Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2011)
A. Bargteil and M. van de Panne (Editors)

Graph-based Fire Synthesis

Yubo Zhang[1], Carlos D. Correa[2], and Kwan-Liu Ma[1]

[1]University of California Davis
[2]Lawrence Livermore National Laboratory

Figure 1: *Our synthetic fire results. These simulations are synthesized from exemplar simulations at a low computational cost using our novel* flow-graph *. The accompanying video shows that fire animations using our approach appear smooth and with visually plausible transitions.*

Abstract

We present a novel graph-based data-driven technique for cost-effective fire modeling. This technique allows composing long animation sequences using a small number of short simulations. While traditional techniques such as motion graphs and motion blending work well for character motion synthesis, they cannot be trivially applied to fluids to produce results with physically consistent properties which are crucial to the visual appearance of fluids. Motivated by the motion graph technique used in character animations, we introduce a new type of graph which can be applied to create various fire phenomena. Each graph node consists of a group of compact spatial-temporal flow pathlines instead of a set of volumetric state fields. Consequently, achieving smooth transitions between discontinuous graph nodes for modeling turbulent fires becomes feasible and computationally efficient. The synthesized particle flow results allow direct particle controls which is much more flexible than a full volumetric representation of the simulation output. The accompanying video shows the versatility and potential power of this new technique for synthesizing realtime complex fire at the quality comparable to production animations.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Simulating a realistic fluid such as fire remains a challenge mainly due to its computational cost. Lower cost methods such as sprite-based compositing are available and have been used in games and other interactive applications. In production-quality animation, however, realistic fluids are typically obtained by solving the Navier-Stokes equations at a high computational cost. Because of the computational cost of physically-based simulations, animators often need to wait a long time to see the simulation results. The situation becomes even worse when multiple fire instances are present in the scene.

© The Eurographics Association 2011.

In order to alleviate the costs associated with solving these equations, different acceleration techniques were applied such as using adaptive grids and parallelization. These techniques are effective but also bring complicated technical issues in implementation. In this paper, we present a data-driven method that is relatively easy to implement for synthesizing fire animations at a much lower computational cost. Inspired by data-driven animation approaches such as the *motion graph*, we pre-compute a motion data structure from a few short simulations, called the *flow graph*. Although motion graph can be applied to volumetric flow data, it cannot create smooth transitions without introducing a large amount of samples with complex and costly constraints. Unlike the motion graph, every node of the flow graph comprises flow particle pathlines instead of individual motion states, which makes it both spatial-temporal compact and computationally efficient. The compactness is optimal because pathlines automatically track the desired physical properties. It is much more efficient than naively applying a time-varying hierarchical tree structure on volumes. In addition, much less samples are needed since it is easier to find plausible transitions between spatio-temporal pathlines than entire volume states. We use this pathline-based structure to synthesize fire animations through a stochastic walk on the flow graph where constraints can be applied to limit the traversal on subgraphs. Our method can generate long and smooth animation sequences in realtime only using a few short simulations with different parameter configurations. The synthesis time is independent of the simulation complexity (i.e. the complexity of model equations and numerical schemes). Such a particle-based flow design also gives animators extended controls. While data-driven fluid animation received far less attention, our work is the first attempt to develop a practical data-driven framework that can synthesize high-quality 3D fluid flows at interactive rates.

2. Related Work

Solving the Navier-Stokes equations is one of the main difficulties in physically-based fluid animation, and many classical numerical methods from the CFD community have been applied and popularized in the fluid animation field. For example, Stam [Sta99] presents a framework to achieve stable numerical simulation based on Eulerian grids and the smoothed particle hydrodynamics (SPH) method is adopted in [MCG03, SP09]. To simulate specific visual effects, different simulation models as well as rendering techniques are proposed, such as fire [NFJ02, HG09], explosions [YOH00, FOA03], smoke [FSJ01, WP10], liquid [FF01], viscoelastic materials [GBO04, CBL*09], controls [FL04, MTPS04, PCS04, ANSN06] and general natural phenomena [ND01]. In recent years, incorporating small details into fluid flows has received much attention [HSF07, KTJG08, NSCL08, SB08, MWGZ09, PTSG09] in order to produce high quality animations with special effects. Most of these techniques introduce artificially synthesized details due to the complexity and high computational cost of physically accurate high-resolution models.

