning objects breaking into pieces*
- *Interactive simulation and visualization.* A coarse resolution simulation and tools to interact with the simulation are described.
- *Simulation control.* Interactive simulation is used as preview to guide a higher resolution simulation that generates finer scale detail.

E. Overview

The rest of the dissertation is organized as follows. Chapter II addresses simulation of fire in computer graphics (CG). We will go over the background work, and describe our three-gas flame model. A brief introduction of fluid dynamics and its application in

our framework is explained. In Chapter III we propose a new model for simulation of burning and decomposing solid objects. We also propose a simple yet efficient model to approximate bending/crumpling-like behavior. Integration with the flame model is explained in Chapter IV. In Chapter V we address interactivity issues. We will propose preview-based simulation control, and will go over the details of the matching process. In Chapter VI we will discuss visualization issues, and provide graphics processing unit (GPU) based inbetweening for real-time visualization. Chapter VII will summarize our results and address future work.

CHAPTER II

SIMULATION OF FIRE

Fire creates a unique problem compared to other natural phenomena. It has a very complex visual signature, both in a still image and video. To accurately capture its properties, one should generate its visual properties, as well as its very complex motion. Early models used a simplified approach, generating fractal textures and particles to create complex visuals and motion. Later models included physically based properties.

The very earliest fire models used particles and textured or warped blobs to generate flames. This technique, combined with some simple physically based air motion, is widely used in the motion picture industry and games, because of its ease of controllability. On the other hand, to achieve realistic or plausible fire behavior a large amount of initialization time is required by an artist/technician, and seemingly simple changes might require large amounts of labor. These models are still in use in the games (interactivity) and entertainment (visual quality) industries. One should note that, in both applications, the motion and behavior is predefined.

Physically based modeling of fire has been an ongoing research area, and many algorithms are available in the computational fluid dynamics (CFD) literature. Direct implementation of these algorithms requires relatively high resolution grid sizes to be accurate or even -in some cases- to be valid. The high computational cost and large storage requirements makes these algorithms impractical for CG applications. Hybrid approaches using partially physically based data have been favored in CG applications. Usually these use some simplified flame representation (flame skeleton, geometric primitive, etc) and complement it with physically based motion of the air.

A. Background

1. Fluids in CG

Before we investigate previous approaches to fluid dynamic simulations in CG, we need to understand Lagrangian and Eulerian approaches.

The Lagrangian approach defines points in space and labels them as separate particles. Properties such as velocity, density or pressure are defined at these particles and are advected with the moving particles. The Eulerian approach on the other hand defines fixed points in space and tracks changes of the properties. Both methods have their strengths and weaknesses.

The Lagrangian approach is basically a particle system, which is trivial to implement. Particle motions could be computed very fast. To increase detail in parts of the system, one only needs to add more particles at the desired location. On the other hand, spatial derivatives are not easily defined; since one requires spatial relations between particles. Other difficulties include finding nearest neighbors or enforcing incompressibility. Particle systems are very popular in CG and they are used extensively in the game and movie industries.

The Eulerian approach defines points (vertices)in space, and the connectivity of these points (mesh). The simplest Eulerian method uses a fixed regular grid, but unstructured meshes have been used as well. Since we know the connectivity, spatial derivatives are easily defined at any point. Interpolation is required in between points (such as during advection), which causes dissipation/smoothing of the data. The scale the system can resolve depends on the size of the grid. Spatially adaptive methods such as octrees can be used, but this makes spatial derivatives harder to compute. Eulerian methods are used extensively in scientific simulations but their memory requirements have limited their use in the CG community until recently. With the

increase of computational power and memory on common desktops, as well as with the recent advancements of coarse-grid fluid solutions such as "Stable Fluids" and with the introduction of level set methods, Eulerian methods have gained immerse popularity in CG community.

Recently, increased interest is in combining two methods to take advantage of best of the two worlds. Particle systems are gaining popularity with Smoothed Particle Hydrodynamic (SPH) methods combining an Eulerian phase to reconfigure the particles [96]. Eulerian approaches on the other hand are using semi-Lagrangian schemes to speed up calculations and use different methods to minimize the effects of the inherent smoothing. It is an exciting time for fluid simulations in CG.

In the earliest Eulerian work in CG, Kajiya and Von Herzen [56] worked on very coarse grids, using the very limited computer power they had at that time. Foster and Metaxas [35][37] reintroduced the method to the CG community, and were able to capture nice swirling motions using relatively coarse grids. Their model used an explicit integration scheme, bounding their time step in order to keep the simulation stable. They also used a costly relaxation step to ensure mass conservation.

Stam [108][109] used a semi-Lagrangian advection scheme and implicit solver to make the simulation unconditionally stable; even large time steps are allowed in his "Stable Fluids" model. He also used pressure-Poisson equations to conserve mass. Because he used a first order integration scheme, these simulations suffered from numerical dissipation. This model can handle boundaries inside the computational domain.

Fedkiw at al. [28] included vorticity confinement in the "Stable Fluids" model, keeping the smoke "alive" and increasing the quality of the flow with a very small amount of additional computational cost. Note that we included a practical introduction to the "Stable Fluids" method in Appendix A.

Simulation of liquids require a free surface tracking method. Foster and Fedkiw [34] introduced the level set method, and Enright et al. [25] used a particle level set method to correct for the volume loss due to the interpolation. Bargteil et al. [4] formulated the surface tracking as a contouring problem to avoid the volume loss. Losasso et al. [67] extended the particle level set method to as many regions as desired, and proposed techniques for simulating interactions between them, such as surface tension forces, complex chemical reactions, or one or two materials converting into another one. They demonstrated liquids mixing, reacting, and releasing combustible gas that then burns in one single simulation framework.

To overcome the large space requirements of Eulerian methods to simulate fluids with fine scale features, different methods are proposed, such as using octrees [65] or run length encoding (RLE) [49]. Shah et al. [105] take advantage of the Galilean Invariance to let the grid follow the important part of the fluid. Similarly, Rasmussen et al. [98] used whole grid increments to follow the fluid. To handle complex, deforming, or moving boundaries, Eulerian fluid simulation is generalized to tetrahedral meshes [30][31][58]. Irving et al. [52] combines two and three dimensional techniques for an efficient simulation.

Particle based methods have shown significant promise. Yoshida et al. [127] used smoke particles (puffs) to model the motion of the smoke and vorticity fields to take obstacles into account. Smoothed particle methods such as the smoothed particle hydrodynamics (SPH) [22] [78] and the moving particle semi-implicit method [96] have been proposed. Lagrangian methods require a large number of particles, and to address this issue, hybrid methods have been proposed [130]. Kim et al. [57] use marker particles for turbulent and splashing water. Instead of the fluid velocity, particles have been used to carry the vorticities and have been combined with a grid [40][103].

Recently, Yuksel et al. [128] proposed a new method for the real-time simulation of fluid surface waves and their interactions with floating objects using wave particles.

Other than incompressible fluid solutions, compressible fluid solutions have been proposed [126]. Goktekin et al. proposed a method for modeling viscoelastic fluids [42].

2. Combustion

Combustion is an ongoing area of research. Fuel is preheated by ignition, and the gases that are mixed with air (the oxidant) in "good" concentration start the combustion, giving an exothermic reaction. Besides water and carbon dioxide, the burning process results in other combustion byproducts, some of which might themselves be subject to further combustion.

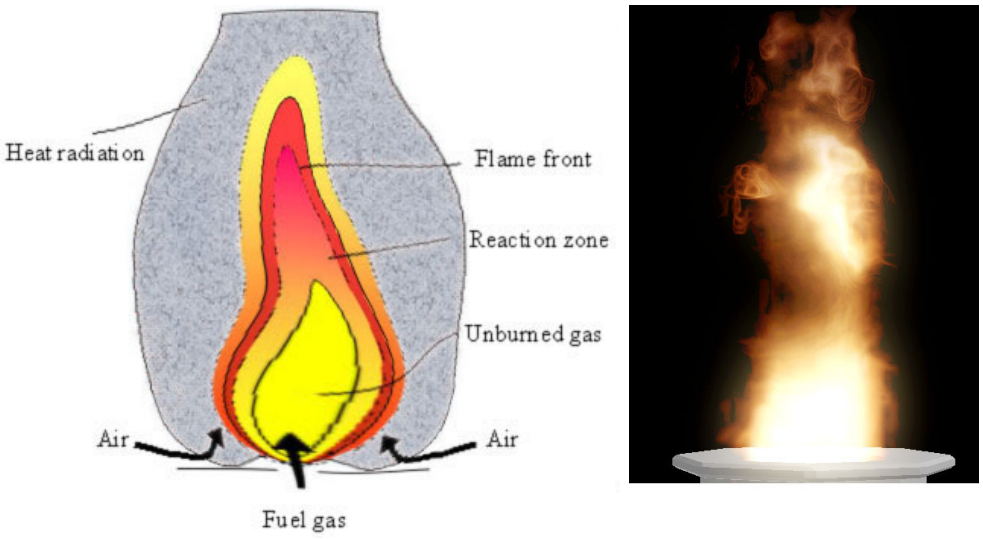

Fig. 3. Combustion process.

The fuel gas mixes with the oxidizing gas creating a mixture zone. The ratio of gas to air varies across the mixture zone (Fig. 3). The pure fuel is heated as it moves from its source at the center of the flame toward the reaction zone. Because there is no

oxygen in this case, the fuel is broken down into smaller molecules and radicals, which also forms soot (carbon). This process is very luminous and gives diffusion 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 outward throughout the reaction zone until the stoichiometric contour (flame front) is passed and the reaction is completed. The heat generated during the reaction keeps the reaction zone at the combustible temperature, triggering more combustion, thus creating more heat and combustion byproducts [23]. Diffusion (laminar) flames are created when "pure" fuel gas coming out of the fuel source mixes with the oxidizing gas through the mixture zone. Diffusion flames are the type commonly seen when an object burns. On the other hand, if fuel and oxidizing gases are well-mixed before entering the reaction zone, they create a premixed flame.

Once ignited, a diffusion flame stabilizes. Flames tend to be carried upward due to convection of the air that has been heated by the combustion process. Smoke, consisting of water vapor and other byproducts, is carried along with this column of air. The heat produced by the flame can also radiate to vaporize combustible products from nearby matter through chemical decomposition. This process is called pyrolysis [8]. Thus, when an object (such as a piece of wood) burns, the flames are formed from combustible portions of the object being vaporized and then oxidizing in the region just beyond the surface.

The color of a flame in the reaction zone is determined by the energy released in the exothermic oxidation process. Thus, a "pure" burning chemical may emit a single wavelength of light, while a more complex substance (e.g. wood) may emit at numerous wavelengths. In addition, the heat generated by combustion can cause combustion materials to emit light across a broad spectrum in the form of thermal radiation [8].

3. Flames in CG

Within the CG community, early models of fire include Reeves's particles [100] and Perlin's procedural noise function [90]. Inakage [51] uses a physical model to emit light in the regions of combustion for still images. Chiba et al. [18] and Takahashi et al. [117] both use a grid based representation of the space, and flames themselves are modeled by placing geometric primitives around particle trajectories. Some physical properties are linked into these models, but they are not physically based models per se. More or less the same models are still used in games or movies for interactive or visually stunning results respectively. Since they are not physically based, they are choreographed completely, and getting the desired behaviour/look depends on the skills/patience of the artist.

Recent physically based models include Lamorlette and Foster [60], combining particles with a flame skeleton, and Nguyen et al. [80], defining the flame front as a moving boundary between two fluids and using a level set method to capture very complex motion of the fire. The actual combustion process is not modeled, but an expansion term is introduced to add fine scale detail without using any randomized term. Another recent work by Feldman et al. [29] uses incompressible fluid equations to model suspended particle explosions. By enabling $\nabla \cdot u > 0$, they model an expansive flow without the need to simulate compressible fluids. Ihm et al. [50] used this result to model expansion of chemical reactions.

In the scientific simulation domain, NIST has developed the CFAST fire simulator [54], giving an accurate simulation of the impact of fire and its byproducts. Bukowski and Séquin [11] have integrated CFAST and the Berkeley Architectural Walkthru program, making the results of CFAST more understandable. This is an example of using scientific simulation for visualization. The visuals make investigation of

simulation results easier, but for CG purposes the visual results are too simple. A newer simulator implemented by NIST is the FDS [72], which has a very similar framework to what we propose to model combustion. Also, while CFAST and FDS are more accurate models, they are not suitable for interactive simulation.

The spread of fire has been another area of research. Perry and Picard [92] represent the flame front using particles, adding new particles as the front expands. Stam and Fiume [111] use a map to define the amount of fuel and temperature on every point on the object, and use turbulent wind fields and warped blobs. Beaudin et al. [7] use a similar but more accurate flame front technique to model the spread, and guarantee that the boundary always lies on the object. Wei et al. use the Lattice Boltzmann Model to simulate fire at interactive rates [125], and add visual detail by using textured splats. They recently [129] extended their model to track flame fronts over volumetric implicit solids. Jones [53] introduced a method that accurately models both thermal flow and the latent heat during the phase change to model melting solids. A CFD based physical simulation outlined in [21] treats combustion and turbulence independently. The turbulence is modeled using a multi-resolution turbulence model, and particles are used to track the flame front for the combustion reaction.

B. Flame Model Overview

The initial module of any fire simulation framework is a flame model. The two fluid boundary approach in [80] is exciting, yet compressing the reaction zone into a surface is not always realistic. It is also too computationally complex to be considered in an interactive application. We propose a simpler *three gas flame model*. Amounts of fuel, smoke and oxygen are defined on a coarse grid, and the reaction zone is defined

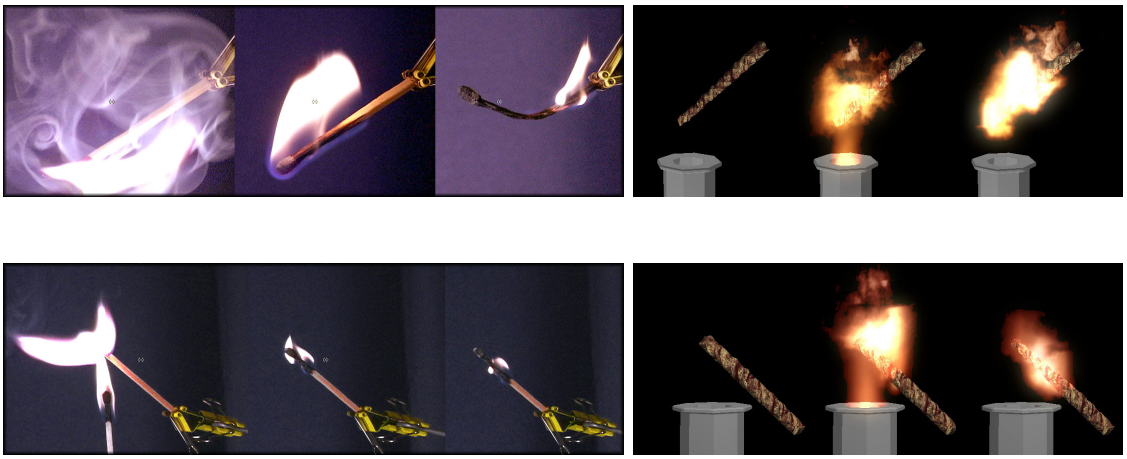

Fig. 4. Real vs simulated matches in different orientations.

by the conditions that satisfy the combustion reaction. This simple scheme enables us to model the chemical combustion process, as well as simulate the motion of the gases within the simulation region. Heat, coming out of the simulation, drives the combustion process forward, and causes air currents. Air motion is simulated as a single fluid, assuming the governing motion comes from the hot air currents. This simplification gives an accurate overall motion, but fails to incorporate as much small scale motion as the previous model can capture. We demonstrate a wide range of flame conditions, including ignition, self sustaining flames, various combustion reactions resulting in wider or narrower reaction zones, and self-extinguishing flames (Fig. 4). Note that we have published parts of the three gas flame model in [74].

C. Three Gas Simple Flame Model

Details of the "Stable Fluids" model could be found in Appendix A. In "Stable Fluids" a non-reactive substance such as smoke is carried with the motion of the fluid (Eq. A.6). We use the same vector field solver, and use the resulting vector field to advect and diffuse three quantities, fuel gas d_f, exhaust gas (including smoke) d_s, and

heat T.

$$\frac{\partial d_f}{\partial t} = -(u \cdot \nabla)d_f - \alpha_f d_f + S_f \tag{2.1}$$

$$\frac{\partial d_s}{\partial t} = -(u \cdot \nabla)d_s - \alpha_s d_s \tag{2.2}$$

$$\frac{\partial T}{\partial t} = -(u \cdot \nabla)T - \alpha_T T + K_T \nabla^2 T \tag{2.3}$$

where K_T is a diffusion constant, $alpha_a$ is the dissipation rate and S_a is a source term for gas a. Notice that the dissipation values can differ for the three quantities. Thus, the characteristics of the motion of the fuel, smoke, and temperature can differ significantly, although all are carried by the motion of the single 3-gas fluid system.

For fuel gases α_f is small if not zero, whereas smoke has a larger α_s. S_f is initially empty, except the initial source flame. During the simulation, as objects start burning, they too become sources. Smoke and temperature do not have source terms, since smoke and heat comes from the burning process, according to the amount of fuel and air consumed within one time step (Fig. 5).

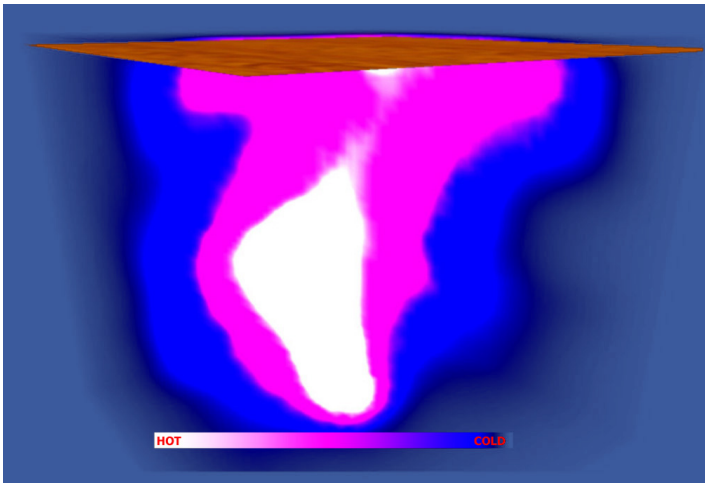

Fig. 5. Heat distribution around the flame.

Temperature has a large K_T which simulates diffusion of the heat and small α_T value, diffusing fast but not dissipating within the system. Diffusion of the temperature field is solved with a similar approach to that used in [108], using an implicit integration step, which gives a sparse linear system when discretized. Note that we have not used a diffusion term for the gas motion. The implicit solver, being unconditionally stable, comes with a cost; it introduces smoothing to the gas motion similar to diffusion.

The fluid system is modified by the presence of the fuel, exhaust gas, and temperature. Referring to Eq. A.2, the external force, f, acting in any one cell is given by:

$$f = f_g(d_f + d_s)\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} + f_T(T - T_{amb})\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.4}$$

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 smoke and fuel gas will tend to fall, though this effect is usually not very noticeable (i.e. f_g should be fairly small).

D. Combustion Reaction

The combustion (burning) in a cell is defined by:

$$C = rmin(d_o, bd_f) \;\; if\; T > T_{thres} \tag{2.5}$$

$$\frac{\partial d_f}{\partial t} = -\frac{C}{b} \tag{2.6}$$

$$\frac{\partial d_s}{\partial t} = C\left(1 + \frac{1}{b}\right) \tag{2.7}$$

$$\frac{\partial T}{\partial t} = T_0 C \tag{2.8}$$

where the oxygen density d_o is defined such that the total amount of gas in each cell is constant ($d_f + d_s + d_o = const$). Conceptually, four parameters are used to control the shape and the stability of the flame. r is the burning rate ($0 < r \leq 1$) defining the percentage of the fuel gas that can be burned in a second. T_{thres} is the lower flammability temperature where burning can occur, and T_0 is the output heat from the reaction. It can be sufficient to start a reaction in a neighboring cell, or it might not be sufficient, extinguishing the flame. b is the stoichiometric mixture, controlling the oxygen requirement of the combustion and used to model different reactions.

The fuel gas is ignited either by reaching a sufficient temperature, or by a special "ignition" step. This combustion increases the heat in the system and can provide enough heat to the neighboring cells to continue the burning process on the next time step. Smoke is produced as a by-product of the combustion, and tends to rise with the hot air produced by the flame.

E. Results

Our results show that we can simulate a variety of scene setups successfully. We can get plausible results with even not-so-suitable-to-our-model cases, such as explosion-like fast burning and smoke filling a closed room and extinguishing the burning flame (Fig. 6).

This simple yet flexible model described above is strong enough to handle various fire related phenomena and can be used as the underlying flame model in our

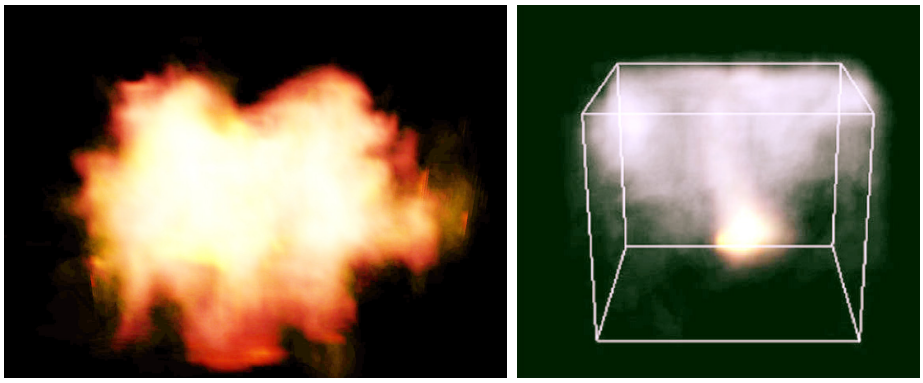

Fig. 6. Two extreme cases. Our fire simulator can handle these extreme cases: (a) an explosion like fast combustion process, (b) flame burning in a closed area uses all the oxygen and suffocates itself.

simulation framework. We will integrate this model into our simulation framework and demonstrate the flexibility of the model. We also will discuss how the fine scale motion lost in our simplification can be included during visualization in chapter VI.

Before we move to burning and decomposing objects, we will briefly summarize the simplification assumptions in our flame model. We will also list the parameters controlling the combustion reaction.

1. Simplifications and Analysis

Even though our framework for simulation is general, the individual modules we have proposed make a number of simplifying assumptions, which lead to some shortcomings. We describe below some of our key assumptions, simplifications, and shortcomings, along with the reason we accept them.

- *Single fluid model.* Although air, fuel gas, and exhaust gases will have different properties, we choose to model the system with a single moving fluid. We therefore do not capture properties such as the mixing of fluids of different viscosity, which hurts our ability to capture some fine-scale detail. However, our

model still captures the larger-scale motion of heated air, and is significantly cheaper computationally. We also believe that the fine detail is more of a rendering issue than a key to simulation, and such details can be better added as a visual postprocess.

- *Constant density air.* Our assumption that the sum of the air, fuel, and exhaust gas in a cell is constant is somewhat unrealistic. In particular, the effects of combustion, such as the pressure generated and the resulting density changes are not directly modeled (we do simulate some effects via the buoyancy forces). This simplification, together with the previous one, simplifies the equations for fluid flow. Although our framework is general, the sample implementation presented here is aimed for an interactive and responsive simulation.

- *Expansion of the reaction zone.* One specific property of the combustion reaction is expansion of the gas, which gives a more round shape to the flames. This could be added to our model by introducing outward forces at the reaction zone, similar to Nguyen et al. [80], or setting $\nabla \cdot u$ to be larger than zero at the reaction zone, similar to Feldman et al. [29].

- *Coarse resolution simulation for interactivity.* One might argue the need for a low resolution physically based simulation, since there are some production quality simulation models around ([80]). These other methods, though, are non-interactive, miss key parts of the entire burning process, and are difficult to fine-tune. We present a framework to model many aspects of fire related phenomena together in one single simulation framework, and the coarse simulation used here is *one* of the possible implementations. Our particular implementation runs at interactive rates, enabling the user to fine tune the simulation behavior quickly, giving plausible results. As mentioned below, having the interactive method

can allow us to more easily control behavior in more complex simulations.