Straightforward ways to reduce the simulation time include adaptive mesh grids [LGF04] and GPU acceleration [Har03]. Treuille et al. [TLP06] propose a dimension reduction technique which projects the solution to a lower dimensional space. Wicke et al. [WST09] present a new approach to balancing the speed of model reduction with the flexibility of grid-based methods. Fourier-based methods [Sta01, LR09] can also speed up the computation but are limited to regular domains. Lentine et al. [LZF10] present a method that solves the pressure-Poisson equation on coarse grids. However, the computational cost still increases with the grid resolution and animators need to run simulations with various physical parameters to get the desired results.

Alternatives to direct simulation are data-driven approaches. Kwatra et al. [KSE*03] present an image and video synthesis method using graph-cuts. Bhat et al. [BSHK04] synthesize flow phenomena from video. These approaches produce visually plausible results but the view is fixed and the resulting flow cannot be further manipulated. Cha et al. [CLC*09] propose a data-driven approach to re-create fire effects from simulation data. Their focus, however, is in data processing and sampling. Octrees and multi-resolution techniques are used for sampling the original volume data. Their storage cost is high as needed for high resolution time-varying volume data. In this paper, we solve this problem by leveraging fluid simulation data to synthesize desired fluid motion, with low computational and storage cost. Synthesizing similar data from sample data is a general idea (e.g. video textures [SSSE00], texture synthesis [LLX*01, MWGZ09], etc.). Motion synthesis has already been studied for human motions [KGP02, AF02, LCF05] and relatively simple object movements [JTCW07, ZZJ*07]. The key ingredient used in these techniques is the motion graph [KGP02], a directed graph that encapsulates different stages of a motion and the possible transitions between them. But directly applying this technique to fluid flow only gives limited benefits with high cost (i.e. huge amount of graph nodes and edges), partly because fluid flow is turbulent and the degrees of freedom for fluid motion are generally higher than for character motion. To alleviate this problem, we introduce a novel graph-based fire synthesis approach to create animations of arbitrary length from existing simulation data at low storage and computational cost.

3. Overview

Our fire synthesis method consists of two main stages: flow data construction and interactive flow synthesis. During the first stage, we take a small number of fire simulations with desired physical parameters, such as turbulence magnitude and external forces, and generate a directed graph based on pathlines. These pathlines are created by tracking particles as they flow over time and are grouped in nodes according

© The Eurographics Association 2011.

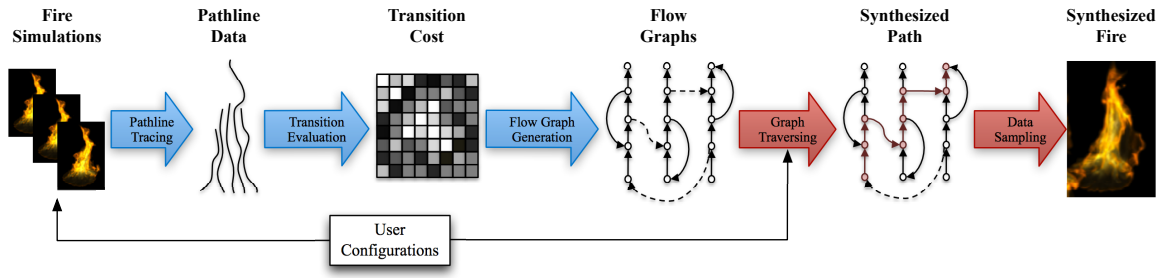

Figure 2: *The workflow of our synthesis framework. In a preprocessing stage (blue arrows), we trace pathlines from each simulation and build a transition cost that measures the similarity between pairs of pathline groups. The flow graph is a subset of this transition matrix. In an interactive stage (red arrows), we synthesize new animations as random walks on the flow graph.*

to their starting time. The graph encodes groups of pathlines that are temporally coherent or exhibit some flow similarity, encoded in a cost matrix. During the second stage, we synthesize new fire animations through a random walk on the flow graph. Each random walk defines a new group of spatially and temporally coherent pathlines. Since each pathline node contains physical quantities from the simulation, we can generate a new animation using integration (see Fig. 2).