2. Controlling the Shape of the Flame

In order to simulate different burning reactions, it is important for an animator to have some control over the flame shape. The shape of the flame can be controlled by modifying a few key parameters:

- Turbulence controls the amount of additional random motion added to the fluid motion field. If zero, the flame will eventually stabilize in a fixed shape. Introducing turbulence causes the flame to have a more realistic motion (Fig. 7).

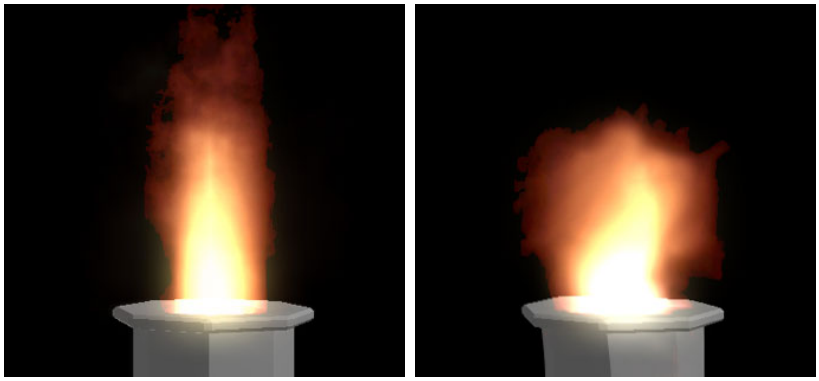

Fig. 7. With and without turbulence. The flame at left experiences no turbulence whereas the right flame does. The flame at left will remain in that shape, while the one on the right will change over time.

- Burn rate gives the amount of fuel that can be burned in one timestep, as a percentage of the maximum amount of fuel that can combust (i.e. is at an adequate temperature and is mixed with air appropriately). Large (=1.0) burning rates tend to create very thin and sharp flames, since fuel is usually consumed soon after mixing with air, while slower burning rates create larger and more blurry flames (Fig. 8).

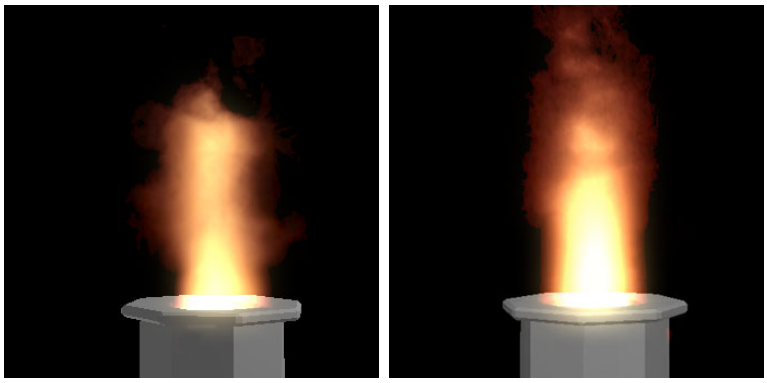

Fig. 8. Comparison of different burning rates. The flame at left has a faster burning rate than the one at right.

- Air use (i.e. the stoichiometric mixture) describes how much air is needed for the fuel to combust. This can be used to model different combustion reactions, requiring more or less oxygen.

  Generally, as air use increases, the flame size becomes much larger and more spread out (similar to a low burn rate), since the gas must spread and diffuse more in order to mix with enough air to combust (Fig. 9).

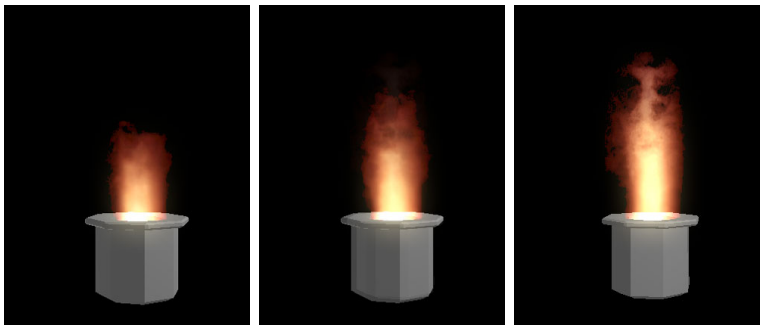

Fig. 9. Effect of air usage. From left to right, the combustion requires more and more air to burn the same amount of fuel gas.

- Output temperature controls the intensity of the combustion. Lower output temperatures can lead to flames that are not self-sustaining or that let some fuel go unburned. High output temperature can cause flames to spread rapidly,

ensuring full combustion and inducing self-ignition in other regions.

- Ignition threshold sets the temperature needed for fuel to combust. A lower ignition threshold is used to model highly flammable gases, while a higher ignition threshold can model almost nonflammable gases. Relating this to the pyrolysis temperature of any solids helps to simulate very flammable or nonflammable materials.

Although other parameters can be modified as well (e.g. the diffusion rate for temperature), these are more fundamental to the simulation and will tend to change the simulation as a whole, rather than just allowing simulation of different types of burning. They also tend to be less predictable in their effects.

Our examples and descriptions have focused on diffusion flames, where pure fuel is released that gradually mixes with air. These are the common flames seen when objects burn. We note, however, that our method can just as easily simulate premixed flames, where the air and fuel are mixed when introduced to the environment.

CHAPTER III

BURNING SOLIDS

Within the CG community, there have been only a small number of attempts to address burning objects. Burning solids have been modeled either as a simple boolean switch, burning/not-yet-burning, or a flame front sweeping a surface. In the former model, after the solid reaches a self-pyrolysis temperature, it turns the switch on and starts emitting fuel gas at a constant rate. In the latter model, the front is marked as actively burning, and the area swept is marked as burned. Many models have been proposed to track the sweeping flame front.

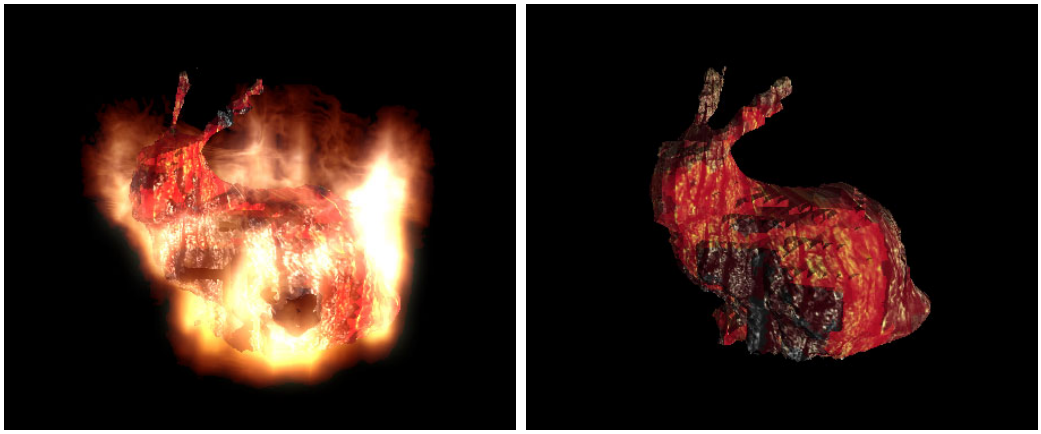

Fig. 10. Burning bunnies. OpenGL output from the interactive simulation.

A burning solid decomposes, and changes both geometrically and topologically, which makes this an interesting problem from a modeling point of view (Fig. 10). Both approaches mentioned above treat the solid as a nonchanging object, and lack the ability to capture the geometrical and topological changes a burning object goes through. The most closely related work has been on melting solids, which takes the change of the shapes into account.

In this chapter we will start with the related background work, including level

set methods and free-form deformations. Note that we provide a brief practical introduction to the level set methods in the appendix. Our simulation method for burning objects will be presented in three sections. First we will start with objects catching fire, namely the pyrolysis process. Than, we will present the decomposition of the burning objects, together with the discussion of our proposed multi-representation simulation framework. Finally we will discuss bending and crumpling behavior.

A. Background

In this section we will present related previous research on decomposition of objects. We will also briefly look at previous work on the level set methods and free-form deformation tools, since we will use them in our model (Fig. 11).

1. Previous Work on Breaking/Deforming Objects

Since our motivating case is deformations of burning objects, we will first look at related work in this area. Losasso et al. [66] models burning objects using remeshing, and can model fine scale decomposition structures. They present a novel technique for simulating phase change of solid objects into fluids, such as melting volumetric solids and burning thin sheets. However, this work does not addresses secondary deformation effects due to structural changes. Although this effect is minor for large objects, it creates a dominant deformation on small objects. Examples include the upward bending seen in burning matches and the crumpling of burning paper. The current state-of-the-art for creating such deformations involves an artist modeling the deformation manually.

There has been some work on deformations related to physical processes. One example is given by O'Brien et al. [83]; their work deals with fracturing solid objects.

Fig. 11. Decomposition in action.

Yngve et al. [126] combine shock waves together with the fracturing of solids. Carlson et al. [13] solve the melting problem by treating the solid as a liquid with high viscosity that changes in space and time, allowing the same material to exhibit different states. Modeling internal stresses on a moving viscoelastic fluid is presented by Goktekin et al. [42] using a level set method [104][85][27].

Other deformation work models imprints and tracks in the ground. Methods used for doing so include bump maps [68][69], heightfields [115], and elastic sheets [15]. Some methods also include models of soil dynamics [61][84]. Similarly, soft body deformations of colliding objects have been investigated [41][24][88]. Adaptively sampled Distance Fields (ADFs) have been used to detect collision of soft bodies [38]. The contact deformation is modeled using forces defined inside the overlapping regions.

2. Level Set Methods

Level set methods, first introduced in [86] are a simple yet powerful approach for computing moving interfaces. They have recently started to gain popularity within the computer graphics community. Many applications, including incompressible and compressible flow problems and the recent thin flame model [80], are solved using level set methods. More information on level set methods and their applications in computer graphics can be found in recent publications [104][85][27]. We will provide a brief introduction on level set methods in Appendix B.

3. Free-Form Deformation

Many different methods have been proposed for modeling clay-like deformations including Octrees [2], Freeform Deformations (FFDs) [20], Finite Element Models (FEMs) [122][39], Adaptively Sampled Distance Fields [9], and Level sets [3]. Different user interfaces have been presented to control the deformations [71][9][63]. Recently, vector fields have been used to control deformation on surfaces [121].

Free-Form Deformation (FFD) tools are widely used in soft-object animation and shape deformation. The points on the object are represented as an affine combination of the vertices of the encapsulating shape around the object. Barr [5] introduced a set of hierarchical transformations for deforming an object, including twist, bend and taper operations, using the position vector and surface normal of the undeformed object and a transformation matrix. Each level in the deformation hierarchy requires an additional matrix multiplication. Sederberg and Parry developed a technique for deforming the solid geometric models in a free form manner [87][102]. Their method could be applied to quadrics, CSG based models, parametric surface patches, or implicit objects with derivative continuity. The FFD is usually defined in terms of a tensor product trivariate Bernstein polynomial. Coefficients of the Bernstein polynomial are the control points. Coquillart presented an extension to this method, Extended FFD (EFFD) [20], adding arbitrarily shaped bumps or bending the object along an arbitrarily shaped curve using B-spline control points. Chang and Rockwood [16] deform objects using a Bezier curve and affine maps controlling handles.

Affine combinations of vertices encapsulating a shape form an interpolation scheme. Vertex positions could be used as data values in these interpolation functions to form deformations. Usually, these methods are used for deformations defined by a user. Many interpolation schemes have been proposed [70][59]. Warren et al.

[123][124] extend the Washpress interpolant for convex shapes into 3D. Ju et al. [55] use mean value coordinates [32] for 3D non-convex shapes. Similar results were produced concurrently by Floater et al. [33].

B. Burning Solids

We will start with a brief overview of the burning process. The heat produced by the flame can vaporize combustible products from nearby solid matter through chemical decomposition. This process is called pyrolysis [8]. Thus, when an object (such as a piece of wood) burns, the flames are formed from combustible portions of the object being vaporized and then oxidizing in the region just beyond the surface.

In our simulation model, solid materials inside the computational domain are voxelized and the corresponding grid cells in the fluid's computational domain are marked as filled. This filled/empty information is used in flow calculations [108][28]. In the advection step, the particle tracer hits and stops on filled grid cells, and in the diffusion step the filled grid cells are assigned the same amount of density as their empty neighbor, making the incoming and outgoing density flows equal. If a filled grid cell has more than one empty neighbor, average values are used, but it is possible for material one voxel thick to "leak".

Each filled grid cell is a potential fuel source. Active fuel sources emit fuel gas density into the neighboring unfilled cells at every time step. Potential fuel sources can later self-ignite, becoming active fuel sources. Every filled grid cell has a self-pyrolysis temperature threshold, which is a material property (nonflammable materials simply have an arbitrarily high pyrolysis temperature). When a filled grid cell reaches its pyrolysis temperature, it becomes a fuel source. Since the self ignition temperature of the fuel is generally much smaller than the pyrolysis temperature,

if there is enough air around, the resulting fuel starts burning once it mixes with air. Note that the combustion reaction occurs in the flame simulation module we presented in the previous chapter.

Since heat is driving the pyrolysis process, we will discuss the heat exchange between the solid and air first. Next, we will talk about the heat diffusion inside the solid object. Finally, we will present the pyrolysis process and how we model the combustible gas release.

Fig. 12. Burning logs during interactive simulation.

1. Heat Exchange

We model the heat transfer within the simulation framework in three stages: heat transfer in the air, heat transfer between the air and the solid, and heat conduction within solids. This three-stage heat transfer model enables us to treat solids with varying thermal properties (Fig. 12).

For the heat transfer between the solid and the air, we first find all solid-air boundary voxels in the distance field representation and mark them as boundary solid cells (BSC) or boundary air cells (BAC) depending whether they are inside or outside of the solid. We initialize the heat in the BACs from the flame simulation module by interpolation of the heat info from the air.

For each BSC, we exchange heat to/from the adjacent BACs, using the heat differential. Since the inter-object heat diffusion is processed in the next stage (Sec 2), we do not process the neighboring filled solid cells. For simplification, we only consider direct neighbors, 6 for the three dimensional case. We use the average heat differential by averaging the heat differential in three axes:

$$BC_{i,j,k} = \begin{cases} 1, & \text{if cell i,j,k is BAC} \\ 0, & \text{o.w.} \end{cases} \tag{3.1}$$

$$\frac{d}{dt}T^i_{i,j,k} = BC_{i+1,j,k}(T_{i+1,j,k} - T_{i,j,k}) + BC_{i-1,j,k}(T_{i-1,j,k} - T_{i,j,k}) \tag{3.2}$$

$$\frac{d}{dt}T^j_{i,j,k} = BC_{i,j+1,k}(T_{i,j+1,k} - T_{i,j,k}) + BC_{i,j-1,k}(T_{i,j-1,k} - T_{i,j,k}) \tag{3.3}$$

$$\frac{d}{dt}T^k_{i,j,k} = BC_{i,j,k+1}(T_{i,j,k+1} - T_{i,j,k}) + BC_{i,j,k-1}(T_{i,j,k-1} - T_{i,j,k}) \tag{3.4}$$

$$\frac{d}{dt}T_{i,j,k} = k_E(\frac{d}{dt}T^i_{i,j,k} + \frac{d}{dt}T^j_{i,j,k} + \frac{d}{dt}T^k_{i,j,k}) \tag{3.5}$$

Finally the change of heat in the boundary air cells is interpolated back to the heat information in the air.

2. Heat Diffusion

Heat transfer inside solids is modeled as a diffusion process using implicit integration. In this way, the heat transferred from outside the object as above spreads into the interior of the object being burned (This is similar to heat diffusion in the air as we discussed in the previous chapter II).

$$\frac{d}{dt}T = k_S\nabla^2 T \tag{3.6}$$

where k_S is a constant based on density, thermal conductivity, and specific heat of

the material. For most objects being burned, this heat diffusion is quite slow (k_S is small), and constant through the object. For nonuniform material, we can incorporate variations of thermal conductivity in our simulation by adding one more property, *thermal conductivity*, as a volumetric grid. The implicit solver defined in [108] and discussed in the previous chapter (Chapter II) is used to solve the diffusion equation above (Eq. 3.6), using a conjugate gradient method.

3. Pyrolysis

Once a filled solid cell reaches the self-pyrolysis temperature threshold $T_{pyrolysis}$, it is marked as a pyrolysis cell and a pyrolysis process is applied at every simulation time step until it runs out of solid fuel contained in the cell. This temperature can be set low for volatile solids, arbitrarily high for nonvolatile solids, or vary throughout the solid (if we add another property) to model mixed solids. Note that this information could also be generated from a texture map to allow variations across the object. During the pyrolysis step, some of the solid fuel is converted into fuel gas until it runs out of the predefined amount of solid fuel contained in the cell (V). Again, this is a material property.

When $T > T_{pyrolysis}$

$$\frac{\partial}{\partial t} d_f = r_s \frac{\partial}{\partial t} V \tag{3.7}$$

where V is the solid fuel amount, d_f is the density of fuel gas, and r_s is the conversion rate from solid fuel to fuel gas. Each cell is checked for whether it is in pyrolysis, and fuel gas is generated, consuming solid fuel, if so. We have an in-depth analysis of the amount of solid fuel converted in the next section.

Note that although there are a large number of parameters controlling the py-

rolysis process, the decomposition model is integrated into our interactive flame simulation framework, thus enabling the user to tweak the parameters for the desired simulation behavior easily and fast. Parts of this burning model have been published in [75].

C. Modeling Decomposition

The biggest problem for modeling decomposition of burning solids is how to represent the solid object (Fig. 13). It is easy to see that simple explicit boundary representations, such as polygons, are not suitable for this job: the object is undergoing radical topological changes, and could reach an invalid (non-manifold) object representation. Either a large amount of special case detection and handling is required, or volumetric representations should be used. There are multiple possibilities to represent a solid object, some more suitable in some cases, and not suitable in the others. For example, solid fuel contained inside the object represented volumetrically is suitable for modeling the fuel release during pyrolysis. On the other hand, a signed distance field representation of the object is suitable for modeling topological changes using level set methods, and easier to visualize.

Level set methods are usually used for highly changing boundaries, such as liquids, but have also been used to model changing boundaries of solid objects. A level set is a powerful method to track moving interfaces. We will start with the multi-representation framework used to model interactions between simulation properties. Later, we will present the overview of the simulation, and will go into details of each step of the simulation.

Fig. 13. Decomposition of a log. Flame is not shown for clarity.

1. Multirepresentation Framework for Physically Based Simulation

For simulations involving complex objects, a number of different properties must be represented. An object undergoing combustion is an example of this—heat amounts, fuel consumption, and even object shape must be modeled and changed over time. Ideally we would put everything into a unified representation, but this is sometimes not possible/feasible due to measurement limitations or the suitability of a specific representation.

Traditionally, material properties are modeled procedurally or volumetrically. While such a single-representation modeling approach may be effective for many applications, it also has various drawbacks. Also, within a simulation framework, it is not usually feasible to convert between representations at every time step, and such conversions often tend to lose accuracy through repeated conversion. An alternative approach is to keep multiple representations of the same object and use the appropriate one in different parts of the simulation. In these approaches, however, the different representations actually refer to the same object and do not contain

additional information about the object.

We have defined a model for supporting interactions between different properties of an object kept in different representations, as well as functions based on this interaction. Note the difference to the previous methods: we are proposing using different representations to define different properties, every representation is only a partial definition of the object, and all representations together define the object in the simulation (Fig. 14). This model is especially useful in physically based modeling, where the time variation of some properties affects other properties including geometry or topology.

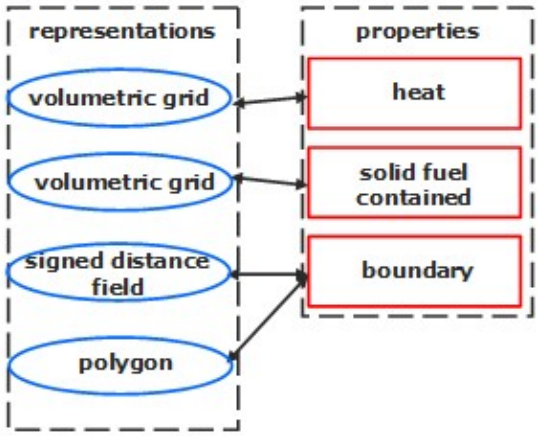

Fig. 14. Separation of properties from the representations.

A *multi-representation framework* basically puts each different property into whichever representation(s) best suits it. An example of such a dual representation in use is in Guendelman et al. [44], where a dual representation (one implicit and one boundary) is used for a rigid object. Within a simulation, the properties are accessed and modified, and some of these changes can affect the representation(s) used for that property. This framework allows us to separate the properties from their representations, and separate the main simulation modification and interaction among the

properties (Fig. 15). Doing this requires shifting the simulation design from the representation to the property itself. More details of our multi-representation framework could be found in [76].

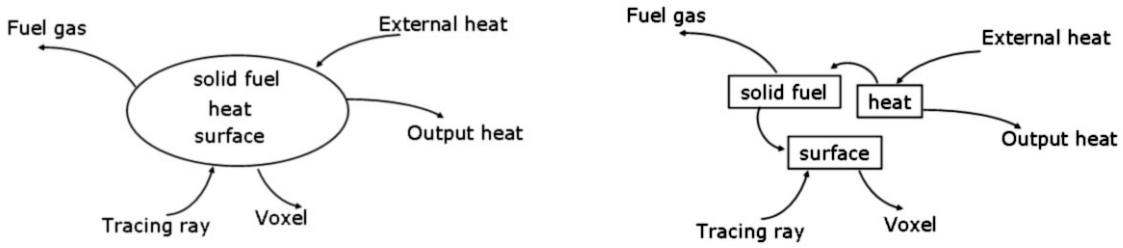

Fig. 15. Going from single representation to multirepresentation.

Since we heavily depend on data exchange between separate representations, all properties (and thus all representations) should support an interpolation interrogation, to get the value of any property at any point inside or on the boundary of the object. When one property shares more than one representation, both representations refer to the same thing, and thus we must ensure that all these representations are synchronized throughout the simulation.

2. Overview of the Simulation Structure

We structure the simulation around object properties, defining property manipulation functions (PMFs).

- *External* PMFs account for actions that take place outside/around the object during the simulation and that affect the object. Some examples of these are external heat, collisions, wind forces, magnetic fields, etc. Note that external PMFs do not necessarily change the object—they can model how the object affects others.

- *Internal* PMFs account for changes of some properties within the object itself; these can be (but do not have to be) secondary effects of external PMFs. Internal PMFs can be defined as chains of functions where changes in the values of one property of the object triggers changes in other properties (and are thus called *property interaction functions*) or other values of the same property. Examples of internal PMFs include heat propagation within a solid, force propagation through an object, and changes in object composition due to chemical or physical reaction.

The simulation structure using a multi-representation paradigm is as follows.

- First, the heat exchange to/from the outside is modeled by an external property manipulation function on the heat property.
- Second, internal heat diffusion is modeled as an internal PMF on the heat property itself.
- The new heat distribution is used to trigger pyrolysis from the volume representation of the solid fuel amount property and the released fuel gas is output to the fire simulation. The change in the solid fuel amount is modeled as an internal PMF.
- The release of fuel from the solid also has a secondary effect, in modifying the boundary property representation (internal PMF). This is how the decomposition of the object is achieved.
- For visualization, a temporary polygonal representation is created; this can be thought of as an internal PMF on the boundary property.

- The items must be able to undergo rigid body motion (a global property) within the simulation. Coupled with this, we need to perform collision detection on the objects. These are external PMFs.

- As objects burn, their topology can change, eventually causing them to split into two or more pieces. We must detect these changes and modify the object topology.