4. Flow Graph Construction

In the first stage of our technique, we construct the flow graph, which stores the space of possible particle tracks from a set of input simulations in a compact manner. As an input, we assume that a set of simulations are given, and that we have a mechanism to retrieve particle data and sample physical quantities such as velocity and temperature. For the sake of completeness, we include the discussion on how we simulate fire data to provide input to the synthesis pipeline.

4.1. Fire Simulation

We have implemented a GPU-based 3D fluid simulator using NVIDIA CUDA [NVI07]. The solver is based on [Sta99], which solves the Navier-Stokes equations

$$\nabla \cdot \mathbf{u} = \phi \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \mathbf{f} \tag{2}$$

where $\mathbf{u}$ is the velocity, p is the pressure, $\mathbf{f}$ is external force including buoyancy and ϕ is zero except the reaction area. We designed a simple combustion model that converts fuel into heat and soot

$$\frac{\partial F}{\partial t} + (\mathbf{u} \cdot \nabla)F = -rF \tag{3}$$

$$\frac{\partial S}{\partial t} + (\mathbf{u} \cdot \nabla)S = k_S rF \tag{4}$$

$$\frac{\partial T}{\partial t} + (\mathbf{u} \cdot \nabla)T = \kappa \Delta T - c\left(\frac{T - T_0}{T_{\max} - T_0}\right)^4 + k_T rF \tag{5}$$

where F is the fuel, S is the soot, T is the temperature, T_0 is the ambient temperature, r is the reaction rate, c is the cooling rate and κ is the thermal conductivity. Here the temperature cooling model is the same as [FOA03]. Vorticity confinement [FSJ01] is applied to compensate the numerical dissipation. The simulation must include an emitting region for fuel injection. Parameters such as reaction rate, cooling rate, buoyancy, turbulence magnitude as well as external force fields are configurable. These parameters are are used to simulate the desired fire states required to synthesize similar effects. For each selected configuration, we simulate minimum time length just enough to capture the desired flow features. We use fixed time step $\Delta t = 0.5\frac{\Delta x}{U}$ where Δx is the uniform grid spacing and U is the characteristic velocity. The simulation usually takes an hour on a 256^3 grid.

4.2. Pathline Data Sampling

The next step is the extraction of flow information that can be reused for subsequent synthesis. A straightforward way to reuse simulation data is to produce a series of volumetric fields and blend them selectively to create new volumetric fields. Instead, we sample and store the physical data along pathlines of fluid particles. This approach is more effective than reusing volumetric fields for the following reasons:

- Since flames are usually turbulent, finding smooth transitions between two non-consecutive time steps in volumetric fields is hard and computationally expensive. However, due to the flow coherence of particles, pathlines often exhibit similarity with other pathlines in different time steps.
- The storage of pathline data is more compact because pathlines track only the visible flow features automatically due to their Lagrangian nature. It produces the optimal result with less cost than time-varying tree-based structures.
- Using particles generated from pathlines provides better physically-meaningful manipulation than using volumes directly, where the structure of flow is no longer explicit.

The pathline data store velocity, soot density and temperature at equally distributed sample points along the lines.

© The Eurographics Association 2011.

During simulation, we seed a group of particles at the flame source at each time step for pathline tracing. The particles are uniformly distributed and the interval is half the grid resolution Δx. Given any start point $\mathbf{x}_0$ and start time t_0, its corresponding pathline is defined as

$$p(\mathbf{x}_0, t_0, t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{u}(p(\tau), \tau) d\tau \tag{6}$$

where t is the time, and $\mathbf{u}(\mathbf{x}, t)$ is the velocity at a given location and time. The pathlines are integrated through the fourth-order Runge-Kutta (RK4) method [But87].