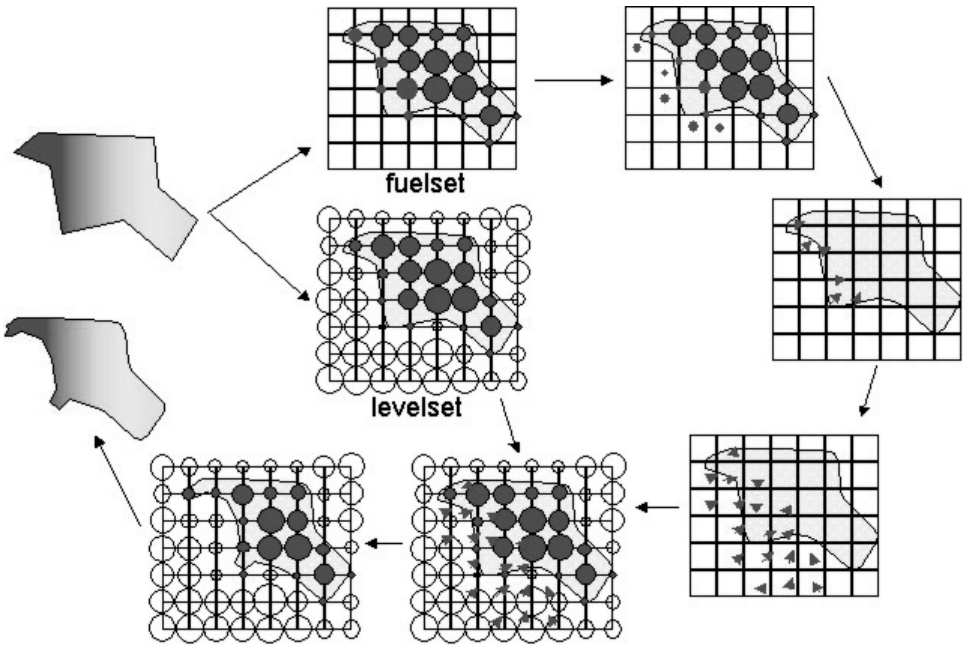

Fig. 16. Decomposition steps. Two representations of the object are used to define the decomposition. First a volumetric fuel set is used to track the material loss during pyrolysis, and a vector field is formed, defining the decomposition direction. The second representation is a distance field, which is advected using this vector field.

a. Decomposition

The decomposition of the burning solid is modeled as a moving boundary in the distance field representation of the solid. Level set methods, which we use to model

decomposition, are a simple yet powerful approach for computing moving interfaces [86][85]. The decomposing solid is defined implicitly as

$$\phi(\vec{x}, t) = 0 \tag{3.8}$$

We define the decomposition of the object with two components: d the amount of decomposition, and $\vec{D}$ the direction of decomposition.

$$\frac{\partial}{\partial t}\phi = d(\vec{x})\vec{D}(\vec{x}) \cdot \vec{N}(\vec{x}) \tag{3.9}$$

Defining d, the amount of decomposition, is trivial. We define it proportional to the amount of solid fuel released as fuel gas in a time step.

$$d(\vec{x}) = k_1 \frac{\partial}{\partial t} V(\vec{x}) \tag{3.10}$$

$$0 \leq k_1 \leq 1 \tag{3.11}$$

where V is the solid fuel representation and k_1 is a constant controlling the strength of decomposition. Note that k_1 represents the physical quantity of the ratio of residue (nonflammable material) vs the solid fuel (flammable material) and thus controls the amount of ash left.

Once the amount and direction of the decomposition is defined, semi-lagrangian levelset methods [113][114] are used to update the implicit distance field.

Defining $\vec{D}$, the direction, is dependent on how we define the pyrolysis process. We have implemented two versions, one with a constant pyrolysis process and one

with time varying pyrolysis. They are covered in the next section.

3. Defining Decomposition Direction

As stated earlier, we have implemented two versions of decomposition, one with a constant pyrolysis process and one with time varying pyrolysis.

a. Constant Rate Pyrolysis

Our first implementation is based on a constant fuel gas release model. During simulation, any object cell passing the self-pyrolysis threshold converts a fixed amount of solid fuel into gaseous fuel until it consumes all of its solid fuel. It is a simple on/off switch, continuing until it runs out of fuel (or until the temperature drops low enough, which is unlikely in realistic simulations).

$$\frac{\partial}{\partial t}V(\vec{x}) = dt \begin{cases} k_2, & \mathrm{T}(\vec{x}) \geq T_{sp} \text{ and } \mathrm{V}(\vec{x}) \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.12}$$

$$-1 \leq k_2 \leq 0 \tag{3.13}$$

where $T(\vec{x})$ is the temperature at $\vec{x}$. k_2 controls how fast fuel gas is released. Together with the self-pyrolysis temperature threshold (T_{sp}), it controls how volatile the solid will be. Since the fuel gas release is constant from burning cells, tracking the flame spread direction is not possible. For the direction, then, we used for our decomposition direction the normal from the implicit model, which is a "sure" way to ensure the boundary continually moves inward.

$$\vec{D}(\vec{x}) = -\vec{N}(\vec{x}) = -\frac{\nabla\phi(\vec{x})}{|\nabla\phi(\vec{x})|} \tag{3.14}$$

Putting equations 3.10 and 3.14 together, the decomposition is defined as

$$\frac{\partial\phi}{\partial t} = -k_1 \frac{\partial}{\partial t} V(\vec{x}) \tag{3.15}$$

b. Time-varying Pyrolysis

Instead of having a constant release rate, we can define the amount of fuel released as being proportional to the solid fuel left. There are then multiple ways of defining the pyrolysis, one of them being

$$\frac{\partial}{\partial t} V(\vec{x}, t) = dt \begin{cases} k_3(1 - \frac{V(\vec{x},t)))}{V(\vec{x},t=0))}), & \mathrm{T}(\vec{x}) \geq T_{sp} \\ & \text{and } \mathrm{V}(\vec{x},\mathrm{t}) \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.16}$$

We choose a monotonically decreasing function, since it will give us the benefit of determining the direction of the moving flame front. Cells newly entering into the pyrolysis stage will have a higher output than the cells in a more advanced state of pyrolysis. By examining these differences, we can define the direction of flame propagation.

$$\vec{D}(\vec{x}) = \frac{\nabla \frac{\partial}{\partial t} V(\vec{x})}{|\nabla \frac{\partial}{\partial t} V(\vec{x})|} \tag{3.17}$$

Putting equations 3.10 and 3.17 together, the decomposition is defined as

$$\frac{\partial \phi}{\partial t} = -k_1 \frac{\partial}{\partial t} V(\vec{x}) \frac{\nabla \frac{\partial}{\partial t} V(\vec{x})}{|\nabla \frac{\partial}{\partial t} V(\vec{x})|} \cdot \vec{N}(\vec{x}) \tag{3.18}$$

Note that equation 3.18 is valid for a wide range of pyrolysis definitions. We can adjust the release rate over time to be any monotonically decreasing function, and equation 3.18 holds.

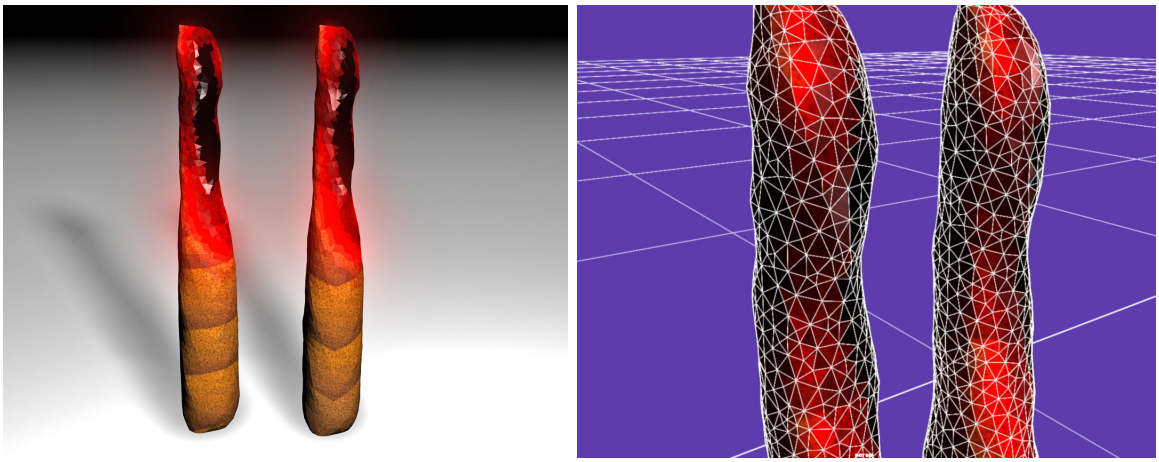

Fig. 17. Constant-rate vs time-varying pyrolysis. Constant-rate pyrolysis is at left, time-varying at right.

c. Comparison

As seen in figure 17, both methods of decomposition give similar results. This is to be expected, since the amount of decomposition applied is the same in both cases. Even in the closeup view the results are almost the same. The only differences are that the direction of decomposition changes slightly, and that the normal possibly gives a nicer approximation. To summarize: in most cases, time-varying pyrolysis is not needed to model a realistically decomposed model. Note that the color change rendered is based on how long a given cell has been in pyrolysis.

d. Rigid Body Motion

Collisions are determined by comparing objects pairwise. For one object, we use a set of particles placed at the vertices of the visualization polygons. After a global-to-local transformation of the vectors, interpolation on the distance field grid of the other object directly gives us the approximate distance to the boundary, letting us know whether the objects have collided (and if so, where) [44].

Note that the particles live only for one frame/timestep of the simulation, since the visualization polygon is discarded and recreated in the next timestep, keeping only the position, velocity, and angular velocity of the solid piece from frame to frame. Standard collision response approaches then model response of the pieces. Note that rigid body motion is a global change affecting all representations of the object in the multi-representation framework.

e. Disconnected Pieces

Pieces that become disconnected in the burning process should be detected. The polygons created from the implicit representation are used to detect such separations. Here, note that we need to have a complete solid object representation (including topological connections), compared to the faster polygon-soup algorithms.

We begin by tagging all vertices as "not visited". We select an unvisited vertex. We tag it as "visited", and put it into the process queue. We process the first vertex in the queue by traversing all connected unvisited vertices. They are tagged as "visited" and put into the process queue. This iterative process tags all the vertices forming a connected component in a breadth-first-search fashion. We process the rest of the vertices similarly, and find all the connected components.

Once the connected components are detected within the given solid, all the cells

containing the same connected component are copied to a new solid object. This procedure splits the volume and distance fields accordingly, creating two or more separate solid objects.

This structure allows us to simulate the motion of individual pieces, while each still burns and decomposes inside its local computational domain. Finding the changed center of mass and estimating the moment of inertia is straightforward once each object has been separated. Here, we use unit mass particles placed at the filled cells of the new solid to estimate the new center of mass and moment of inertia.

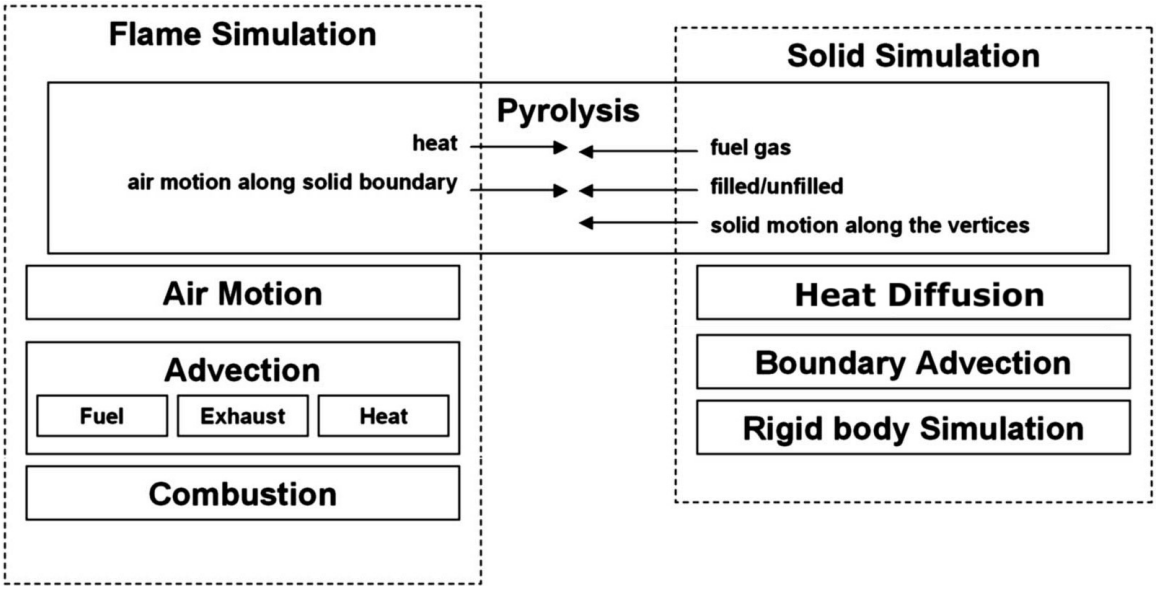

Fig. 18. Integration of simulation modules.

4. Algorithm

Figure 18 shows an overview of the entire fire simulation process, including how our solid decomposition method fits in. The simulation occurs in two separate "threads" connected via the pyrolysis and heat transfer steps. Within this simulation framework, the solid simulator incorporates the following steps:

- Heat is exchanged to/from the outside through the solid/air interface. The heat property is updated.
- Internal heat diffusion is modeled using Eq. 3.6. The heat property is updated.
- The solid fuel amount property is changed in the cells undergoing pyrolysis based on the heat property.
- The fuel release in the previous step is used to define the amount of decomposition, and used to update the implicit boundary representation using a levelset method. The zero isosurface of the updated implicit boundary representation is used to visualize the object.

5. Parameters

There are many parameters controlling the decomposition behavior of the burning objects. We briefly overview these:

- *Self Pyrolysis temperature* of the material determines how volatile a solid is (at what temperature it releases combustible gases).
- *Diffusion rate* controls the heat diffusion in the solid, and thus the spread of the flame front on the solid.
- *The fuel/residue rate* controls how quickly and how far the object decomposes, and how much ash is left.
- *The fuel gas multiplier* controls the amount of fuel gas coming from the burning solid, and thus the size of flames.

There are also additional parameters that affect the combustion reaction, and hence indirectly control the decomposition behavior.

- *Oxygen use rate* (stoichiometric mixture) controls the size of the reaction zone as well as the survival of the flame under low oxygen conditions.
- *Output heat* determines how easily a flame becomes self-sustaining and spreads.

These parameters reflect physical quantities; if actual physical quantities are known (e.g. the thermal conductivity), they can be used.

Our parameters give a user a great deal of control for describing object behavior, and the interactive framework enables rapid testing of values.

D. Bending Matches, Crumpling Paper

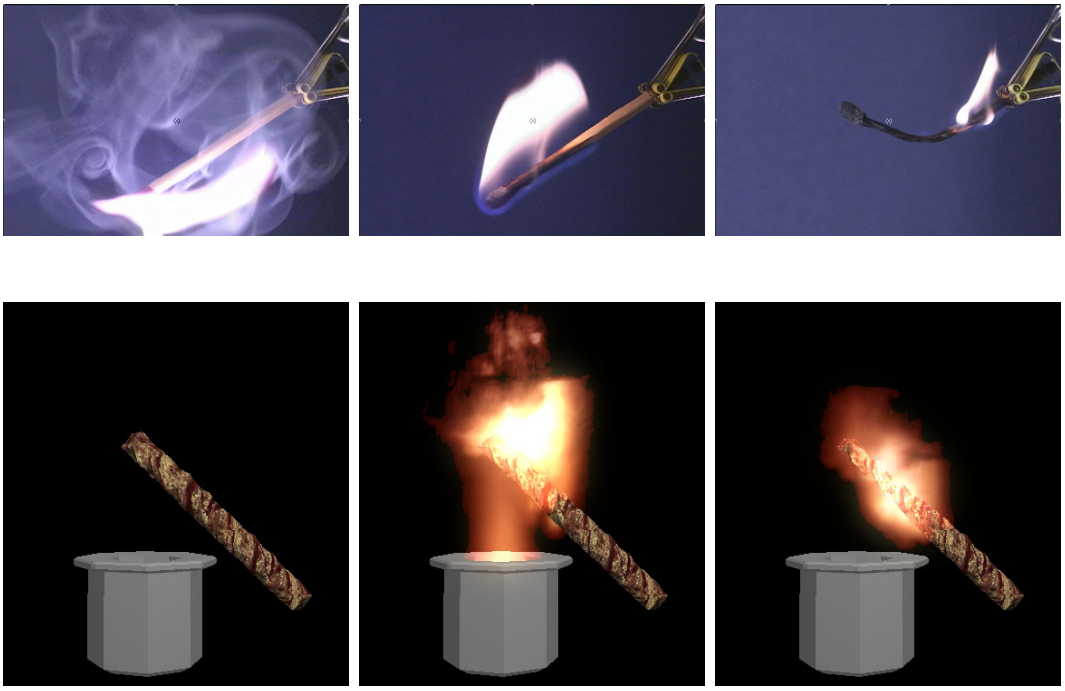

Fig. 19. Real vs simulated matches without bending.

Burning objects is a complex physical process. To simulate a burning object, the combustion reaction, heat distribution, fuel consumption, and even object shape must

be modeled and changed over time. The pyrolysis process, where an object releases combustible gases, causes decomposition and additional structural changes in burning objects. Although these structural effects are usually minor, some create a dominant deformation on burning objects. One such structural change is caused by microscopic contraction of fibers within a burning object. This results in effects that are quite noticeable at the macroscopic level, such as the way matches bend when burned, and the way burning paper tends to crumple. We can simulate the combustion reaction, the object catching fire and burning, and even the decomposition of the object as it burns, but unless we model the fiber contraction, the simulated matches will not behave like actual burning matches (Fig. 19).

Many complex physical simulations require modeling of a number of different phenomena. Depending on the particular application, these phenomena can require different levels of accuracy in order to create an overall visually plausible result. Though an "ideal" simulation might accurately simulate all details of all physical phenomena, this is usually impractical, given constraints on time, processing capability, and even underlying knowledge of the physical process. Instead, what is usually done in the computer graphics (CG) community is to simplify the physical model, use a simpler simulation, and eliminate certain secondary effects in order to achieve a plausible result in reasonable time. This is the case for all simulations, but it is particularly magnified in interactive applications.

In this section we will present an efficient model for certain secondary effects in burning objects, such as bending burning matches and crumpling burning paper, thereby increasing visual plausibility of the overall simulation for only a reasonable cost in efficiency. We do this by attempting to model the large-scale effects of certain physical processes, rather than spending a disproportionate amount of computation on a minor yet potentially complex phenomenon. In particular, we propose a way of

approximating larger-scale deformations of objects guided by the results of a simulated physical process.

The major contributions in this section are

- We present a framework for creating deformations guided by physical simulations. This includes:
  - Defining a proxy object.
  - Mapping simulation parameters to the proxy object.
  - Modifying the proxy object based on the simulation results.
  - Creating a deformation using the modified proxy object.
- We apply this global framework on our burning objects to model physics-driven simulation of two physical phenomena, namely:
  - Bending burning matches.
  - Crumpling burning paper.

Note that we have published this deformation model in a more generalized form [77].

1. Simulation Guided FFD

Rather than modeling the actual chemical process fully, we propose a simplified model to mimic similar behavior for secondary deformations. Similar to the multi-representation framework described earlier, we use the change of object properties during the simulation to control the deformation. Note that we assume there is some physically based simulation determining the "major" processes acting on the object,

and the effects of any "secondary" processes are either too complex to model, or too time-consuming to simulate. We therefore use a simplified deformation, guided by the primary simulation, to model the deformations created by these secondary effects, rather than modeling the effects themselves.

The overview of our proposed method is as follows (Fig. 20)

- Run the simulation for the primary physical processes affecting the initial object.
- Place a proxy object around the deforming object of interest.
- Map the simulation results from the initial object onto the proxy object.
- Determine an approximated deformation for the proxy object using the mapped properties.
- Use the deformed proxy object to control a deformation applied to the original object.

In the remainder of this section, we will describe the elements in the most general terms. Later sections will show how this can be specialized to the burning process.

a. Placing a Proxy Object

The basic idea here is to define a simplified object which can later be used to deform the object of interest. We refer to this simplified shape as a "proxy object." Given a particular object or region of interest, there are many ways of defining a proxy object. Options include a bounding box, the convex hull of the object, a simplified version of the original object, or a user-specified simple shape. This proxy object simplifies the calculations required for deformation, such that we can define the deformation using the proxy and than apply the same deformation to the high resolution object.

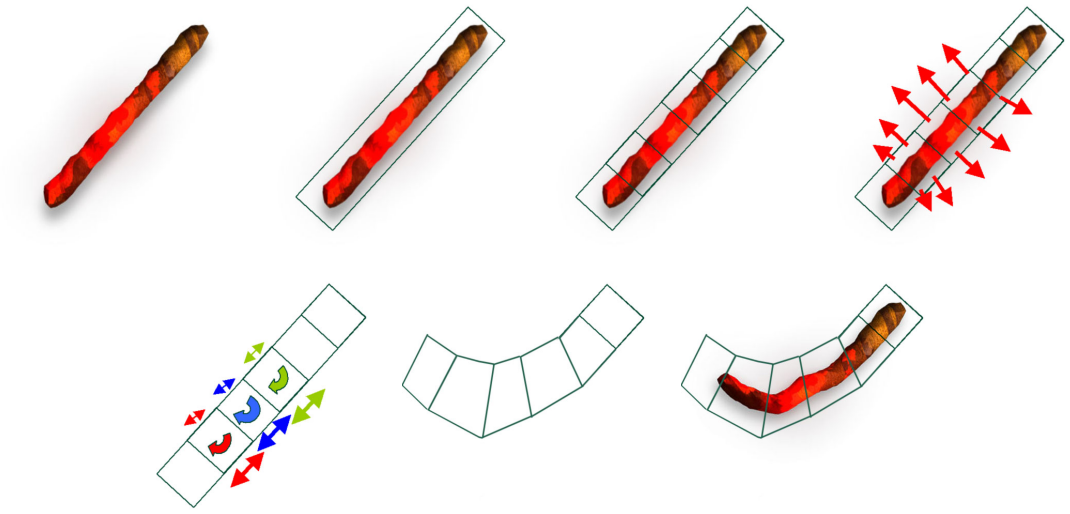

Fig. 20. Simulation driven deformation process. (a) initial object, (b) placing proxy, (c) subdivide along the deformation axis,(d) mapping parameters onto the proxy, (e) deformation defined on the proxy, (f) apply the deformation onto the initial object.

In general, we will consider the proxy object to be a polygonal boundary representation that encapsulates the object of interest. However, it is certainly possible to use other proxy objects. For example, a tricubic spline function would be useful for some applications where deformations of the internal structure must be modeled. In other applications, a medial or skeleton-based representation might be useful. There is no particular restriction on the proxy object that can be used, other than that it must be usable in the later stages of the process; the later need to map properties onto the proxy and define a deformation from the proxy might influence the particular choice of a proxy object. Obviously, in most cases the proxy object should be significantly simpler in some sense (e.g. fewer polygons, lower genus, or more convenient representation) than the original object, since the idea is to simplify the simulation.

Note that while this is not a strict restriction, we will make an assumption that we have a single proxy object of genus zero. In reality, even if we start with a genus zero

object, solids might undergo topological changes during complex physical simulations. We usually do not want to have to model such topological changes explicitly in the proxy, as it would defeat the simplicity we are seeking in the proxy. So, we assume that a genus zero proxy will be sufficient to represent any deformations and that any changes in the deformation due to topological changes in the underlying object will have only a minor effect. Clearly this is not always the case; if a single deformation cannot be used to model the underlying deformation, or if the proxy would need to change in genus, another method might be more appropriate.

b. Mapping Simulation Results

Given a proxy object and a simulation on the underlying object, the second step involves mapping the "interesting" simulation properties from the original object onto the proxy. The exact nature of this mapping will vary depending on exactly what proxy object was chosen. Note that we are not mapping the results of the deformation we are trying to simulate, rather we are mapping properties that can be used to later define the deformation. For example, for the examples we provide below related to burning objects, we map internal heat distribution and rate of pyrolysis onto the proxy faces.

If we assume that we have an encapsulating convex proxy object, the mapping process can be as simple as a cylindrical or spherical map. If a skeletal proxy is used, the mapping process can be a simple retraction to the skeleton. Note that this mapping onto the proxy is not necessarily one-to-one; several points of data from the underlying simulation can map to the same point on the proxy object, or none of the underlying points on the object might map to a particular point on the proxy. Also note that parameters could map to different parts of the proxy object. For example, for an encapsulating polygonal proxy, we could conceivably map parameters to the

faces, or to the edges, or to the vertices. The part that we want to map to should be determined by the way the proxy object itself will be deformed, as described in Section c.

c. Defining Deformation of the Proxy

The critical part of our proposed method is defining the approximated deformation on the simplified proxy object. However, it is difficult to give details for this process as this portion will be very specific to the particular simulation. The main point is that we take the parameters that have been mapped onto the proxy object, and use these to define a deformation of the proxy object itself.

A few examples of the ways that the mapped properties (the "values") might define a deformation on the proxy object include:

- The values can be used as constants for a spring system along edges of the proxy. The proxy object can then be simulated to equilibrium.
- The values can be used to define a local transformation (e.g. scaling, translation) of the proxy mesh. The deformed proxy is formed from the superposition of these local transformations.
- Weighted averages of the values can be used to determine parameters of the proxy. For example, a medial proxy representation could have the radius information set by the mapped parameters.
- Values can be used to apply weighting to particular points. For example, for a tricubic rational spline proxy, some values could adjust the positions of control points, and others the weights.

Generally, a "good" deformation will have these properties:

- The deformation will have the same effect on the proxy that one would want it to have on the underlying object. That is, the proxy object should "behave" in the same way (though at a coarser level) as you would want the underlying object to behave in response to those simulation parameters.

- It will be significantly cheaper to compute than the simulation on the original object would have been. Note that you could potentially use the *exact* same deformation as would have been used on the original object, but save significant time by simulating over a much simpler object.

- The user should be able to control the way the deformation behaves. Generally, this deformation is an abstract model for some more complex underlying process. Because of this level of abstraction, it is likely that a user will need to set some parameters manually (e.g. the ratio between a particular mapped property and the spring constant it defines on the mesh). In addition, user control of *parameters* allows greater artistic control in some cases, while still ensuring that the deformation is defined by the physical simulation.

d. Applying the Deformation

Finally, we apply the deformation defined on the proxy to the encapsulated object. The proxy will define the deformation that will be used to warp the local coordinate system of the actual object. The specific way this is accomplished will depend on the way the proxy was defined.

For basic encapsulating polygon proxies, the simplest approach is to use the proxy object as a free form deformation (FFD) lattice around the deforming object. We can decompose the cells into tetrahedra [81] and apply piecewise linear interpolation using barycentric coordinates inside the tetrahedra. This approach can be made arbitrarily

more complex. If more continuity is desired, a nonlinear interpolation method could be used. Or, one could easily use mean value coordinates [55] or similar interpolation schemes to define a deformation from a given polygonal proxy.

Other types of proxies would afford different deformations. For example, tricubic spline proxies could directly define an FFD. Skeletal or medial proxies could be used as the basis for a distance-field based deformation of space.

2. Bending Matches

We describe here how the deformation described above can be used to model the bending of burning matches (Fig. 21).

Fig. 21. Before and after simulation driven bend deformation.

Matches are made out of fibrous material oriented along the length of the match. During burning, the fibers lose water and other chemicals. Though there is still debate about the precise mechanism, this loss of material causes the fibers to contract at a microscopic level. Due to the shape of a flame, the upper part of a match

receives more of the heat generated from the combustion reaction that forms the flame. Thus, it is hotter on the upper side of the match compared to the bottom; this means that the fibers on the top contract more than the ones at the bottom. This imbalance accumulated at the microscopic level forces the match to bend upwards at the macroscopic level. This is exactly the type of situation our proposed method was designed for: a simulation-driven deformation, where modeling the true process (fiber deformation at a microscopic level) is impractical, but the overall behavior is significant.

We begin by selecting a proxy object for the match. Due to the shape of the match, we use a simple bounding box aligned along the match axis as a proxy. This bounding box is subdivided into a number of individual segments along the bending axis (the length of the match), creating a 1x1xN FFD lattice surrounding the match.

During the simulation of burning and decomposition, the rate of pyrolysis $\frac{d}{dt}P$ and internal heat T of the object are mapped onto the faces of the proxy object. Each face of the proxy will store the average of the pyrolysis rate and heat values that map to it. Since the proxy object we used in the previous step is rectangular, we can just use a cylindrical mapping from the burning match onto the lattice. Note that this simplification does not take changing topology during the decomposition process into account, but this should introduce only a minimal amount of error.

Before we define the deformation, we need to clarify one issue. We can define a simple cylindrical mapping at the start, but what will happen as we bend the match? Since the match will also decompose, the faces will change and we cannot fix the mapping. One thing to note here is that although the match is bending in world space, it is still undeformed in its own local space, and so is the proxy. Hence we can define the mapping from the object to the proxy in the unbent local object space.

At this point, we have our proxy object and some simulation values (pyrolysis rate

and heat) stored at the faces, and must use that information to deform the proxy object. To achieve the bending behavior we are seeking, we consider opposing faces of the proxy object. The proxy faces are contracted or expanded according to the difference on facing faces. This individual contraction/expansion of the individual proxy cells defines a smooth deformation along the proxy object. This contraction/expansion of the opposing faces also causes rotation of the rest of the lattice.

$$D_T^{ij} = \Delta T_i - \Delta T_j \tag{3.19}$$

$$D_P^{ij} = \frac{d}{dt}P_i - \frac{d}{dt}P_j \tag{3.20}$$

$$D^{ij} = \begin{cases} \alpha D_T^{ij} + \beta D_P^{ij}, & P_i, P_j \geq P_{thresh} \;\&\; T_i, T_j \geq T_{thresh} \\ 0, & \text{otherwise} \end{cases} \tag{3.21}$$

where i and j are opposing faces, D^{ij} is the rotation amount, and α and β control how much the heat and rate of pyrolysis affect the final deformation respectively. These are the parameters that allow an element of user control to the deformation. T_{thresh} and P_{thresh} are thresholds for starting the deformation. Note that the axis around which the rotation occurs is perpendicular to both the $\vec{ij}$ axis and the axis along the match length.

On a rectangular proxy, we have two sets of opposing faces, hence two sets of deformation values for every cell (D_{ij},D_{kl}) one pair for each axis, orthogonal to the match length. We rotate the cells centered around the "active" cell using these deformations (Fig. 22). Here, we can either apply half of the required rotation to both sides, or fix one side (if the match is anchored or held at one end) and apply the whole rotation to one side only.

Finally, we use our deformed proxy to deform the match itself. We subdivide the proxy lattice cells into tetrahedral elements [81] and apply piecewise linear inter-

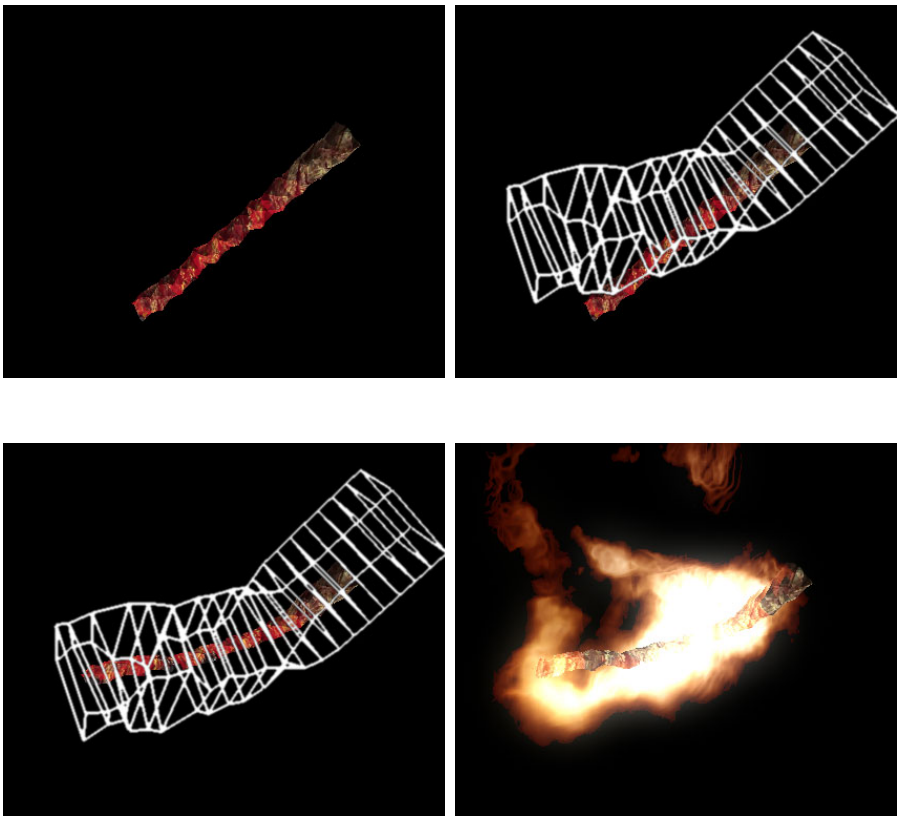

Fig. 22. Bending match using simulation guided FFD.

polation using the barycentric coordinates inside the tetrahedra to deform the space around the burning match. Although this approach is only C^0 continuous, we have not observed any artifacts even with a fairly small number of segments ($N = 20$). This, creates the overall deformation that we were seeking.

Note in all of this the importance of the choice of proxy object. The particular proxy object we chose to use plays a big role in what mapping can be used, how the proxy object itself can be deformed, and how easy it is to define a deformation from the proxy. While a different proxy object could be used, it might require significant modification of the approach, particularly for determining how the mapped values are used to define the deformation.

3. Crumpling Paper

We describe here a second application, crumpling of burning paper, using a different type of proxy object. Like matches, paper is also made out of fibrous material. When paper is burned, the contraction of the fibers can cause the paper to crumple (the specific nature of crumpling depends on the fiber structure). A simple single axis lattice proxy object like that presented in the previous section would not be applicable here, because of the planar structure of a sheet of paper.

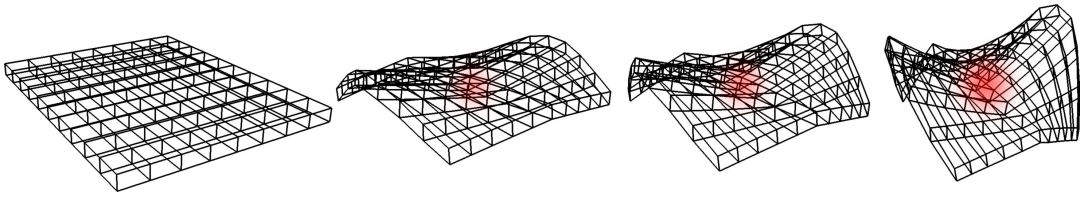

Fig. 23. Deformation of the center cell on the "2D lattice". Note that the deformation falls off as we move farther from the deformation axis.

First, we choose a proxy object suitable for representing the deformation of the planar paper. We use an enclosing bounding box, subdivided along two axes to form a NxMx1 lattice around the object. We will call this lattice a "2D lattice" (Fig. 23). One should notice that this approach is suitable for a burning sheet of paper, but not for burning folded origami artwork.

Similar to the case of bending burning matches, the rate of pyrolysis and heat are mapped onto the faces of the proxy, with each face storing an average. In this case, however, we use an orthogonal mapping from the object to the proxy. The face normal of the object determines whether the properties of that face are mapped to the top or bottom of the proxy.

The proxy object is then deformed. Similar to the case for matches, we will use rotations defined by the difference in values on opposing faces. For the match bending case, a simple rotational scheme worked fine using the deformation D_{ij}, but we need

to use a more complex scheme to deform our 2D lattice.

We will deform the proxy lattice by combining the deformation effects computed at each cell of the lattice. For a given cell of the lattice, i, we set an amount of deformation, D_i, by the same process as for D_{ij} in Equation 3.21. There are several possible ways to deform the proxy cell from this value. We choose a simple approach that works as follows:

- We choose an axis for the cell. The cell will cause the lattice to tend to "fold" about this axis, $\vec{u}$. Generally, we choose $\vec{u}$ to be one of the two lattice axes, passing through the center of the cell, though any axis through the center would be valid. We have not observed any difference in choosing one of the two lattice axes randomly versus alternating the deformation axis between neighboring cells.

- We then define how the deformation in that cell, D_i, will affect other cells in the lattice. That is, we model the *result* of the deformation of that cell on the nearby cells. This will be defined as a rotation about the axis chosen above. The amount should be highest near the axis and fall off as one moves away from the axis. We choose to use a cosine function to describe this weighting (Eq. 3.23), though a different function (e.g. gaussian) could be used instead. This effect corresponds to the rotation around the deforming cell in the match case, except only cells near the axis of the deforming cell are modified.

- The deformation defined above will tend to cause stretching in the lattice as we move along the deformation axis away from the cell. To minimize this effect, we add a deformation that pulls the corners of the cells closer to the axis (Eq. 3.24) as we move along the axis away from the original cell.

- At this point, we have defined a deformation function independently for each active cell. We then need to combine the contribution of these deformations within all the cells of the mesh. So, each cell of the lattice will end up in a new position defined by the deformation amounts given by the other cells (Eq. 3.22). Note again that after summing these deformations, we immediately have the final position of that cell – i.e. the result of the deformation.

To summarize, then, the new position of a vertex, V_j, in the lattice is determined as follows:

$$V'_j = V_j + \sum_i D_i * dist_u(i,j) * (D^1_{ij} + D^2_{ij}) \tag{3.22}$$

$$D^1_{ij} = \alpha * cos(dist_v(i,j)) \tag{3.23}$$

$$D^2_{ij} = \beta * dist_v(i,j) \tag{3.24}$$

where $dist_u(i,j)$ is the distance *along* the deformation axis of cell i to vertex V_j, and $dist_v(i,j)$ is the distance of cell V_j *orthogonal* to the deformation axis of cell i. α and β are parameters that allow user control of the deformation - i.e. controlling the amount of deformation created by these effects.

The collection of individual deformations results in a deformation that mimics the crumpling action seen in burning paper. Note that for simplicity we omitted any explicit tests to avoid self-intersections; self-intersections were never seen as a problem in our examples.

Because of the additive deformations, we observe excessive stretching behavior around the corners (Fig. 24). This is an expected result, since we are not using a volume preserving scheme. While controlling α and β can help reduce this, they are not sufficient. To avoid the artifacts that would result from such a stretching,

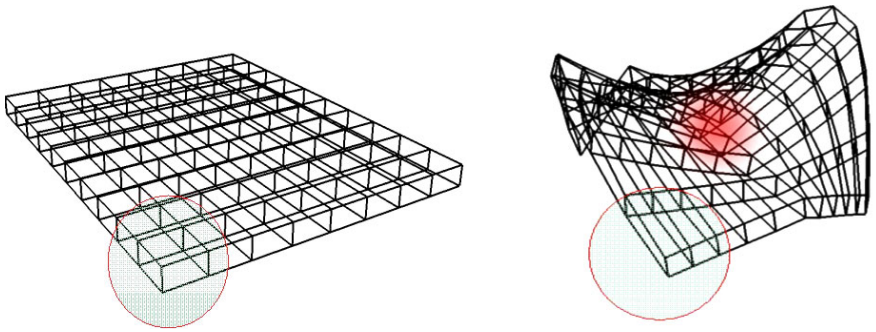

Fig. 24. Excessive stretching at the corners.

a relaxation step is applied to pull back those over-stretched areas to create a more plausible deformation. This is done by setting a spring mass system with the rest lengths defined from the initial state, and iteratively allowing the system to come to a rest state. In practice, we have observed that just a couple of iterations of relaxation are sufficient to avoid noticeable artifacts from the stretching. Note that such a cloth-like solver could have been used as an alternative (but slower) method for deforming the proxy lattice.

Now that we have deformed the lattice mesh, the final deformation is then applied to the paper itself (Fig. 25). Again, we can easily tetrahedralize the lattice, and use this to apply a linear FFD. A more complex deformation could be used but did not appear necessary in our tests.

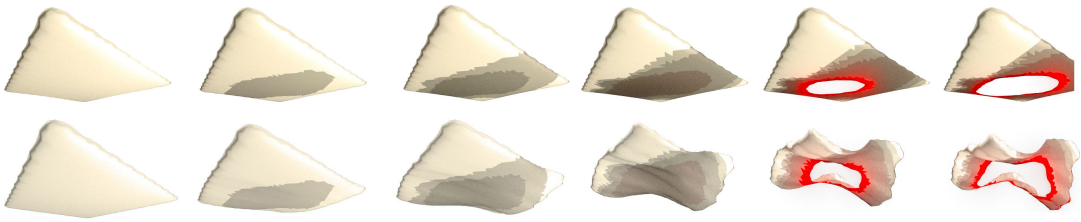

Fig. 25. Crumpling paper during burning. (top) decomposition alone, (bottom) decomposition and crumpling. Flames are not shown.

Note again the significance of our choice of proxy shape. Besides requiring a

different mapping from the previous example, the deformation process applied was also significantly more complex.

4. Evaluation

a. Limitations and Drawbacks

Though our model presented here is simple, there are some drawbacks that accompany this simplicity:

- We ignored topological changes during the mapping of physical properties onto the proxy object.
- Material compression during the bending is ignored.
- Self collisions are not handled.
- The proposed model is suitable for simple deformations and might be of more limited use for complex deformations.
- By definition, we are not simulating the "real" physics.

b. Advantages

Although we do not simulate the actual physical process, we can approximate visually important deformations easily using our proposed model. Our method thus also has several advantages:

- This framework has a minor computation overhead that is easily integrated into existing systems.
- The user has some control over the desired behavior. Though the deformation is still physically-based, we have not eliminated the opportunity for user input.

- The method is "based" on simulation data, and thus is driven by a physical process, but does not actually simulate the "real" physics behind the process, which might be overkill for many purposes.

- Plausible results are easy to construct for the cases presented.

E. Results

Our results show that we can split the flame and solid simulation into two separate modules. By representing the solids on different grids from the fluid simulator, the solid module can be executed as a separate (but synchronized) thread. We will discuss the integration and synchronization issues in the next chapter. It is also possible to handle multiple solids with different material properties simultaneously as multiple threads. The simulations of the air flow and of the burning solid can be run at different rates.

Our results show that we can handle many kinds of burning objects with our simulation. Our model is also the very first model to address decomposition of burning objects [75][76] in CG. Our model is flexible, and easily extendable. We demonstrate extensibility by incorporating a bending/crumpling model, which is the very first model to address this phenomena.

CHAPTER IV

INTEGRATION

Even though there has been some work on simulation of flames in CG, there is not even a partially complete framework addressing fire related phenomena. Oxygen use, smoke filling, tracking flame fronts on burning objects, or simple fire spread has been simulated, but never in an integrated framework. Burning object decomposition, and internal stresses associated with that has not been investigated to our knowledge, especially within the graphics community.

The physical process enables us to separate the combustion and pyrolysis processes, and simulate them in their respective spaces as presented in the previous two chapters. We presented the properties of each module, and in this section we will discuss the required communications among them (Fig. 26).

Fig. 26. Volatile and non-volatile bunnies.

A. Solid-Fluid Coupling Overview

Solid-fluid coupling is done two ways. In the single direction coupling, or one-way solid-fluid coupling, either solid or fluid motion is predetermined and its effect on the other is found. In two-way coupling on the other hand, solid can affect fluid and vice versa. In general, two-way coupling is a difficult and computationally expensive problem. For an extensive overview about solid-fluid coupling we encourage the reader to review two recent dissertations [14][43] focusing specifically on this issue.

Two-way coupling is performed using fluid pressure to generate forces on the solid, with solid motion providing boundary conditions for the fluid velocity. Particle based methods such as smoothed particle hydrodynamics (SPH) use boundary particles to track the fluid-solid interface [96]. The drawback of particle based methods is preventing leaks.

In a recent work, Carlson et al. [12] presented the rigid fluid method, using distributed Lagrange multipliers (i. e. particles) to ensure two-way coupling that generates realistic motion for both the solid objects and the fluid as they interact with one another. Their simulator treats the rigid objects as if they were made of fluid. Guendelman et al. [66] developed a new method for dealing with thin objects that do not contain an interior, coupling cloth-like objects with the particle level set method. A related work, though not a solid-fluid coupling, is Losasso et al [67], which presents an impressive extension to particle level sets by dealing with multiple liquids simultaneously.

Another classification for fluid-solid coupling is between "strong" and "weak" coupling. Strong coupled systems evolve both solid and fluid together. Weak coupled systems on the other hand use a partitioned approach, and update solid and fluids iteratively. Strong coupling is more stable but less efficient.

B. Integration

We have developed a simulation framework, presented first in [75], consisting of two main modules, along with a synchronization and data exchange interface. The two main components are a fire/flame simulation and a solid object simulation; each of these is composed of several sub-modules dealing with different phenomena. The two simulations are coupled together by: pyrolysis (transferring fuel from the solid representation to the fluid representation), heat interpolation (transferring heat information from the fluid simulator to the solid representation), and external forces (changes in air motion due to object movement and the reverse).

The fire module is responsible for air motion, gas distribution, and heat generation. It models the combustion process, which generates heat and drives the air motion.

Input/Output of the flame module is as follows:

- Fuel sources, including solids in pyrolysis, provide fuel gas to the module.
- The interface with the solid object module also provides occupancy information (filled/unfilled voxels), which are taken into account during air motion.
- Fire simulation provides heat information to the decomposition process.
- Moving air can affect, and could be affected by, moving objects.

The solid objects module consists of heat transfer, object decomposition, and a rigid body solver. The decomposition module interfaces to the fire simulator to obtain heat near the object, and diffuses this heat into the object. If any part of the solid is in pyrolysis, it outputs fuel gas to the fire simulation, by converting solid fuel held in the combustible solid to gaseous fuel in the fire simulator.

Input/Output of the solid module is as follows:

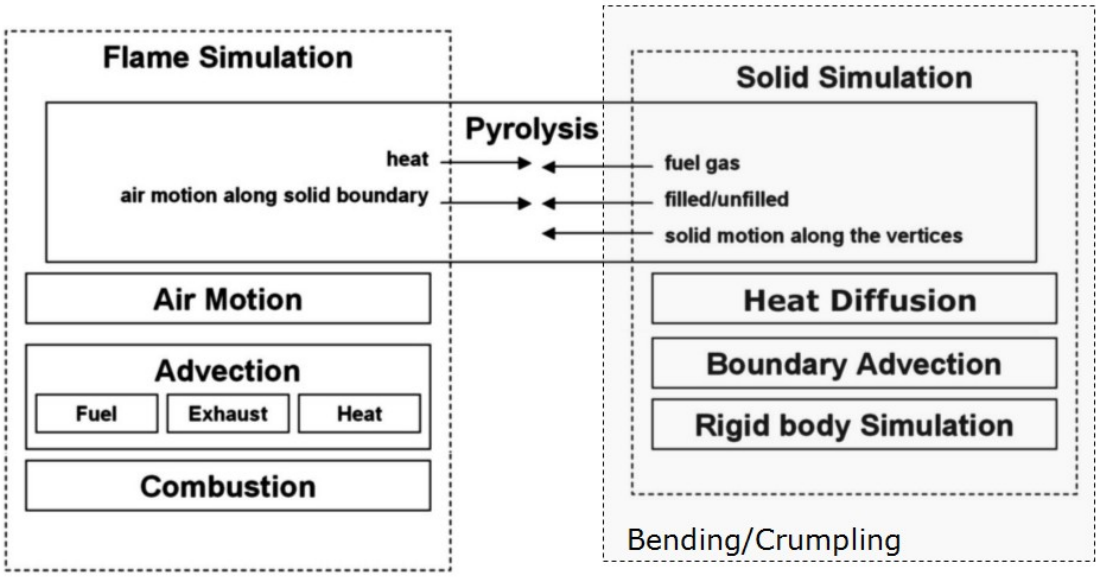

Fig. 27. Extending the model to include bending/crumpling.

- Heat information should be provided from an external source, from the flame module in this case.
- Air motion can affect rigid body motion, hence air motion near the solid should be provided. Although this is a relatively minor effect, it might be necessary to model floating of burning paper pieces for example.
- During pyrolysis fuel gas is injected into the fire simulation module with some direction.
- The motion of the solid object can affect the air motion, so the motion vectors along the solid-air boundary should be provided to the air motion module.