4.3. Flow Graph Computation

Motivated by [SSSE00, KGP02, JTCW07, ZZJ*07], we have designed a new type of graph for synthesizing combustion phenomena in the time domain, the *flow graph*. A flow graph is a directed graph $G = (V, E)$ which has $N = |V|$ nodes. Unlike the motion graph, whose nodes define single motion states, each node in a flow graph represents the entire flow pathlines which started at the same time step. Let $V = \{\{P_k(t), U_k(t), R_k(t), T_k(t)\}, 0 < k \leq N\}$, where $P_k(t)$ represents the pathline group started at the kth time step, and $U_k(t)$, $R_k(t)$ and $T_k(t)$ are the corresponding velocity, density and temperature, respectively. Therefore, we can define P_k as the set of pathlines with starting time at the k-th time step, i.e. $t_k = k\Delta t$,

$$P_k(t) = \{p(\mathbf{x}, t_k, t), \forall \mathbf{x} \in \Omega\} \tag{7}$$

where $\Omega = \{\mathbf{x}_i\}_{i=1}^{N_s}$ is the set of uniformly seeded source particles inside the fuel injection region of the simulation. The velocity, density and temperature groups are defined analogously. Here t is defined in the local time space $[0, t_{\max}]$ which corresponds to $[t_k, t_k + t_{\max}]$ in the global time space. $t_{\max}$ is chosen such that all the fire particles dissipate before $t_{\max}$. Figure 3 illustrates the idea of how the pathlines are organized on the graph. We can see that each node defines a collection of flow lines that are spatially coherent, since they comprise all the point sources in the domain, and temporally coherent, since they encode the entire lifetime of a particle.

To enable synthesis, we must first connect the nodes in the graph with edges. Ideally, edges should connect pathlines that result in a plausible transition, so that a random walk on the graph results in a plausible animation. Note that we only allow transitions to happen at the beginning of a pathline set (i.e. particle emitting time) otherwise the physically-correct flow structure can be simply destroyed due to the Lagrangian nature of flow particles. To do that, we define two types of edges:

Temporal edges connect two consecutive nodes in time and are useful to reconstruct parts of the original fire animations,

$$(i, j) \in E \iff i = k \text{ and } j = k + 1, 0 < k < N \tag{8}$$

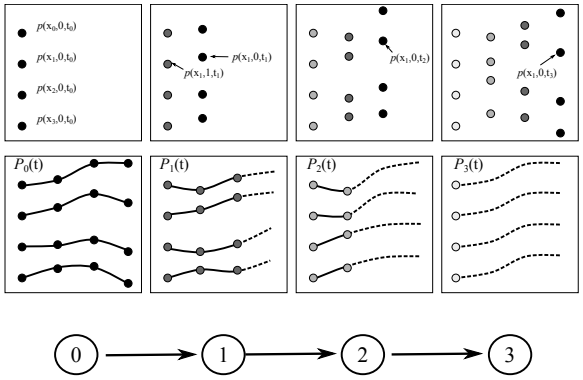

Figure 3: *How nodes are extracted. Top row: Four time steps and the corresponding advected particles from source particles (left column of nodes). Middle row: We group the pathlines $P_k(t)$ that start at the same time and encode them in individual nodes (bottom row). This ensures that the transitions in the flow graph are coherent.*

Transition edges connect non-consecutive nodes that exhibit some degree of similarity and allow us to synthesize new path flows from several input animations. To measure the similarity between two nodes i and j, we define a cost function c_{ij}, as:

$$c_{ij} = \sqrt{\int_{t_0}^{t_{\max}} \left\|U_i - U_j\right\|_2^2 + \alpha \left\|R_i - R_j\right\|_2^2 + \beta \left\|T_i - T_j\right\|_2^2 dt} \tag{9}$$

where

$$\left\|U_i - U_j\right\|_2 = \sqrt{\frac{1}{|\Omega|} \sum_{\mathbf{x} \in \Omega} \left|\mathbf{u}(\mathbf{x}, t_i, t) - \mathbf{u}(\mathbf{x}, t_j, t)\right|^2} \tag{10}$$

is the sum of differences between the two path velocities, and is similarly defined for $\left\|R_i - R_j\right\|_2$ and $\left\|T_i - T_j\right\|_2$, $|\Omega|$ is the number of particles seeded in Ω, α and β are weighting parameters controlled by the user to give more importance to the density and temperature differences, respectively. In general, α and β are set such that the differences of velocity, density and temperature are in the same order of magnitude. We compute the similarity cost from Equation 9 using the whole lifetime of particles because the visible regions of fire rely on a user-defined color map which are unknown at this stage.