C. Flame - Solid Interaction

By representing the solids on different grids from the fluid simulator, the solid module can be executed as a separate (but synchronized) thread (Fig. 27). It is also possible

to handle multiple solids with different material properties simultaneously as multiple threads. The simulations of the air flow and of the burning solid can be run at different rates.

Fig. 28. Burning match.

The two simulations are joined through the pyrolysis step:

- A pyrolysis time step begins with the initialization of filled/unfilled information from the solids into the flame module. The solid grids are examined to find whether the point at the center of a fluid grid cell is inside any solid. If so, that fluid grid cell is marked filled, restricting air flow in the fluid simulation.

- The air-solid boundary cells are traced and the heat gradient between air and solid is used to exchange heat explicitly. Heat diffusion is computed for the air and solids separately, using different diffusion rates.

- The next step is transferring forces in the fluid motion field. The fluid flow exerts forces on moving solids, and moving solids generate forces within the fluid field.

- The fire module takes one time step in the fluid dynamic simulation, using the forces and filled/unfilled information set in the previous steps. The resulting air motion field is used to advect fuel and smoke gases, as well as heat. The resulting gas distribution combined with the heat is used for the combustion process. In each cell, fuel gas (and oxidizer) is consumed, and smoke and heat are formed.

- Solid objects are simulated within their own local space. Each cell is checked for whether it can undergo pyrolysis or not. The pyrolysis cells convert solid fuel into fuel gas as a function of heat. This fuel gas is passed to the flame simulator. The fuel conversion differential drives the object decomposition process, as described above. Decomposing objects are checked to see if they have separated into two or more parts, and rigid body motion is determined. Secondary effects, such as bending and crumpling are performed by warping the local space, which requires no changes in the flame-solid interface setup.

D. Results

Our results show the strength of the hybrid representation. Different solids, as well as separated pieces, such as ash, could be packaged separately, and the individual decomposition processes could be run as parallel threads, taking advantage of modern multithreaded/multicore CPU architectures. It is also possible to handle multiple solids with different material properties simultaneously as multiple threads (Fig. 28). The simulations of the air flow and of the burning solid can be run at different rates. The pyrolysis step of the simulation will be the synchronization and data exchange point. The packages themselves are good for rigid body simulation, and a distance implicit set is used for the collision detection. Here, one should note that during

decomposition, the center of mass of the solid changes, as well as the moment of inertia, and both are accounted for.

The FFD model is placed between the flame simulation and solid decomposition processes. The proposed process is simple, and could also be used to model similar behavior in other small-scale objects. By using an FFD approach, we can deform complex models without the additional simulation cost associated with modeling the actual physical process of the secondary deformation.

There are also some drawbacks to our hybrid synchronized approach. Our strong coupled and simultaneous execution of flame and solid processes has the problem of losing some fluid mass. To point out a sample problem, when solids move around the fluid, the formerly unoccupied fluid will now be occupied. Since our model is targeted for interactive applications at coarse resolutions, we can assume that this fluid loss is minor compared to other dissipation effects, and the benefits of having a parallel thread with loose synchronization is more beneficial. Although it is not a major problem in interactive applications, it should be taken into account and addressed in cases of high resolution and more accurate applications.

CHAPTER V

INTERACTIVITY AND SIMULATION CONTROL

As we discussed in the first chapter, the simulation and visualization of natural phenomena has been an area of great interest within the computer graphics community. Advances in the fluid simulations underlying many natural processes have made visual simulation of such phenomena quite popular. Despite recent hardware and algorithmic improvements, however, these simulations of water, smoke, clouds, fire, etc. can take a long time (many minutes to hours) to compute.

There are two particular challenges faced in controlling simulations in many applications. One of these is making the simulation match some artificial, user-defined keyframe (e.g. smoke forming a shape). This desire has driven recent research in which the fluid system is forced or "encouraged" to match those keyframes as closely and smoothly as possible. A second challenge is the long time taken to produce a production-quality simulation. Only after a long period of simulation might one discover that a simulation is not producing the right effect, and evaluating the effect of any change is also time consuming. Our work presented here is designed to address the latter issue.

A reasonable approach is to first run a fast, low-resolution simulation before the "full scale" one, but these very changes in the grid and time resolution can result in different simulation behavior. Beyond the increases in complexity of the events and changes in the initial conditions, the numerical simulation itself will behave differently when the fluid grid is resized. Numerical dissipation in the vector field, which causes an artificial viscosity inherent to the semi-lagrangian methods, will decrease as the cell size decreases. Also, better sampling of boundary cells for objects in the grid changes the shape of the fluid volume (this has been combatted by adaptive subdivision [65]

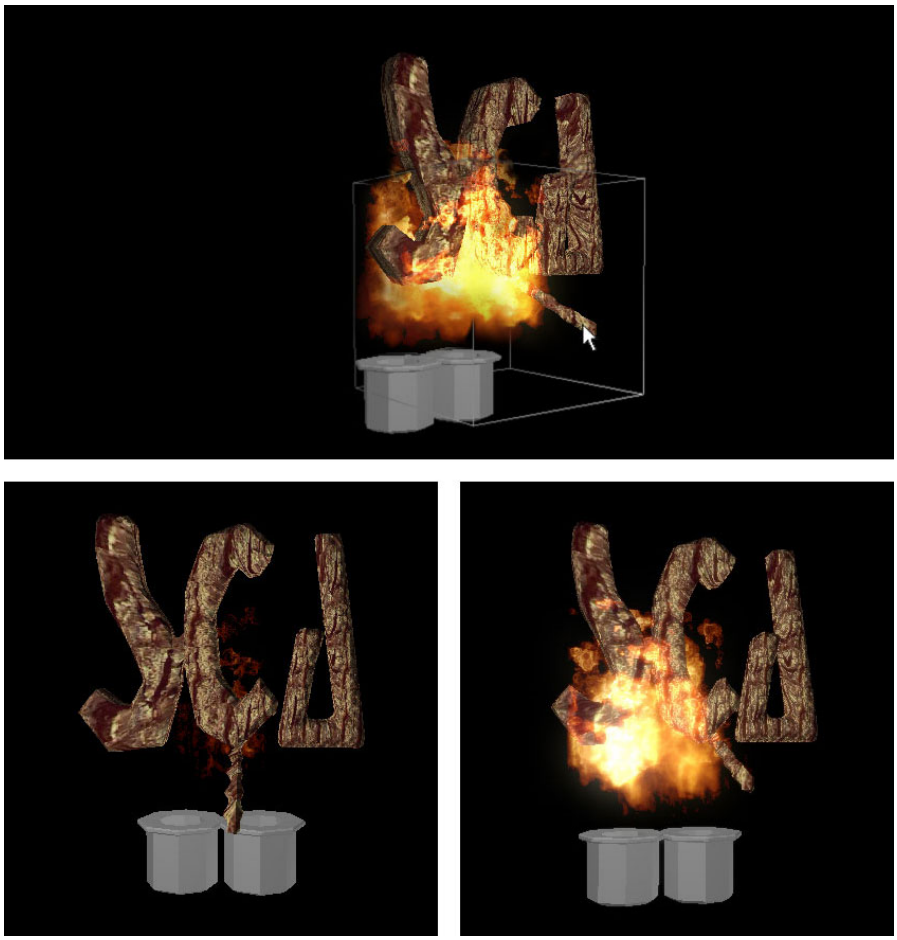

Fig. 29. Uncontrolled vs controlled simulation. Uncontrolled simulation on the left, where the logo does not catch fire, and controlled simulation on the right, with the logo catching fire as seen in the preview (top).

and tetrahedral meshes [30][58]).

In this chapter, we present a simple method for matching a fine grid fluid simulation to a coarse resolution "preview" simulation. This enables one to first use a simple interactive simulation (e.g. a GPU based fluid solver [46]) to produce a plausible, but rough, simulation. With the interactive simulation, a user can fine-tune and choreograph initial conditions and parameters quickly while getting visual feedback. We sample the data from this preview simulation in order to capture the gross properties

of the simulation that we are most interested in maintaining. Our matching process makes the final simulation more closely match these samples, and thus maintain the properties of interest (Fig. 29).

The obvious concern is that by matching to a coarse simulation, the benefits of the high-resolution simulation could be lost. The key observation here is that we do not require "exact" matching at the single grid cell level, but rather ensure that the gross (and specific smaller) properties of the simulation at user-defined scales are maintained. The controls placed on the high-resolution simulation will, in general, have only a minor effect on the simulation, and will have little effect on the fine-scale detail present.

Note that in contrast to some earlier methods that deal only with simple non-reactive gases or liquids, our method is better suited to match multiple properties at once. We demonstrate our implementation on a reactive gas flame model.

In summary, the main aspects of this chapter are that we:

- present an automated method for obtaining guidance data from coarse interactive simulations,
- present a method for adjusting a complex simulation from our guidance data, and
- can handle complex, reactive systems with multiple densities.

This is all tied together in an efficient method that is easy to incorporate into existing systems. Together, these aspects give a designer of a complex fluid simulation a valuable tool to control the details of a simulation, reducing the time costs of repeated reruns and restarts.

A. Simulation Control Background

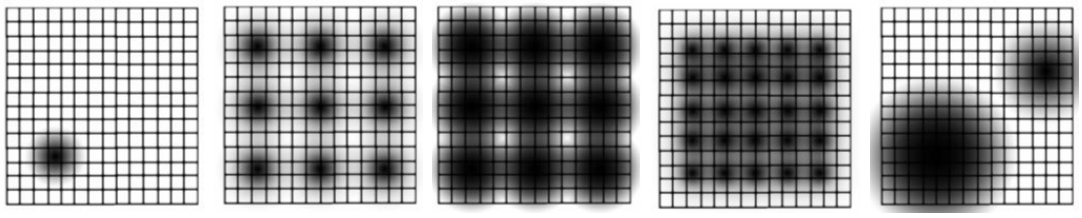

Fig. 30. Placement of data collection points. Data collection points could be placed in various ways. The examples show cases of changing the spacing of match points, and the size of the Gaussian filters.

The goal of control in a physically-based system is to modify the "true" simulation in a way that is visually *plausible* [6]. Control of general rigid body simulations has been one of the more explored topics [17][95][94]. Control of fluid systems was first investigated by Foster and Metaxas [36] and later by Foster and Fedkiw [34]. Rasmussen et al. [98] proposed a production model using particles to control liquid simulations. Shi and Yu have proposed methods for matching smoke [106] and liquids [107] to changing target shapes. Pighin et al. [93] have combined Eulerian and Lagrangian representations to create a system for interactively manipulating fluid flows. Their method parameterizes a fluid simulation from advected particle paths, allowing fine-grained control of the system by manipulating these paths. Hong and Kim [47] uses a geometric potential field to generate forces to match target shapes. Schpok et al. [101] control the simulation by manipulation of automatically extracted simulation features. Recently, Angelidis et al. [1] and Kim et al. [57] provide path control for smoke simulations. Though not controlling simulation, the work of Treuille et al. [118] also allows a user to generate high resolution simulations interactively. However, this requires massive and time-consuming preprocessing of scenes similar to the one the user wants.

Closer to our approach, several algorithms have focused on the keyframe concept. Treuille et al. [119] proposed a fluid control system based on user-defined smoke density keyframes. The fluid system is controlled by parametric wind and vortex fields that are manipulated to make the simulation achieve the specified keyframes. The amount of adjustment to the vector field is minimized via a non-linear optimization. This optimization is expensive, and dimensionality grows with the length of the simulation. The approach was practical only for 2D simulation, but McNamara et al. [73] improved on it by using a discrete adjoint method for gradient calculation. They achieve much faster convergence for control parameters, enabling fine-grain keyframe control even on large 3D fluid animations.

Instead of optimization, Fattal and Lischinksi [26] introduce new terms into the standard flow equations, offering a closed form solution for the control parameters. Matching target densities using only advection is usually not possible, and so they propose smoke gathering, diffusing the error field. This allows complex smoke animations to be controlled with little additional expense. Although our system was developed to match more general simulations, it resembles the keyframe matching in this work. We use different sampling, matching, and error computations. Also, while they use a global optimization to hit an arbitrary keyframe, we use local optimization to hit keyframes that are closely related to those computed without keyframing.

In these prior methods, control of fluids is usually achieved by controlling external forces f [119][73][26], and external sources S [73]. McNamara et al. [73] use sources and sinks only for the liquid simulation, similar to Foster and Metaxas [35], and Fattal and Lischinski [26] use smoke gathering instead of sources and sinks. Except for Fattal and Lischinski, previous fluid control systems focus on single density fields only. When multiple densities are advected with the flow, controlling only the flow field can not guarantee simultaneous matching of all density distributions. Although

there may be potential for previous methods [119][73] to be extended in this way, there is no such published demonstration.

B. Preview Based Control

As described earlier, we begin with an initial (low resolution) simulation we call the *preview.* We will refer to the preview data at a time t, and a position $\vec{x}$ as $v(\vec{x}, t)$. Note that v is actually a vector function, which may include various scalar and vector data from the simulation. Similarly, we have a second (high resolution) simulation that we refer to as the *final*, $\overline{v}(\vec{x}, t)$. Our goal is to produce an "adjusted" final version, $\overline{v}'(\vec{x}, t)$ that more closely matches the preview. Note that the final may differ from the preview in both spatial and temporal resolution.

Generally, we would like the ability to control how closely we want the final to match to the preview. At some regions/times we might wish to match exactly, at others we might not care about matching at all, and at others we might want something in between. We define a function, $m(\vec{x}, t) \in [0, 1]$ to describe how closely we wish to match.

The rough intuition, then, is that our matching process attempts to adjust the final such that

$$\overline{v}'(\vec{x}, t) = \overline{v}(\vec{x}, t) + m(\vec{x}, t)(v(\vec{x}, t) - \overline{v}(\vec{x}, t)) \tag{5.1}$$

We now describe the way we specify the matching function (section 1), how all the information is sampled (section 2), and how it is used to match our final to the preview (section 3).

1. Describing the Matching Function

The matching function m can be defined continuously over time and space. However, such a continuous representation tends to be impractical, first because the simulation data itself tends not to be continuous (but rather sampled at a grid), and second because defining this matching function continuously could use excessive amounts of memory.

Instead, we choose to specify the matching function as a collection of delta functions. That is, we choose a number of points, in time and space, at which we want to match the preview. For each of these points, we will have a single scalar value ($m_{i,j} \in (0, 1]$) that specifies the level of matching we desire, i.e. $m(\vec{x_i}, t_j)$.

There are several potential methods for choosing the positions ($\vec{x_i}$ and t_j) at which to specify the matching function (see figure 30). The most straightforward method is to place the sample points on a regular grid across the entire simulation domain and evenly through time. However, there are other options for choosing these points: e.g. probabilistically in either a completely random fashion or weighted to capture some feature/simulation event, or interactively based on user-identified regions of high importance.

2. Sampling Simulation Data

In order to perform matching, we must be able to determine the data values for the simulation at the $\vec{x_i}$. Note that simulation data is provided over a grid. Analogous to sampling that occurs in computer vision contexts, we consider this simulation data as representing only the lowest available layer in scale space [62]. Scale space can be thought of as adding an additional dimension, scale, to the data. The scale dimension is effectively the width of a blurring kernel that is applied to the data. Any sample of

simulation data, then, must be taken at a particular scale, r. Therefore, we can extend our match points to also take into account scale, i.e. $m^r_{i,j}$, allowing for collection (and match weighting) of multiple scales at a single spatial point.

While we could treat temporal data in a scale sense, there are significant problems that arise with such assumptions. Thus, we will assume that our sample is taken at a fixed point in time, t_j.

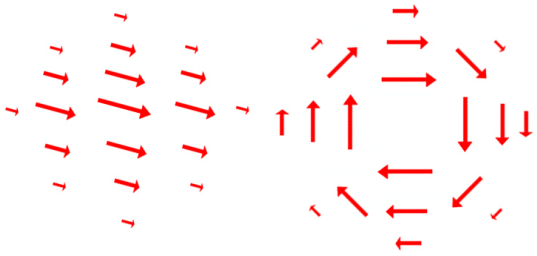

Fig. 31. Two types of wind forces. Directional and angular. The diagram at left has a high directional (but no angular) component, while the diagram at right has a high angular (but no directional) component. The length of the arrows show the weights relative to a Gaussian kernel.

a. Scalar and Vector Data Sampling

For a scalar property, d, we obtain a sample at time t_j, at position $\vec{x_i}$, and at scale r as follows:

$$d^r_{i,j} = \frac{1}{\sum G_r} \sum_x G_r(\vec{x_i}, \vec{x}) d(\vec{x}, t_j) \tag{5.2}$$

G is the Gaussian weight function with radius r:

$$G_r(\vec{x_i}, \vec{x}) = e^{-r|\vec{x_i} - \vec{x}|^2} \tag{5.3}$$

That is, we use Gaussian weights to sample the data. Note that our samples will occur only at the i, j, r defined by the match points.

Vector data $\vec{D}$ is sampled similarly, but we collect two pieces of information: direction $\vec{D}_1$ and angular momentum around the sample point with unit mass $\vec{D}_2$, each stored as vectors (figure 31). Storing this angular momentum allows us to maintain angular motion in the vector field at a scale relative to the size of the filter kernel, and does not directly affect the finer-grained vortices simulated in the higher-resolution simulation.

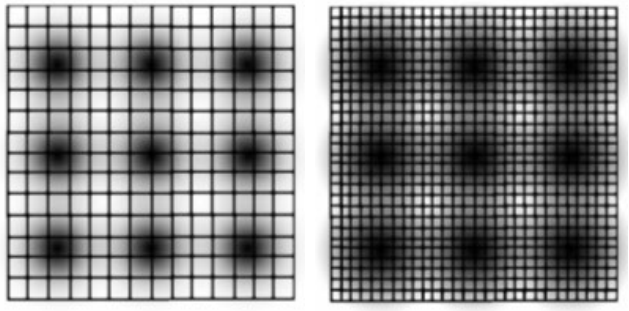

Fig. 32. Sampling positions and scales are independent of the grid size resolution.

$$\vec{D}^r_{i,j,1} = \frac{1}{\sum G_r} \sum_x G_r(\vec{x_i}, \vec{x}) \vec{D}(\vec{x}, t_j) \tag{5.4}$$

$$\vec{D}^r_{i,j,2} = \frac{1}{\sum G_r} \sum_x G_r(\vec{x_i}, \vec{x}) \vec{D}(\vec{x}, t_j) \times (\vec{x} - \vec{x_i}) \tag{5.5}$$

Position and scale is always recorded in terms of the physical space being simulated, not the grid-based space (thus, it will apply equally, regardless of simulation grid size - see figure 32). We can sample all of the key simulation data, including density fields (e.g. particle concentrations, heat, gas amounts) and vector fields.

We allow r to be chosen and set arbitrarily. However, note that the effectiveness of later matching will be governed by an interplay between r and the spacing between

the $\vec{x_i}$. In practice, we typically use a uniform grid of $\vec{x_i}$, and in order to obtain a good coverage of the simulation domain, we use a single scale, r, such that nearby sample points overlap in influence (see figure 30).

b. Keyframe Storage

For a given time, t_j, we refer to the collection of match points and all sampled simulation data for those match points as a keyframe. This is a slightly different notion of "keyframe" from that of the traditional sense, but conveys a similar idea—namely that it is the information that we want to match. Each keyframe records the following information:

- *Time* (t_j) for the keyframe sample.
- *Weight* ($m^r_{i,j}$) for each of the match points.
- *Position and scale* (x_i, r) of each set of sampled variables.
- *Keyframed simulation variables* ($d^r_i, \vec{D}^r_{i,1}, \vec{D}^r_{i,2}$) such as density values, heat, and flow fields.

3. Matching the Final

In the matching process, we sample the final (unmatched) simulation at the same points specified in the keyframe. The difference between the final and the keyframe will be used to modify (match) the final. It is important to first understand the simulation structure.

a. Flow Equations

As has been discussed in appendix A, the inviscid Euler equations for incompressible flow u are

$$\nabla \cdot u = 0 \tag{5.6}$$

$$\frac{\partial u}{\partial t} = f - (u \cdot \nabla)u - \nabla p \tag{5.7}$$

and density d advected by the flow is defined as

$$\frac{\partial d}{\partial t} = S - (u \cdot \nabla)d + k_d \nabla^2 d \tag{5.8}$$

Here, f accounts for external forces and S defines a density source (or sink). More detailed explanation is given appendix A.

Instead of changing the governing equations or adding additional controls, we opt to control the equations using external forces and density sources at the data collection locations. That is, we will define f and S in order to obtain a match. Our use of wind forces and density sources prevent physical inaccuracies, such as negative pressure. Just as importantly, the basic simulation computation is unchanged from an "uncontrolled" simulation. This makes our approach very easy to integrate into an existing simulation framework.

b. Matching

The matching is done in two passes. In the first pass, the simulation is sampled at the x_i and scale(s) r. For scalar data we obtain the (weighted) current value $\overline{d_i^r}$:

$$\overline{d_i^r} = \frac{1}{\sum G_r} \sum_x G_r(\vec{x_i}, x) d(\vec{x}, t) \tag{5.9}$$

To provide matching, all we need to do is to specify the source S for that scalar variable:

$$S(\vec{x}, t) = \max_{i,r}(W(t_j, t) G'_r(\vec{x_i}, \vec{x}) m^r_{i,j}(d^r_i - \overline{d^r_i})) \tag{5.10}$$

The max function should be taken to mean the maximum absolute value (maintaining the sign); the reasoning is explained below. Note that we use two functions to spread data, a spatial one, G'_r, and a temporal one, W. Both of these are discussed below (sections c and d respectively).

Similar to sources controlling densities, we use external forces (f from Eq. 5.7) to control the flow. $\overline{\vec{D}^r_{i,1}}$ and $\overline{\vec{D}^r_{i,2}}$ are computed in the obvious way (see equations 5.4, 5.5, 5.9)

$$f(\vec{x}, t) = \max_{i,r}(W(t_j, t) G'_r(\vec{x_i}, \vec{x}) m^r_{i,j}(f_l + f_v)) \tag{5.11}$$

$$f_l(\vec{x}, t) = \vec{D}^r_{i,1} - \overline{\vec{D}^r_{i,1}} \tag{5.12}$$

$$f_v(\vec{x}, t) = (\vec{D}^r_{i,2} - \overline{\vec{D}^r_{i,2}}) \times (\vec{x_i} - \vec{x}) \tag{5.13}$$

Here the max function should be taken to mean the vector with maximum magnitude.

The max functions that appear in equations 5.10 and 5.11 are chosen to prevent overcompensation during matching. Notice that any point can fall within the influence region (i.e. within the G'_r radius) of several match points. These separate controls may give differing or even conflicting information. For example, one match point

might want to increase density, and another to decrease it. Or, two points might both want to increase the density. If we were to sum these contributions, we may add far too much density. An averaging process is feasible, but would likely overweight less important additions (e.g. a point near the center of one match point could get minimized by points from the fringes of other match points). The "ideal" would be to formulate this as a more complicated optimization problem, but this would require several iterations with no guarantee of convergence. Instead, we compromise with a method that provides convincing results, with far less work, by simply taking the maximum (magnitude) contribution from any of the sources. In this way, we are guaranteed to maintain the most important changes, while reducing chances of overcompensation (Fig. 33).

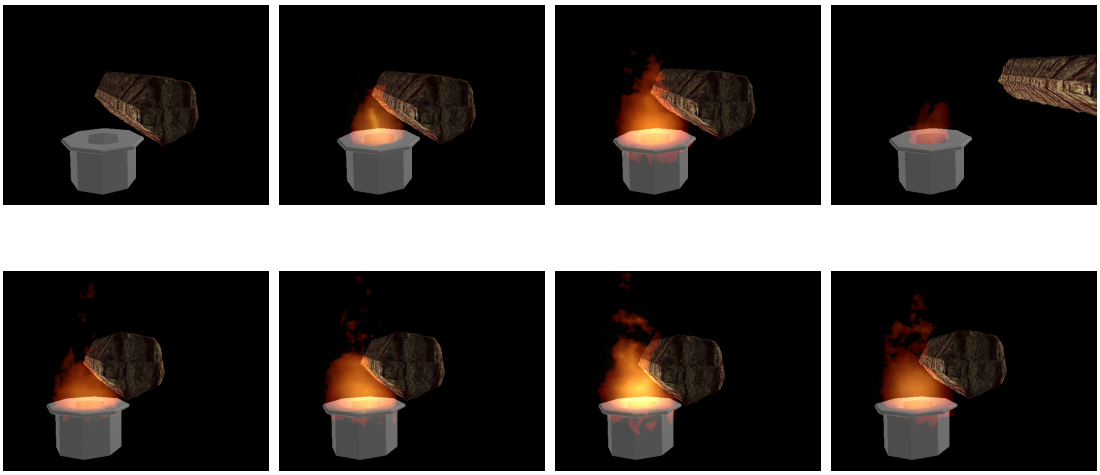

Fig. 33. Stress testing the matching process. Simulations produced strictly from the matching process. Fire data is sampled in the preview. The burner is turned off in the final simulation, and all fire is created from the matching function. The top row shows the final with "burst" control, and the bottom row shows the final with "continuous" control.

c. Spatial Blending

The spatial blending function, G'_r, spreads a factor of the difference value over a region. We use the same Gaussian weighting function as for sampling. The multiplicative factor is one over the sum of the squares of the Gaussian weights. The intuition behind this choice is described here.

Assume we have a difference Q between the sampled density of the preview and the final at some point. In order to make the final match the preview, the sampled final value must therefore increase by Q. That is, we need to have

$$Q = \sum_{\vec{x}} G_r(\vec{x}_i, \vec{x}) Q'_{\vec{x}} \tag{5.14}$$

where $Q'_{\vec{x}}$ is the increase in density at point $\vec{x}$. One way to do this is to increase the density value of each point contributing to the sample by Q—i.e. $Q'_{\vec{x}} = Q$. However, if the sample point is of a large scale (high r), this can involve a very large number of points. Adding Q to all of them introduces a significant amount of overall density to the scene (roughly Qr^2), and also affects points far away from the sample as much as it affects the sample itself. This is undesirable.

We would prefer to instead distribute any additional density closer to the sample point (where it will have more effect), but still want to spread the additional density smoothly, to avoid artifacts. An obvious choice (but not the only one) is to use a Gaussian spread function, of radius equal to the scale we aim for. That is, we have $Q'_{\vec{x}} = G_r(\vec{x}_i, \vec{x})Q''$. We now must find Q'', the amount to apply via a Gaussian spread to result in the correct overall change, Q. Substituting into equation 5.14, we have:

$$Q = \sum_{\vec{x}} (G_r(\vec{x}_i, \vec{x}))^2 Q'' \tag{5.15}$$

Solving for Q'' gives us the correct weighting to use, yielding the final spatial blending function:

$$G'_r(\vec{x}_i, \vec{x}) = \frac{G_r(\vec{x}_i, \vec{x})}{\sum_{\vec{x}} (G_r(\vec{x}_i, \vec{x}))^2} \tag{5.16}$$

d. Temporal Blending

Since keyframes are defined at only a point in time, if we perform matching only at the keyframe times, we would have a sudden introduction of density/force into the system. This would be extremely unrealistic and jarring, and therefore we choose to gradually match the simulation state incrementally, starting just before the keyframe time. This incremental matching enables us to use "dumb" sources/sinks, rather than running an optimization procedure to obtain an exact control solution, as is done in most keyframe matching approaches. As a result, we introduce a temporal spread function, W, to gradually match keyframe data.

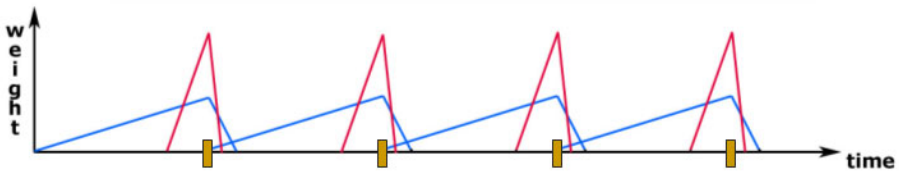

Fig. 34. Two possible weight functions. The blue curve provides continuous control, while the red curve provides "burst" control. The yellow markers are keyframed points during preview.

W is defined as a function (e.g. piecewise-linear or Gaussian) that gradually increases over a number of timesteps (t_s), peaks at the actual keyframe time t_i, and

falls off sharply for some time afterwards (t_e). For example, a linear temporal blending function would have the form:

$$W(t_i, t) = \begin{cases} \mid t - (t_i - t_s) \mid, & (t_i - t_s) < t < t_i \\ \mid (t_i + t_e) - t \mid, & t_i \leq t < (t_i + t_e) \\ 0, & \text{o.w.} \end{cases} \tag{5.17}$$

By changing t_s and t_e we can vary from "burst" like keyframe control to "continuous" control (see figure 34). A burst control minimizes control interference, but creates sharp potentially visible changes in the simulation state. Still, it is applicable to a reactive simulation, such as fire simulation.

C. Controlling Fire Simulation

As mentioned previously, part of our motivation is in the control of complex simulations. One such complex fluid based simulation is fire. We have implemented such a simulation by integrating a fluid-based flame simulator, combustion model, heat transfer model, solid pyrolysis model, solid decomposition model, and rigid body simulator into a common framework. Using coarse resolution simulation modules we were able to achieve somewhat interactive speeds (7 fps for 30x30x30 resolution on a 1.8 GHz P4).

1. Implementation and Results

Using our fire simulation framework, we record keyframes from a coarse simulation that runs at interactive rates. We place data points in a regular grid across the entire simulation environment, recording user interactions and taking keyframe data. The amounts of fuel gas, smoke, and heat are keyframed as density values, and air motion

is keyframed as a vector field.

We achieved successful matching of the final simulation to the preview under a variety of example cases. This included cases where the behavior of the unmatched system was very different from that of the preview (and matched final).

Due to the number of competing variables, particularly within the simulation model itself, comprehensive performance analysis is not possible. However, based on our testing, we are able to make several observations regarding performance:

- The additional overhead added in the preview simulation due to keyframe generation was less than 5% of the simulation time in our experiments.

- The additional overhead for keyframe fitting during the final simulation was 10-30 % of the simulation time in our tests. Generally, for a fixed set of data collection points in a keyframe, as the resolution of the final simulation increases, the additional cost of matching decreases. This is to be expected, in that the keyframe matching time grows relatively slowly compared to the additional simulation cost on a larger grid.

- Using too sparse of a data collection point resolution tended to give poor matching results. This will vary, of course, based on the absolute resolution of the preview, and the relative increase in resolution for the final. In effect, the control is diffused over too wide a region.

- Finally, relatively lower resolution preview grids tended to yield larger matching overhead during the final simulation. This is due to the fact that as the resolutions become much more disparate, the simulations become less and less similar.

Fig. 35. Simulation control in action. During the coarse resolution interactive simulation (top row), the user ignited a match and burned the conference logo. During higher resolution simulation (second row), the match barely lit, hence the logo did not burn at all. The controlled simulation (last row) injected additional combustible gas and heat into the system, and thus burned in a manner similar to that in the preview simulation. Note that the simulation control matches the fluid domain only, and hence does not have direct control on how the logo burns and decomposes.

Figure 35 shows the user igniting a match in the preview, and than proceeding to burn a SCA logo. A 30x30x30 interactive simulation is used for the preview. An uncontrolled simulation running at 60x60x60 resolution causes the match to burn in a slightly different fashion. At first, the match does not burn strongly enough to ignite the logo, and although later the match bursts into flame it is not enough to ignite the logo. The controlled simulation uses half second separated keyframes using 10x10x10 uniformly placed data collection points. The controlled simulation more closely matches the preview, creating additional heat sufficient to ignite the logo.

Figure 33 shows a stress test on the method. In this scene, we ignite a burner, and the user puts a log into the flame, and the log starts to burn. For the stress test, we turn off the burner. Without the burner, the log would not burn. If we enable matching and use burst control with only a single data collection point placed at the burner, one can observe the control process in action as bursts of fireballs occur at keyframe intervals. If we use a temporally wider control, the burner appears to be burning.

D. Parameters and Control

We briefly summarize here the key parameters that govern the simulation matching behavior; many are referenced in the previous sections.

- *Scale and Density of Match Points.* The resolution of both the final simulation and the preview simulation, as well as the size of the simulation features we want to match, determines the scale and number of the match points we should use for sampling.

- *Temporal Resolution.* Controls on the temporal resolution of both the keyframe capture (t_j) and the temporal spread function (W) used during matching can

have a significant effect. If the t_j are too far apart, the simulation may diverge to a point that control becomes obvious. The temporal spread function is important to adjust the speed at which matching occurs, and thus affects how obvious the control appears.

- *Threshold.* Not discussed here, but useful for a practical implementation, are thresholds that set a level for deviation from the keyframe, below which no matching will occur. This reduces unnecessary controls for nearly-matching simulations, and can reduce computation time.

E. Discussion

Note that the data collection points and keyframe matching step are defined locally in both space and time. This structure does not provide mass conservation of densities throughout the system. For reactive systems where densities should change over time, such as fire simulation, this is often desirable. In addition (as mentioned below), due to the source of our keyframes, any sourcing/sinking should be minor. For general purpose fluid simulations, mass conservation similar to that of Fattal and Lischinski [26] should be provided.

A second point to note is that we are not solving a (potentially quite large) linear system for the correct solution, rather we let the overlapping control points "fight" to fit the keyframes. As mentioned in the previous section, incremental corrections are small, and diluted over time, to keep the not-physically-based control plausible. In highly constrained conditions (i.e. lots of overlapping sample points), or degenerate cases such as fully overlapping sample regions, multiple data control points could compete against each other, potentially creating instability in the system. We have never observed such instability in any of our sample runs, however it is certainly

possible. By limiting the peak of the weight function, some such instabilities can be prevented. Considering a degenerate case with two fully overlapping Gaussians, we can prevent overcorrection by limiting the peak of the weight function to 0.5.

Our additive/subtractive scheme can create negative densities (i.e. removing density when there was none to begin with). This requires correction after adding/subtracting all sources affecting a point, but is a local operation.

One could argue that our additive/subtractive scheme is blurring the fine-scale detail we are trying to generate. Additional sourced density does tend to blur the simulation results, in that it is added evenly over a region, thus reducing contrast in (though not eliminating) fine-scale detail. However, note that stronger fitting (i.e. more sourced density) only occurs when the simulations diverge significantly, which is neither typical nor expected. Also, nothing about our method prevents the use of other common methods for "faking" fine-scale detail, such as warped blobs, or advected textures.

Finally, we use the "Stable Fluids" (appendix A) model, which favors stability over accuracy, and coarse resolution simulations suffer from artificial viscosity and dissipation. One could argue that by matching to a coarse simulation we are losing all the benefits of using high resolution simulation. We would like to point out that matching is performed only at a gross level, using an even lower resolution than the coarse simulation for matching, and guaranteeing that the results match only in the aggregate. In the sample case seen in figure 35), the user ignites a match, and proceeds to burn the logo. In the uncontrolled case, the match does not ignite, hence during the rest of the simulation the logo is not burned. In the controlled case, we are able to push additional heat and combustible gas seamlessly to ignite the match and start the burning of the logo. In either case, the higher-resolution simulation contains significantly more fine-scale detail.

Our keyframe matching draws easy comparisons with prior methods for keyframing fluid simulations. For simple simulations (e.g. smoke only), one can imagine ways to incorporate our keyframe samples as input to other keyframing approaches, then use those approaches to perform the match. Indeed, those keyframing approaches have the potential of creating more plausible motion, in that they use more sophisticated keyframe matching techniques (e.g. optimization). However, there are two main points that we feel make our approach superior for our range of applications.

First, and perhaps most importantly, the prior keyframing methods can make very few assumptions regarding the status of the system from which they hope to hit the keyframe. On the other hand, we have a system that ought to behave in the same fashion as the system from which we derived our keyframes. Since the preview and final follow the same physically-based rules, and are matched in the previous keyframes, the two simulations should never diverge by much. Thus, the changes to the system to match our keyframes should be generally minor, and our simple sourcing scheme should suffice. We should achieve similar performance with our significantly simpler method.

Second, our system is better suited than prior approaches for fluid simulation systems with multiple densities. Other than Fattal and Lischinski's method [26], earlier methods achieved most of their success by modifying the vector field; this is not a feasible approach for several applications, particularly with multiple density fields. Also, our method works fine for reactive systems, while prior approaches have to enforce a conservation of mass, or allow only limited sourcing/sinking.

On the other hand, systems requiring mass conservation would perform more poorly in our system. As an example, rising smoke could disappear in one location and appear somewhere else in order to fit our preview. We can easily incorporate global mass conservation in our approach, but usually it is local mass conservation that is

desired. Increasing the frequency of keyframes will minimize the visual distraction, yet adding too much control in this manner will tend to blur the simulation *temporally* and defeat some of the purpose of using a high resolution simulation.

Ideally, we would like to have a mathematical model to compensate for the different behavior of the simulations using the characteristics of the underlying fluid equations. Until such intelligent methods are proposed, the simulation matching approach presented in this chapter is trivial to add into fluid systems. No change in the broad simulation structure is required, and the computations, including Gaussian sampling and introducing new sources and sinks, are easy to implement. This method is much simpler than prior keyframing approaches, and yet works well in environments with an assumption that only minor changes to the simulation will be required. Also, no implicit knowledge is required about the simulation, it could be applied to a variety of fluid simulations, and any number of important variables could be matched.

Although most of the shortcomings of this method are addressed here, a very important assumption we make is that a physically based interactive simulation is provided for user interaction and data collection. In our fire simulation, we used a very simple flame model at a very coarse resolution (30^3 grid), that achieves interactive rates (5-7 fps on a 1.8 GHz P-IV) on very simple scenes (one flame source and one wood piece). Increasing either scene size or scene complexity will reduce interactivity.

CHAPTER VI

VISUALIZATION

In the previous chapters, we present a physically based fire model that is used to generate simulation key frames at interactive rates. In this chapter we present GPU based techniques to render the flames and create visual details and in-between frames to achieve real-time frame rates (Fig. 36).

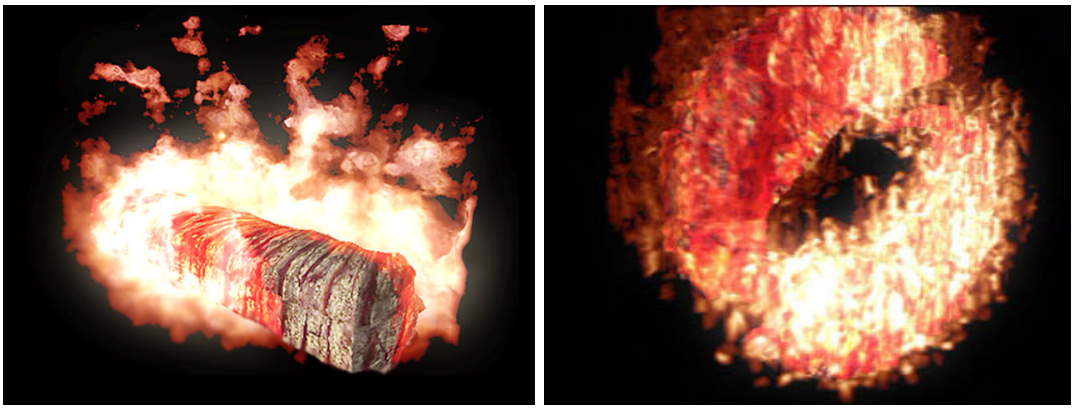

Fig. 36. Burning a log and a Siggraph logo. On the right, a log is in flames, decomposing. The Siggraph logo is burning on the left.

A. Background

Rendering flames is usually done by either using particles or using volumetric techniques. Lamorlette et al. [60] use a statistical approach to generate particles. Nguyen et al. [80] model black body radiation from the flame and use Monte Carlo ray tracing approach to render the flame volume, simulating light transport in a participating medium. Interactive applications on the other hand use particles [100], textured polygonal primitives [18][117], or procedurally generated models [90].

Different techniques have been proposed to increase visual detail during render-

ing. Stam and Fiume [111] use a map to define the amount of fuel and temperature on every point on the object, and use turbulent wind fields and warped blobs. Neyret [79] proposes adding fine detail using advected textures. Losasso et al. [65] use an octree based simulation structure for liquid and smoke simulation; such a method could presumably be used for fire. Selle et al. [103] incorporate vortex particles to reintroduce small scale detail for simulating smoke, fire and explosions, addressing the limitations of vorticity confinement [112][28] on low resolution grids.

To break up the visual artifacts due to the coarse resolution motion fields, several methods are proposed. One such method is to generate a randomized energy spectrum in the Fourier domain. By taking the inverse FFT one forms a vector field, which in turn is combined with the simulated flow field to add turbulence. The turbulent motion is controlled by the energy spectrum used to generate the random field. Stam and Fiume [110] introduced the Kolmogorov energy spectrum to the CG community. They used energy cascading, where the energy introduced at some frequency $k_{initial}$ is propagated to higher frequencies at a constant rate. The Kolmogorov energy spectrum is also used by various other authors [60][99]. Perlin and Neyret [89] proposed the use of flow noise. Flow noise is generated by using correlated rotations and advection of small scales by large scales. Hong et al. [48] uses Detonation Shock Dynamics to generate self sustaining cellular patterns on the simulated flame surface. Bridson et al. [10] proposed curl noise, using Perlin noise and the Kolmogorov spectrum in the vorticity domain to generate procedural fluid flow.

B. Volumetric Rendering

All the data is kept in some sort of volumetric form, hence a straightforward rendering approach will use volumetric techniques. Our visualization steps are as follows:

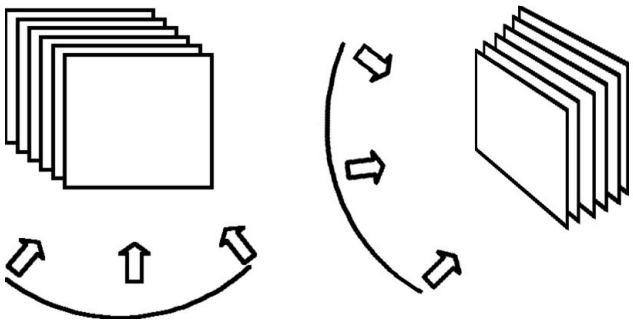

Fig. 37. Two sets of quadrilateral stacks are used for different camera angles.

- A 3D texture is generated using gas densities and heat distribution.
- A set of quadrilaterals are rendered from from back to front.
- GPU based visual effects such as color correction, bloom and noise are added using fragment shaders.
- A 4D noise function is used to generate additional frames, called "inbetween frames", until the next one is ready from the simulation. Textures from the consecutive simulation frames are linearly interpolated for a seamless transition.

The first step is to generate a 3D texture from the simulation data.

Instead of transferring all the simulation data to the GPU, we opt to form the texture on the CPU, to be used as a starting point for flame visualization, to minimize the bottleneck of CPU to GPU transfer. Here we used a simple function, found empirically. By trial we found this function to approximate real flames. There are more advanced rendering techniques, such as [80], taking black body radiation and participating media into account, but these costlier methods are reserved for offline applications. Here note that, we are going to apply color correction, and additional effects on the GPU, and this "initial texture" is just a starting point for flame visualization. The function is as follows:

$$\vec{C_f} = \begin{pmatrix} 1 \\ 1 \\ 0.6 \end{pmatrix} \vec{C_s} = \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \end{pmatrix} \vec{C_b} = \begin{pmatrix} 1 \\ 0.4 \\ 0 \end{pmatrix} \tag{6.1}$$

$$\vec{C} = T * a_f * \vec{C_f} + a_s * \vec{C_s} + C * \vec{C_b} \tag{6.2}$$

where $\vec{C_f}$, $\vec{C_s}$ and $\vec{C_b}$ are colors associated with fuel gas, smoke and combustion areas respectively. Fuel gas emits a yellow color and its intensity varies by both temperature and fuel density. The reaction zone where the chemical combustion occurs, emits a red color, giving the flames the distinct red contour. The smoke forming from the combustion reaction gives a dark gray to black color.

For interactive volume visualization a polygon stack is rendered from back to front (Fig. 37) using the 3D texture. The number of slices determines the quality of the rendering, independent from the simulation. The refresh rate will depend on the GPU only. The standard practice is to place the slices perpendicular to the viewing direction. Here, we opt to use two sets of predefined slices, placed orthogonal to each other, and switch at the $\frac{\Pi}{4}$ degree corners. The switch is seamless, and does not require rotating the slices to face the camera for each refresh. This saves a small amount of GPU-CPU bandwidth if the rotation is performed on the CPU, or execution of a vertex shader, if rotation is performed on the GPU. Note that a third set is needed for the up/down rotation.

C. GPU Rendering

With current hardware, only coarse resolution simulations can be handled at interactive rates. To increase visual detail, Perlin noise is used to warp the 3D texture.

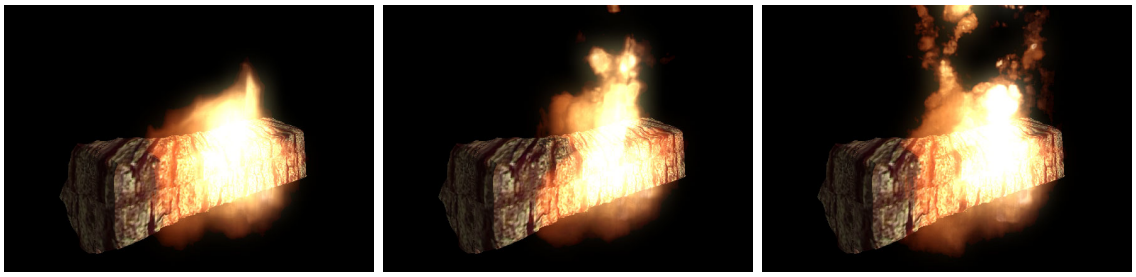

Fig. 38. Increasing the number of octaves increases fine scale features.

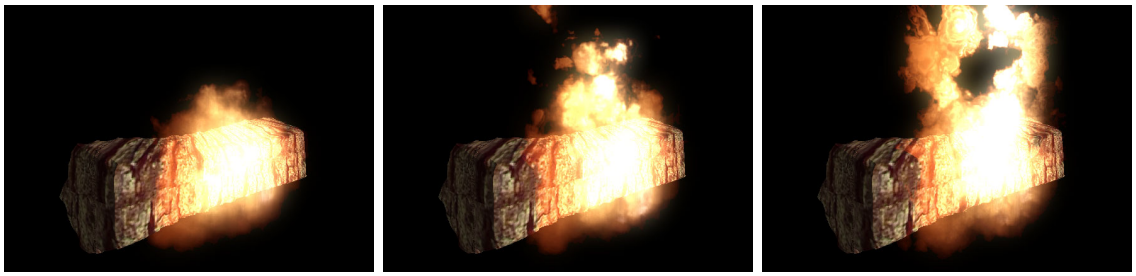

Fig. 39. Increasing the turbulence increases the height of the flames.

Since current graphics hardware does not support random number generation, a 3D base noise texture is created. Perlin noise [90][91] is generated using 4 octaves using this texture using the random noise function [120][82]. Per pixel texture coordinates are translated using the following equation:

$$y+ = mask_{sources} * mask_{objects} * S_{turbulance} * P(\vec{x}, o_p); \tag{6.3}$$

where $P(\vec{x}, o_p)$ is the Perlin noise at location $\vec{x}$ using o_p octaves. Increasing the number of octaves adds more fine scale features (Fig. 38). $S_{turbulance}$ is used to control the maximum amount of translation of texture coordinates (Fig. 39). Here, note that we only translate y-coordinates of the texture space. The mask terms $mask_{sources}$ and $mask_{objects}$ are used to keep the flames attached to the sources and objects re-

spectively. Additional fine scale visual detail could be generated using an additional vector field for turbulence, such as Kolmogorov noise [60] [99].

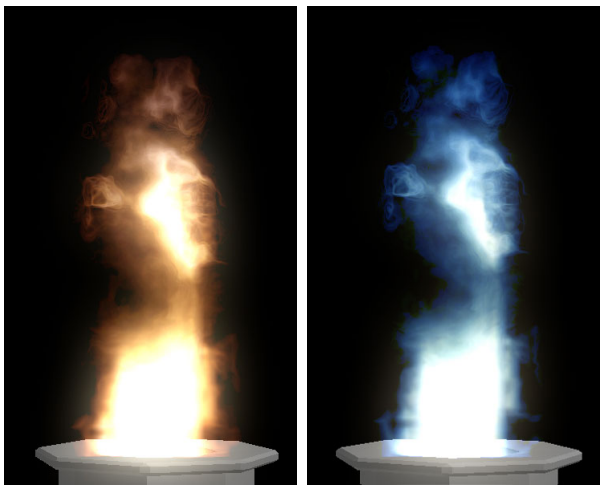

Fig. 40. Coloring the flame.

Although the color textures generated on the CPU give a flame-like color scheme, it is just a starting point. First the flame is rendered into a texture and color correction and interactive fine tuning is computed in a fragment shader. We provide RGB color bias and brightness/contrast controls, and with real-time feedback the user can go from realistic looking camp fires to blue colored gas burner flames (Fig. 40).

Another effect is bloom, and it is used to increase the illusion of a very bright flame. The flame texture is blurred in two passes (horizontal and vertical passes), and decreased in size. A cutoff threshold is used to blur only certain bright areas (Fig. 41).

Here are the passes made during the visualization of flames:

- First, the burning solids are rendered, and copied from the framebuffer into the object texture.
- Using the simulation data, a 3D initial texture is generated at the CPU.

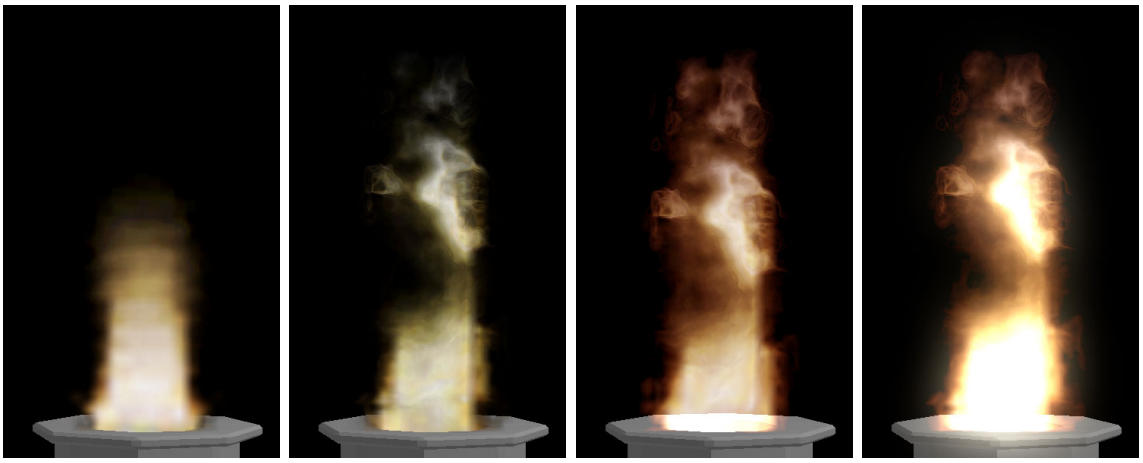

Fig. 41. Flame rendering steps. At left is pure OpenGL rendering. The next one has the 3D texture shifted per pixel using Perlin noise. The third one is color corrected in the fragment shader. The last step adds bloom to the image.

- Volumetric rendering of flames is performed using the initial texture. Texture coordinates are translated using the Perlin noise base texture, generated only when the user changes noise parameters. Solid objects are rendered as black-holes. The framebuffer is copied into the prerendered texture.

- The prerendered texture generated at the last step is used in an image processing step using fragment shader. In two passes (first in the x, then in the y direction) the image is filtered. The framebuffer is copied into the bloom texture.

- The last step is the composition step combining all textures. A prerendered texture is used for color correction and brightness/contrast correction. Then, the bloom texture is added on top. The object texture is added to complete the composition process.

D. GPU Inbetweening

To achieve real-time refresh rates, one can opt for either using a simplified simulation or using extremely coarse resolution simulations. To achieve a balance between performance and detail, we decided to use resolutions that can run at interactive frame rates. On a P4 1.8 GHz, we can create a 20^3 simulation at 5 fps, or on a Core2Duo 2GHz, we can create a 30^3 simulation at 4 fps. We call these "simulation frames." As the next simulation frame is being generated, the GPU is used to keep the flames alive using a 4D Perlin noise. We call these frames "inbetween frames" (Fig. 42).

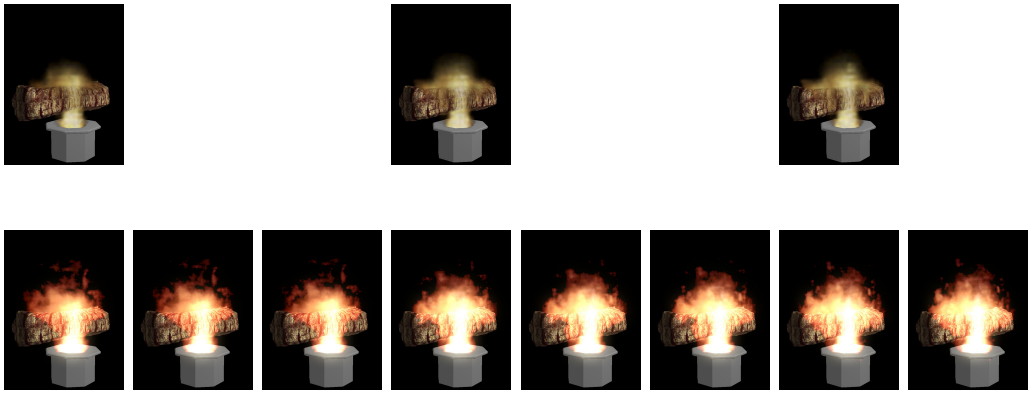

Fig. 42. Inbetween frames generated on the GPU. Top row is the simulation frames, and bottom row is the inbetween frames.

To generate inbetween frames, we used 4D Perlin noise, using the time as the fourth parameter. For the desired frame rate, the simulation frames are delayed, if they arrive early. If the simulation frames arrive more slowly, then the visualization is stopped until the next simulation frame is ready. This scheme works for recording video, and there are no visible "popping" artifacts. For interactive applications, we opt to use a free frame rate approach, where we do not delay or stop the visualization, such that the user always watches a moving flame. Note that this approach generates a varying frame rate throughout the visualization.

For a seamless switch, we interpolate the simulation frame with the previous one. We allow the user to move the objects as the simulation frame is still being simulated, but delay the combustible gas release until the end of the simulation pass such that the emitted gas does not show at the previous location of the object.

E. Rendering Burning Solid

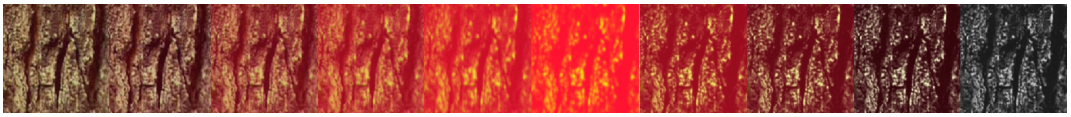

Fig. 43. Texture strip used for different stages of pyrolysis.

The distance field representation is polygonalized by an isosurface generation using tetrahedral decomposition of the boundary cells [81] similar to marching cubes [64]. This method generates a polygon soup, and additional pointers are used to keep track of which simulation cell the generated triangles belong to, and merge the corresponding ones. After each timestep, previous visualization polygons are discarded and new ones are created.

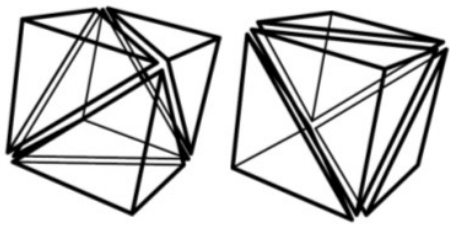

Fig. 44. Two orientations used for the tetrahedral decomposition.

This working scheme simplifies the implementation, yet the lack of frame-to-frame tracking prevents consistent texture coordinate inheritance during the interactive simulation. Instead, we use projection methods to map texture coordinates from

the implicit form onto the visualization polygons.

The faces are tagged with the stage of pyrolysis they are in, by using the centroid to check what percentage of the solid fuel initially contained has been used. This information is used to select a gradually changing texture for the face.

We prepared a texture strip (Fig. 43), where the wood color first gets red to indicate the flame front, and later it turns to black indicating the burned wood [1]. Although we do not explicitly track the flame front like some other models [92][7], our results visualize the flame front successfully because of the underlying physical modeling.

F. Matching Flames and Solids

Fig. 45. Matching the flame to the object. On the left, the turbulence is too low to generate separating flames. In the center image, the turbulence is high enough to create separating flames, but the flame looks like it is not attached to the object. On the right, the object mask is applied on the turbulence parameter, and we get separating flames farther from the object, and the flame is tight around the object.

The texture advection method generates separating flames and adds fine scale visual detail. When combined with the objects and burners (fuel gas sources), the separation of flames at the object boundary generates floating flames. What we would

[1]Thanks to Erin Devoy, she had the original idea and tested for its applicability.

like to have is separating flames far from the burning object, but a tighter fit around the object (Fig. 45).

Fig. 46. Generating the object mask. A binary alpha mask is generated from the objects, and than smoothed for soft blending.

Two masks defined in Eq. 6.3 are used to control the amount of texture translation (Figures 46 and 47). The first mask, $mask_{sources}$, can be replaced with y^2, since the texture coordinates range from 0 at the bottom and 1 at the top of the simulation domain.

The second mask, $mask_{objects}$, requires some processing. First, a black and white alpha mask of the filled/unfilled space is created. Since the flame simulator updates a filled/unfilled mask in each timestep during the synchronization step (pyrolysis), we can use this data to generate the mask. Note that we do not need a high resolution texture. Second, the texture is processed in layers, and filtered in the x, y and z directions in three passes. This will create a soft blending mask.

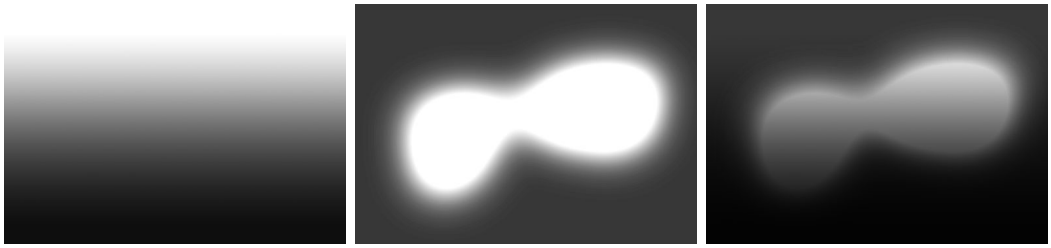

Fig. 47. Blending the source and object masks.

G. User Interaction: Flame Painter

One sample application we implemented to demonstrate the capabilities of the interactive simulation framework is the "Flame Painter". Here the user can click on the solid object to ignite it (Fig. 48). During the simulation, the user can interact at any time to expand the flames and choreograph how the object is going to burn.

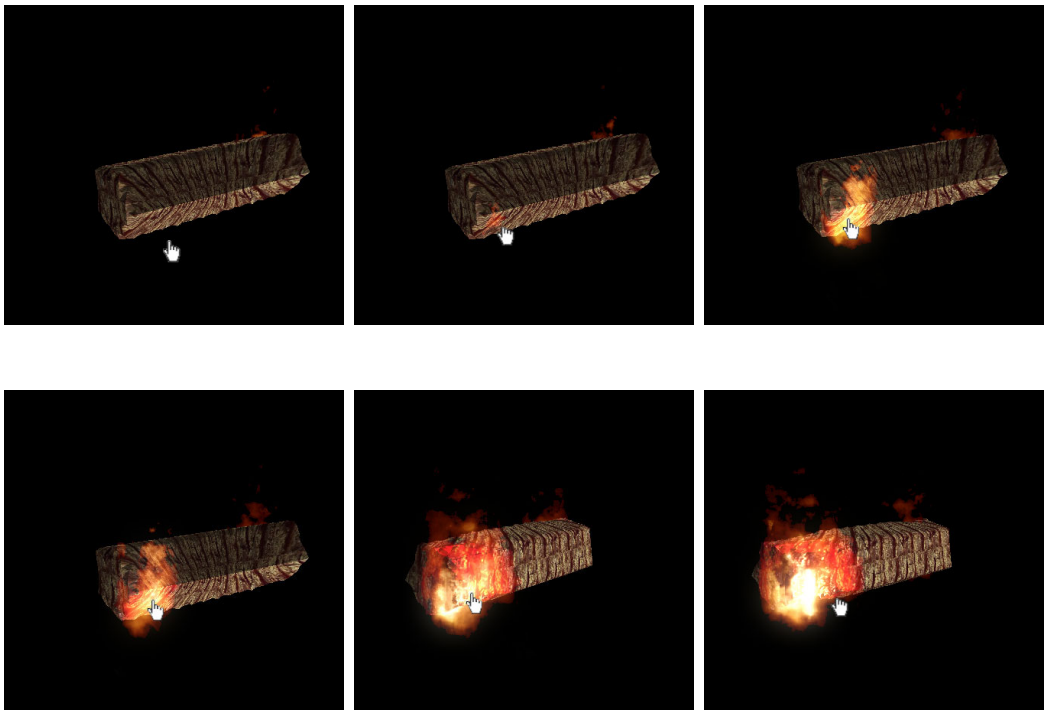

Fig. 48. Flame painter interface. The user controls where and when the object starts to burn.

In our implementation we tried to keep the physical part of the simulation, and decided only to pump heat into the object at the selected location, which in turn ignites the object. When the user clicks anywhere on the object, we use the OpenGL selection buffer to find the corresponding triangle. The triangle centroid is used to inject heat into the system. This additional heat causes the object to enter pyrolysis from the selected location, and it starts emitting combustible gas. The emitted gas

ignites, since we already injected enough heat to start the combustion reaction. This combustion reaction generates more heat, and if the object is volatile enough, it will keep burning by itself.

H. Results

Although fluid simulations using just the GPU look promising [46], there is always a trade-off between accuracy and speed just like in CPU implementations. One can choose to use lower resolution simulation to achieve real-time simulation, or use higher resolution at slower simulation frame rates. There is nothing preventing us from using a GPU based fluid solver, and still use another GPU for inbetweening, as we described here, to achieve real-time visual feedback. The method presented here is useful to keep the user experience realtime, while maintaining additional accuracy.

One drawback of our visualization scheme worth discussing here is the fact that we treat flames separate from the burning object. A series of methods [60][92] plant flames on the surface of the burning object, which makes the flames move together with the moving object. In our approach, we emit combustible fuel gas into the system, and once released, it is not related to the object anymore. At slow simulation refresh rates, one can move the objects around, but the flame will stay in its previous position, until the simulation updates itself in the next simulation frame. By updating the solid object's position just before the pyrolysis stage, we get the simulation update itself one frame earlier than a direct implementation. Still, the user has to wait until the current simulation frame is finished and the visuals are updated.

CHAPTER VII

CONCLUSION

In this dissertation, we highlighted a gap within the CG community for the simulation of fire. There has not previously been a physically based yet interactive simulation for fire. None of the models to date, interactive or not, have been able to address everything related to fire and burning in one unified framework. This dissertation describes a unified simulation framework for physically based simulation of fire and burning.

A. Summary of Results

In this work we bridge this gap and provide a complete simulation framework, together with physically based modules, working at interactive rates. We described our simulation framework, together with implementations of the modules addressing different parts of the related phenomena. Our results show that our implementation can model fire, objects catching fire, burning objects, decomposition of burning objects, and additional secondary deformations. The results are plausible even at interactive frame rates, and controllable. The graphics processor is used to add inbetween frames to achieve even faster (real-time) refresh rates, as well additional visual detail.

Physically based simulations might be hard to control, since many physical conditions should be set for a desired behavior, and it might not always be possible to do so. We described a preview based simulation control, which allows interactive feedback to the user during a coarse simulation, while the fine-scale simulation will be guaranteed to behave as observed during the preview. Our preview based control framework could be used as a training or design tool, or as a part of creative applications. Our results demonstrate the working of the simulation framework.

B. Future Research

There are many open avenues for future research. In this section, we will discuss some of the major areas open to further study.

1. Scaling

Our implementation demonstrates small fires, burning logs or matches, and even campfires, but to simulate a larger scene such as burning houses, forests or, a bonfire requires higher resolution grids. Our method is scalable but will lose the interactive rates when high resolution grids are used. A large scale fire at interactive rates requires extending our method in one or more potential ways:

- Recent research addresses fluid simulations combined with a space partitioning technique, such as octrees [65].
- Since the fluid solution is Galilean invariant [105], one can move the simulation grid to encapsulate the important region of simulation, e.g. a bounding box, and discard the rest. Multiple fires at spatially different places could possibly run in separate encapsulations, but handling boundary conditions, and merging/splitting those simulations requires additional research.

2. Interaction

Although our implementation demonstrates that physically based simulation of fire is possible at interactive rates, we have not investigated potential interaction tools. A sample user controlled flame painter (Section G) is provided to investigate the potential, but a complete toolset and UI requires an additional layer on top of our provided framework.

3. Visualization

Fire visualization is done using volumetric rendering. Visual detail is added on the GPU using a noise function, but there is always more potential to enhance the visualization. The Kolmogorov noise has been popular with offline fire and explosion rendering [60][99], and worth further investigation. Another problem is that volume visualization of flames is not connected to the object itself, and one can move the object, leaving the flames there. The simulation will eventually catch up, dissipating the flame in the old location, and creating a new flame in the new location. The separation of visualization and simulation frame rates does make things even worse. Here, one can extend the visualization such that the flames from the old location are transformed to the new one, until the simulation catches up. Another possible solution includes using a ghost object and delaying the translation of the burning solid until the simulation catches up.

4. Implementation

There are many ways our implementation could be extended:

- In this dissertation we described a unified simulation framework with sample implementations of different modules. Any one of the modules could be changed to a more accurate or to a faster (and potentially less accurate) module depending on the requirements.

- Complete rigid body handling is beyond the scope of this dissertation. Our rigid body implementation is minimal, and provided as a proof of concept at best. It is provided such that we can prove that rigid body handling using an implicit-explicit definition of the solid object is possible.

- Although we take advantage of the graphics hardware, we only used it for visualization purposes. Current research address fluid simulations on the GPU, and recent hardware improvements look promising for a complete fire and burning implementation on the GPU.

C. Hypothesis

The hypothesis of this dissertation is as follows

> *One can simulate different aspects of the burning process in a unified physically based framework by taking advantage of some simplifying assumptions and achieve visually plausible results at interactive rates on current hardware. More accurate simulations could be guided to have global behavior similar to the interactive results.*

The hypothesis is proved as follows:

- Physically based simulation of fire can be done at interactive rates (Chapter II).
- Different aspects of burning objects can be simulated (Chapter III) within a unified simulation framework (Chapter IV).
- Interactive coarse resolution simulation can be used as a "preview", and high resolution simulation can be guided to ensure similar behaviour as previewed (Chapter V).
- Even on regular hardware, not only are interactive simulation rates possible, but one can also take advantage of programmable graphics hardware to achieve real-time visual feedback for increased plausibility (Chapter VI).

REFERENCES

[1] A. Angelidis, F. Neyret, K. Singh, and D. Nowrouzezahrai. A controllable, fast and stable basis for vortex based smoke simulation. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 25–32, 2006.

[2] J. A. Bærentzen. Octree-based volume sculpting. In *Proceedings of IEEE Visualization '98*, pages 9–12, 1998.

[3] J. A. Bærentzen and N. J. Christensen. Volume sculpting using the level-set method. In *Proceedings of Shape Modeling International '02*, page 175, 2002.

[4] A. W. Bargteil, T. G. Goktekin, J. F. O'Brien, and J. A. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics*, 25(1):19–38, 2006.

[5] A. H. Barr. Global and local deformations of solid primitives. In *Proceedings of ACM SIGGRAPH '84*, pages 21–30, 1984.

[6] R. Barzel, J. F. Hughes, and D. Wood. Plausible motion simulation for computer graphics animation. In *Proceedings of Eurographics Workshop on Computer Animation and Simulation*, pages 183–197, 1996.

[7] P. Beaudin, S. Parquet, and P. Poulin. Realistic and controllable fire simulation. In *Proceedings of Graphics Interface '01*, pages 159–166, 2001.

[8] R. Borghi, M. Destriau, and G. D. Soete. *Combustion and Flames, Chemical and Physical Principles*. Editions Technip, Paris, 1995.

[9] P. T. Bremer, S. Porumbescu, F. Kuester, B. Hamann, K. I. Joy, and K.-L. Ma. Virtual clay modeling using adaptive distance fields. In H. R. Arambnia and et al. editors, *Proceedings of 2002 International Conference on Imaging Science, Systems, and Technology*, volume 1, 2002.

[10] R. Bridson, J. Hourihan, and M. Nordenstam. Curl noise for procedural fluid flow. *ACM Transactions on Graphics*, 26(3):46, 2007.

[11] R. Bukowski and C. Séquin. Interactive simulation of fire in virtual building environments. In *Proceedings of ACM SIGGRAPH '97*, pages 35–44, 1997.

[12] M. Carlson, P. Mucha, and G. Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Transactions on Graphics*, 23:377–384, August 2004.

[13] M. Carlson, P. J. Mucha, B. VanHorn, and G. Turk. Melting and flowing. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation*, pages 167–174, 2002.

[14] M. T. Carlson. *Rigid, melting, and flowing fluid*. PhD thesis, Georgia Institute of Technology, 2004. Director-Greg Turk and Peter J. Mucha.

[15] B. Chanclou, A. Luciani, and A. Habibi. Physical models of loose soils dynamically marked by a moving object. In *Proceedings of Computer Animation '96*, pages 27–35, 1996.

[16] Y. Chang and A. P. Rockwood. A generalized de casteljau approach to 3d free-form deformation. In *Proceedings of ACM SIGGRAPH '94*, pages 257–260, 1994.

[17] S. Chenney and D. A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of ACM SIGGRAPH '00*, pages 219–228. ACM, 2000.

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

[19] A. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 135(2):118–125, 1997.

[20] S. Coquillart. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *Proceedings of ACM SIGGRAPH '90*, pages 187–196, 1990.

[21] B. Daniel, O. Benoit, and R. Marcelo. On cfd and graphic animation for fire simulation. In *Eleventh Annual Conference of the CFD Society of Canada (CFD03)*, pages 28–30, 2003.

[22] M. Desbrun and M.-P. Gascuel. Smoothed particles: a new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*, pages 61–76, 1996.

[23] C. Detienne. *Physical Development of Natural and Criminal Fires*. Charles C. Thomas, Springfield, Illinois, 1994.

[24] G. Dewaele and M.-P. Cani. Interactive global and local deformations for virtual clay. In *Proceedings of Pacific Graphics '03*, page 131, October 2003.

[25] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. In *Proceedings of ACM SIGGRAPH '02*, pages 736–744, 2002.

[26] R. Fattal and D. Lischinski. Target-driven smoke animation. *ACM Transactions on Graphics*, 23(3):441–448, 2004.

[27] R. Fedkiw. *Geometric Level Sets in Imaging, Vision and Graphics*, chapter Simulating Natural Phenomena for Computer Graphics, pages 461–479. Springer Verlag, New York, 2003.

[28] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH '01*, pages 15–22, 2001.

[29] B. E. Feldman, J. F. O'Brien, and O. Arikan. Animating suspended particle explosions. In *Proceedings of ACM SIGGRAPH '03*, pages 708–715, 2003.

[30] B. E. Feldman, J. F. O'Brien, and B. M. Klingner. Animating gases with hybrid meshes. *ACM Transactions on Graphics*, 24(3):904–909, July 2005.

[31] B. E. Feldman, J. F. O'Brien, B. M. Klingner, and T. G. Goktekin. Fluids in deforming meshes. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 255–259, 2005.

[32] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

[33] M. S. Floater, G. Kos, and M. Reimers. Mean value coordinates in 3d. *Computer Aided Geometric Design*, 22:623–631, 2005.

[34] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proceedings of ACM SIGGRAPH '01*, pages 23–30, 2001.

[35] N. Foster and D. Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[36] N. Foster and D. Metaxas. Controlling fluid animation. In *Proceedings of Computer Graphics International*, pages 178–188, 1997.

[37] N. Foster and D. Metaxas. Modeling the motion of a hot turbulent gas. In *Proceedings of ACM SIGGRAPH '97*, pages 181–188, 1997.

[38] S. F. Frisken and R. N. Perry. A computationally efficient framework for modeling soft body impact. *Technical Report 2001-TR2001-11*, 2001.

[39] T. A. Galyean and J. F. Hughes. Sculpting: an interactive volumetric modeling technique. In *Proceedings of ACM SIGGRAPH '91*, volume 25, pages 267–274, 1991.

[40] M. N. Gamito, P. F. Lopes, and M. R. Gomes. Two-dimensional simulation of gaseous phenomena using vortex particles. In *Computer Animation and Simulation '95*, pages 2–15, 1995.

[41] M.-P. Gascuel, A. Verroust, and C. Puech. Animation and collisions between complex deformable bodies. *Graphics Interface '91*, pages 263–270, June 1991.

[42] T. G. Goktekin, A. W. Bargteil, and J. F. O'Brien. A method for animating viscoelastic fluids. In *Proceedings of ACM SIGGRAPH '04*, pages 463–468, Aug. 2004.

[43] E. Guendelman. *Physically-Based Simulation of Solids and Solid-Fluid Coupling*. PhD thesis, Stanford University, June 2006.

[44] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics*, 22(3):871–878, 2003.

[45] W. Hackbusch. *Multi-grid Methods and Applications*. Springer Verlag, New York, 1985.

[46] M. J. Harris, G. Coombe, T. Scheuermann, and A. Lastra. Physically-based visual simulation on graphics hardware. In *Proceedings of ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 109–118, 2002.

[47] J.-M. Hong and C.-H. Kim. Controlling fluid animation with geometric potential: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):147–157, 2004.

[48] J.-M. Hong, T. Shinar, and R. Fedkiw. Wrinkled flames and cellular patterns. In *ACM Transactions on Graphics*, 2007.

[49] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical rle level set: A compact and versatile deformable surface representation. *ACM Transactions on Graphics*, 25(1):151–175, 2006.

[50] I. Ihm, B. Kang, and D. Cha. Animation of reactive gaseous fluids through chemical kinetics. In *Proceedings of 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 203–212, 2004.

[51] M. Inakage. A simple model of flames. *Proceedings of Computer Graphics International*, pages 71–81, 1990.

[52] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. In *Proceedings of ACM SIGGRAPH '06*, pages 805–811, 2006.

[53] M. W. Jones. Melting objects. *Journal of WSCG*, 11(2):247–254, 2003.

[54] W. W. Jones, G. P. Forney, R. D. Peacock, and P. A. Reneke. *A Technical Reference for CFAST: An Engineering Tool for Estimating Fire and Smoke*

Transport. National Institute of Standards and Technology, Building and Fire Research Laboratory, Washington, 2000.

[55] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics*, 24(3):561–566, 2005.

[56] J. T. Kajiya and B. P. V. Herzen. Ray tracing volume densities. In *Proceedings of ACM SIGGRAPH '89*, volume 18, pages 165–174, 1984.

[57] Y. Kim, R. Machiraju, and D. Thompson. Path-based control of smoke simulations. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 33–42, 2006.

[58] B. M. Klingner, B. E. Feldman, N. Chentanez, and J. F. O'Brien. Fluid animation with dynamic meshes. *ACM Transactions on Graphics*, 25(3):820–825, July 2006.

[59] K. G. Kobayashi and K. Ootsubo. t-ffd: free-form deformation by using triangular mesh. In *Proceedings of ACM Symposium on Solid Modeling and Applications*, pages 226–234, 2003.

[60] A. Lamorlette and N. Foster. Structural modeling of natural flames. In *Proceedings of ACM SIGGRAPH '02*, pages 729–735, 2002.

[61] X. Li and J. M. Moshell. Modeling soil: Realtime dynamic models for soil slippage and manipulation. In *Proceedings of ACM SIGGRAPH '93*, pages 361–368, 1993.

[62] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, Sweden, 1994.

[63] I. Llamas, B. Kim, J. Gargus, J. Rossignac, and C. D. Shaw. Twister: a space-warp operator for the two-handed editing of 3d shapes. *ACM Transactions on Graphics*, 22(3):663–668, 2003.

[64] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of ACM SIGGRAPH '87*, volume 21, pages 163–169, July 1987.

[65] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics*, 23(3):457–462, 2004.

[66] F. Losasso, G. Irving, E. Guendelman, and R. Fedkiw. Melting and burning solids into liquids and gases. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):343–352, 2006.

[67] F. Losasso, T. Shinar, A. Selle, and R. Fedkiw. Multiple interacting liquids. *ACM Transactions on Graphics*, 25(3):812–819, 2006.

[68] D. Lundin. Motion simulation. In *Proceedings of Nicograph '84*, pages 2–10, 1984.

[69] D. Lundin. Works' ant. In *Special Issue: Fifteen Years of Computer Graphics 1979-1994*, page 100. ACM SIGGRAPH, 1994.

[70] R. MacCracken and K. I. Joy. Free-form deformations with lattices of arbitrary topology. In *Proceedings of ACM SIGGRAPH '96*, pages 181–188, 1996.

[71] K. T. McDonnell and H. Qin. Virtual clay: Haptics-based deformable solids of arbitrary topology. In *Proceedings of Second International Workshop on Articulated Motion and Deformable Objects*, pages 1–20, 2002.

[72] K. B. McGrattan, H. R. Baum, R. G. Rehm, A. Hamins, G. P. Forney, J. E. Floyd, S. Hostikka, and K. Prasad. Fire dynamics simulator technical reference guide. *Tech. Rep. NISTIR 6783*, 2002.

[73] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. *ACM Transactions on Graphics*, 23(3):449–456, 2004.

[74] Z. Melek and J. Keyser. Interactive simulation of fire. In *Proceedings of Pacific Graphics '02*, pages 431–432, 2002.

[75] Z. Melek and J. Keyser. Interactive simulation of burning objects. In *Proceedings of Pacific Graphics '03*, pages 462–466, 2003.

[76] Z. Melek and J. Keyser. Multi-representation interaction for physically based modeling. In *Proceedings ACM Symposium on Solid and Physical Modeling*, pages 187–196,299, 2005.

[77] Z. Melek and J. Keyser. Driving object deformations from internal physical processes. In *Proceedings ACM Symposium on Solid and Physical Modeling*, pages 51–59, 2007.

[78] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159, 2003.

[79] F. Neyret. Advected textures. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 147–153, July 2003.

[80] D. Nguyen, R. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire. In *Proceedings of ACM SIGGRAPH '02*, pages 721–728,

2002.

[81] P. Ning and J. Bloomenthal. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications*, pages 33–41, 1993.

[82] NVIDIA. Animated layered fire. http://developer.nvidia.com/, 2007.

[83] J. F. O'Brien, A. W. Bargteil, and J. K. Hodgins. Graphical modeling and animation of ductile fracture. In *Proceedings of ACM SIGGRAPH '02*, pages 291–294, 2002.

[84] K. Onoue and T. Nishita. Virtual sandbox. In *Proceedings of Pacific Graphics '03*, pages 252–259, 2003.

[85] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer Verlag, New York, 2002.

[86] S. Osher and J. A. Sethian. Front propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, (79):12–49, 1988.

[87] S. R. Parry. *Free-Form Deformations in a Constructive Solid Geometry Modeling System*. PhD thesis, Brigham Young University, 1986.

[88] M. Pauly, D. K. Pai, and L. J. Guibas. Quasi–rigid objects in contact. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 109–119, 2004.

[89] K. Perlin and F. Neyret. Flow noise. In *SIGGRAPH Technical Sketches and Applications*, page 187, Aug 2001.

[90] K. H. Perlin. An image synthesizer. In *Proceedings of ACM SIGGRAPH '85*, volume 19, pages 287–296, 1985.

[91] K. H. Perlin and E. M. Hoffert. Hypertexture. In *Proceedings of ACM SIGGRAPH '89*, volume 23, pages 253–262, 1989.

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

[93] F. Pighin, J. M. Cohen, and M. Shah. Modeling and editing flows using advected radial basis functions. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 223–232, 2004.

[94] J. Popović. *Interactive Design of Rigid Body Simulations for Computer Animation*. PhD thesis, Carnegie Mellon University, 2001.

[95] J. Popović, S. M. Seitz, M. Erdmann, Z. Popović, and A. Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of ACM SIGGRAPH '00*, pages 209–218, 2000.

[96] S. Premoze, T. Tasdizen, J. Bigler, A. Lefohn, and R. Whitaker. Particle based simulation of fluids. In *Proceedings of Eurographics*, volume 22, pages 401–410, 2003.

[97] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, 1992.

[98] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In *Proceedings of*

ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 193–202, 2004.

[99] N. Rasmussen, D. Q. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. *ACM Transactions on Graphics*, 22(3):703–707, 2003.

[100] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108, 1983.

[101] J. Schpok, W. Dwyer, and D. S. Ebert. Modeling and animating gases with simulation features. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 97–105, 2005.

[102] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. In *Proceedings of ACM SIGGRAPH '86*, pages 151–159, 1986.

[103] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics*, 24(3):910–914, 2005.

[104] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, New York, 1999.

[105] M. Shah, J. M. Cohen, S. Patel, P. Lee, and F. Pighin. Extended galilean invariance for adaptive fluid simulation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 213–221, 2004.

[106] L. Shi and Y. Yu. Controllable smoke animation with guiding objects. *ACM Transactions on Graphics*, 24(1):140–164, 2005.

[107] L. Shi and Y. Yu. Taming liquids for rapidly changing targets. In *Proceedings of 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 229–236, 2005.

[108] J. Stam. Stable fluids. In *Proceedings of ACM SIGGRAPH '99*, pages 121–128, 1999.

[109] J. Stam. Interacting with smoke and fire in real time. *Communications of the ACM*, 43(7):77–83, 2000.

[110] J. Stam and E. Fiume. Turbulent wind fields for gaseous phenomena. In *Proceedings of ACM SIGGRAPH '93*, pages 369–376, 1993.

[111] J. Stam and E. Fiume. Depicting fire and other gaseous phenomena using diffusion process. In *Proceedings of ACM SIGGRAPH '95*, pages 129–136, 1995.

[112] J. Steinhoff and D. Underhill. Modification of the euler equations for vorticity confinement: Application to the computation of interacting vortex rings. *Physics of Fluids*, 6(8):2738–2744, 1994.

[113] J. Strain. Semi-lagrangian methods for level set equations. *Journal of Computational Physics*, 151(2):498–533, 1999.

[114] J. Strain. A fast modular semi-lagrangian method for moving interfaces. *Journal of Computational Physics*, 161(2):512–536, 2000.

[115] R. W. Sumner, J. F. O'Brien, and J. K. Hodgins. Animating sand, mud, and snow. In *Proceedings of Graphics Interface '98*, pages 125–132, 1998.

[116] P. N. Swarztrauber and R. A. Sweet. Efficient fortran subprograms for the solution of separable elliptic partial differential equations. *ACM Transactions on Mathematical Software*, 5:352–364, 1979.

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

[118] A. Treuille, A. Lewis, and Z. Popovic. Model reduction for real-time fluids. *ACM Transactions on Graphics*, 25(3):826–834, 2006.

[119] A. Treuille, A. McNamara, Z. Popović, and J. Stam. Keyframe control of smoke simulation. *ACM Transactions on Graphics*, 22(3):716–723, 2003.

[120] Y. Uralsky. Volumetric fire. http://www.cgshaders.org/, 2002.

[121] W. von Funck, H. Theisel, and H.-P. Seidel. Vector field based shape deformations. *ACM Transactions on Graphics*, 25(3):1118–1125, 2006.

[122] S. W. Wang and A. E. Kaufman. Volume sculpting. In *Proceedings of 1995 Symposium on Interactive 3D Graphics*, pages 151–156, 1995.

[123] J. Warren. Barycentric coordinates for convex polytopes. *Advances in Computational Mathematics*, (6):97–108, 1996.

[124] J. Warren, S. Schaefer, A. Hirani, and M. Desbrun. Barycentric coordinates for convex sets. Technical report, Rice University, 2004.

[125] X. Wei, W. Li, K. Mueller, and A. Kaufman. Simulating fire with texture splats. In *IEEE Visualization '02*, pages 227–235, 2002.

[126] G. D. Yngve, J. F. O'Brien, and J. K. Hodgins. Animating explosions. In *Proceedings of ACM SIGGRAPH '00*, pages 29–36, 2000.

[127] S. Yoshida and T. Nishita. Modelling of smoke flow taking obstacles into account. In *Proceedings of Pacific Conference '00*, pages 135–144, 2000.

[128] C. Yuksel, D. H. House, and J. Keyser. Wave particles. *ACM Transactions on Graphics*, 26(3):99, 2007.

[129] Y. Zhao, X. Wei, Z. Fan, A. Kaufman, and H. Qin. Voxels on fire. *IEEE Visualization*, 14:271–278, 2003.

[130] Y. Zhu and R. Bridson. Animating sand as a fluid. In *Proceedings of ACM SIGGRAPH '05*, pages 965–972, 2005.

APPENDIX A

A BRIEF INTRODUCTION TO STABLE FLUIDS

Compressible flow equations introduce a very strict time step restriction associated with acoustic waves. To avoid this strict restriction, incompressible flow equations are preferred whenever possible, and they are very popular in CG. When the speed of the flow is below the speed of sound, the compression effects are negligible [28]. In compact form, the incompressible Navier-Stokes equations governing fluid flow are as follows:

$$\nabla \cdot \vec{u} = 0 \tag{A.1}$$

$$\frac{\partial \vec{u}}{\partial t} = \vec{f} - (\vec{u} \cdot \nabla)\vec{u} - \frac{1}{\rho}\nabla p + \upsilon \nabla^2 \vec{u} \tag{A.2}$$

where $\vec{u}$ defines the motion of the fluid, and Eq. A.2 describes the fluid flow. Eq. A.1 is the incompressibility condition, hence a divergence free flow.

Looking at Eq. A.2 more closely, $\vec{f}$ is the external force applied to the system, which includes buoyancy and gravity forces. Since we use incompressible flow equations, we need buoyancy as an external force term to describe rising hot air.

The next term is the nonlinear part of the Navier-Stokes equation - it is the advection of the velocity field.

The third term accounts for the pressure induced motion, where p defines the pressure, the force exerted by the fluid on unit area. ρ is the density of the fluid. For simplification purposes we can assume uniform density fluid with density equal to one.

The last term is motion coming from shearing. υ defines kinematic viscosity of the fluid defining how much the fluid resists deforming. This term is important for viscous liquids, but for practical purposes we can drop this term for gases. The loss of energy coming from the implicit solution is more dominant than viscosity effects in a gas. Navier-Stokes equations without the viscosity term are called Euler equations, and they describe inviscid fluid flow.

After the simplifications described above, we have uniform density inviscid flow defined as follows:

$$\frac{\partial \vec{u}}{\partial t} = \vec{f} - (\vec{u} \cdot \nabla)\vec{u} - \nabla p \tag{A.3}$$

The "Stable Fluids"[108] solution involves three main steps. We will briefly introduce them here, but we encourage the reader to refer to the brilliant seminal paper [108].

- The first step advects the velocity field itself to the next time step using a semi-Lagrangian advection scheme. A particle at each cell center is traced back in time over the time step and the new velocity for the cell is the interpolation of the velocities that the particle had one time step ago. Instead of a first order integration scheme used in the original paper [108], second (such as the midpoint method) or third (such as monotonic cubic interpolation [28]) order integration schemes are preferred to minimize the smoothing effects of the interpolation.

- The second step determines the force term. The buoyancy force is based on the temperature and the gravity force is based on the amount of fuel and exhaust gas densities in each grid cell.

- The last step combines Eq. A.1 and A.2 and makes the velocity field divergence

free and incompressible using a projection method [19]. The flow field found in previous steps is made incompressible by subtracting the gradient of the pressure.

$$\nabla^2 p = \frac{1}{\Delta t} \nabla \cdot u \tag{A.4}$$

The pressure is found by solving the Poisson equation (Eq. A.4) with the Neumann boundary condition. The Poisson equation becomes a sparse linear system, making the implementation straightforward, using multigrid methods such as the conjugate gradient [45] [116] [97].

A non-reactive substance like smoke is advected at the same time with the fluid:

$$\frac{\partial a}{\partial t} = -(u \cdot \nabla)a - \alpha_a a + S_a \tag{A.5}$$

where α_a is a dissipation rate and S_a is a source term for gas a. This gas could be combustible fuel gas a_f or smoke a_s. Since Eqs. A.3 and A.6 have identical structure, the same solver can be reused. Heat could be also advected as a density using the same framework. Note our use of the inviscid flow equations leaving the viscosity term out. If diffusion is to be included in to the model, the equation becomes

$$\frac{\partial a}{\partial t} = -(u \cdot \nabla)a - \alpha_a a + S_a + +\upsilon \nabla^2 \vec{a} \tag{A.6}$$

where υ is the viscosity. The solution for the diffusion term has a similar structure to Eq. A.4, and the same solver can be used efficiently.

A couple extensions to the "Stable Fluids" model have been proposed after its introduction.

The first one we are going to look at is vorticity confinement [28]. Navier-Stokes equations are "correct" in the limit case, where the grid cell size Δx goes to zero. In practice, we will never work close to the limit, and in CG we usually work with very large grid sizes. This inevitably adds rounding errors in our simulation. The "Stable Fluids" model uses implicit solvers, which add additional smoothing into our solution, which comes out as loss of energy. This is noticeable as small scale vorticities vanish too quickly. Vorticity confinement adds back the "lost" energy as a pedal force keeping the flow "alive".

The vorticity confinement force is defined as:

$$\vec{\omega} = \nabla \times \vec{u} \tag{A.7}$$

$$\vec{f_w} = \epsilon_w \Delta x (\vec{N} \times \vec{\omega}) \tag{A.8}$$

The vorticity confinement force, $\vec{f_w}$, uses local vorticity, $\vec{\omega}$, scaled by the grid size. This scaling will make sure that as we go to the limit case, the artificial vorticity confinement force will drop from the equation.

Yngve et al. [126] used compressible Navier-Stokes equations to model explosions, but this model uses explicit integration and requires very small timesteps for stability. Instead, incompressible fluid equations are used to model compressible behavior whenever possible. The second extension to the "Stable Fluids" model we are going to look at defines a workaround to model explosions using incompressible equations. Feldman et al. [29] modifies the equations as follows:

$$\nabla \cdot \vec{u} = \phi \tag{A.9}$$

$$\nabla^2 p = \frac{1}{\Delta t}(\nabla \cdot u - \phi) \tag{A.10}$$

Divergence, ϕ, is set greater than zero where there is additional fluid introduced into the system, or some process causes the existing fluid to expand by heating it. This model is used by Ihm et al. [50] to model various chemical reactions.

Other extensions addresses the space requirements. Shah et al. [105] take advantage of the Galilean Invariance to let the grid follow the important part of the fluid, keeping the simulation grid size small. Similarly, Rasmussen et al. [98] use whole grid increments to follow the fluid. Recently, Harris et al. [46] implemented the solver on the graphics hardware.

APPENDIX B

A BRIEF INTRODUCTION TO THE LEVEL SET METHOD

Tracking dynamic surfaces presents a series of problems. Pinching and merging changes connectivity of the surface, and methods working with explicit representations of the surface require a lot of special case handling to get the new surface right.

A level set is a powerful method to track moving interfaces. The surface is defined implicitly as the zero isosurface of the distance field, where ϕ denotes the signed distance to the surface.

$$\phi(\vec{x}) = 0 \tag{B.1}$$

Implicit definitions of surfaces offer a set of advantages. The surface normal is defined as follows :

$$\vec{n} = \frac{\vec{\nabla}\phi}{|\vec{\nabla}\phi|} \tag{B.2}$$

Note that the normal is defined not only on the surface, but everywhere. Not only the first derivative, but the second order structures such as principal curvatures k_1, k_2, and thus mean and gaussian curvatures, are also well defined, everywhere. Topology is implicit, its handling is trivial, and there is no such consideration as non-manifold. If required, one can convert to a boundary representation quickly, by using marching cubes [64].

There are also some drawbacks. An implicit distance field as a regular grid representation requires a large amount of space, and the amount of fine scale detail

one can resolve is inversely proportional to the size of the grid cells. Since there is no direct boundary representation available, one needs to be created for interactive visualization.

When discretized on a regular grid, one possible definition of ∇ becomes

$$\vec{\nabla}\phi = \begin{pmatrix} \frac{\partial\phi}{\partial x} \\ \frac{\partial\phi}{\partial y} \\ \frac{\partial\phi}{\partial z} \end{pmatrix} \tag{B.3}$$

$$\frac{\partial}{\partial x}\phi_{i,j,k} = \frac{1}{2\Delta x}(\phi_{i+1,j,k} - \phi_{i-1,j,k}) \tag{B.4}$$

$$\frac{\partial}{\partial y}\phi_{i,j,k} = \frac{1}{2\Delta y}(\phi_{i,j+1,k} - \phi_{i,j-1,k}) \tag{B.5}$$

$$\frac{\partial}{\partial z}\phi_{i,j,k} = \frac{1}{2\Delta z}(\phi_{i,j,k+1} - \phi_{i,j,k-1}) \tag{B.6}$$

Moving interfaces, where the surface changes in time, are defined as a level set

$$\phi(\vec{x}, t) = 0 \tag{B.7}$$

And the evolution of the interface is defined as

$$\frac{\partial}{\partial t}\phi + \vec{\nu} \cdot \nabla\phi = 0 \tag{B.8}$$

where $\vec{\nu}$ is equal to desired velocity at the interface. Actually we only need the normal component of the velocity $\vec{\nu}_N = \vec{\nu} \cdot \vec{n}$. If we plug in equation B.2

$$\frac{\partial}{\partial t}\phi + \vec{\nu}_N \cdot |\nabla\phi| = 0 \tag{B.9}$$

The popularity of the level set methods come from the fact that the velocity field can come from a number of sources, external or internal. One can use the curvature to define the velocity field, and use the level set method for surface smoothing. One can use the fluid flow to define the velocity field, and use the level set method to track the free surface of the flowing liquid. Level set methods are used from image processing to surface reconstruction and physical simulations.

To track the evolution of the level set isosurface, the distance field is advected every time step. Here, semi-Lagrangian methods are preferred. One thing to note here, is that the level set method is accurate only around the interface and will have the correct distance to the isosurface only around the interface. One has to reinitialize to get a correct distance field representation everywhere.

A standard reinitialization algorithm might be slow, especially if it needs to be executed for every timestep, and several solutions have been studied. If the field is monotonically increasing or decreasing, one can use a fast marching method to reinitialize the distance field. A fast marching method consists of sweeps in pre-ordained directions and converges rapidly. Another method for dealing with reinitialization is to define the distance field only locally around the interface. These narrow band methods expand and shrink the band around the interface as the level set propagates.

To overcome the space requirements mentioned earlier, many extensions are proposed. Octrees [65] and run length encoding (RLE) [49] methods are used to decrease the space requirements. To handle the mass (or volume) loss due to the grid resolution and interpolation done at the advection step, Lagrangian extensions such as particle level sets are proposed [25]. Particle level sets introduce massless particles placed inside and outside of the interface, which are advected together with the interface. If inside particles end up outside of the interface or outside particles on the inside, a correction is done on the level set by averaging particles around the problem area.

A recent extension adds the capability to track multiple interfaces using the particle level set method [67].

VITA

Zeki Melek spent almost all of his life as a graduate student, at least he feels like it. Born in 1973, he received his B.S. in Computer Engineering from Bogazici University in 1996. Working part time as a computer animator, he finished his Computer Science M.S. thesis in 1999 at the same university. The thesis was titled "Automated Lip-Synchronized Facial Animation", and was under the guidance of Lale Akarun.

In 2001 he started his Ph.D. study with John Keyser at Texas A&M University. His research interests include physically based modeling, scientific visualization, and computer animation.

Zeki Melek may be reached at Department of Computer Science, Texas A&M University, College Station, TX, USA. His email address is melekzek@tamu.edu