We add a transition edge when the cost function is smaller than a given threshold ε, i.e. when two nodes are similar enough so that a transition between the two is plausible, i.e.,

$$(i, j) \in E \iff c_{ij} < \varepsilon \tag{11}$$

One of the issues with this method is that it may lead to a large number of cluttered transitions, which results in a dense flow graph. Moreover, these transitions may not be distributed evenly throughout the simulation time, which is

© The Eurographics Association 2011.

undesirable for creating plausible fluid animations. To obtain an even distribution of transitions, we order the nodes chronologically and subdivide the cost matrix $C = (c_{ij})$ into blocks of size $B \times B$. Those node pairs corresponding to the local minima of the cost function within each block are then selected as candidate transition edges. Then we filter these edges by an appropriate tolerance value $a < \varepsilon < b$ through (11), where a and b are the minimum and maximum costs of these edges. In this case, the number of transitions is limited by M^2 where $M = N/B$ and the length of dead ends are usually less than cB where c is a small integer if ε is not close to a. We also filter the edges whose nodes are close because it may generate undesired results. The purpose of the filtering is to reduce the complexity of a graph and guide the tolerance selection. The average transition cost of resulting edges depends on the input data which can be carefully designed and simulated in order to provide enough transitions. Figure 4 shows some simple graphs generated with different block sizes. Based on the time scale, we use $B = 32$ and $\varepsilon = (2a+b)/3$ in our implementation.

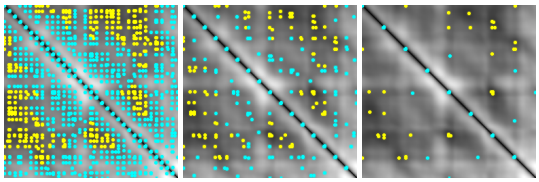

Figure 4: *Example flow graphs generated with different block sizes $B = 8, 16, 32$. Yellow marks are the final transition edges and blue marks are the filtered edges. The gray scale background shows the transition cost where darker area indicates lower costs.*

The graph is computed in parallel during simulation with multiple CPU threads because the CPU is relatively free during GPU-based simulations. Flow graphs are generated for each simulation with different configurations, each of which contain variations on the simulation parameters that result in a richer input set. In a latter state, we use this flow graph to synthesize new animations using random walks.

5. Interactive Fire Synthesis

The synthesis process is as follows: First, we generate novel pathlines using random walks on the flow graph. Then, we integrate these pathlines over time to produce a new fire animation, which is finally rendered with production-quality rendering or interactive rendering using the GPU.

5.1. Traversing Flow Graphs under Constraints

During this stage, we produce a random flow, which is a path of the flow graph starting at node $k = 0$ of a prescribed length L. Let us define a partial path $\{V_1, V_2, \ldots, V_i\}$. Assume $d(V_i)$

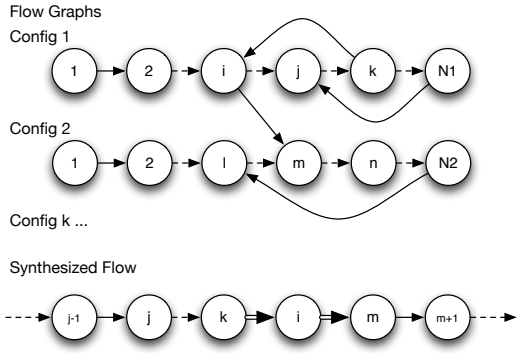

Figure 5: *An example of graph-based flow synthesis. Flow graphs are generated for each simulation with different configurations. Each node represents a group of pathlines started at the same time. Edges that connect discontinuous nodes represent feasible transitions. In the synthesized flow, thick arrows represent transitions.*

is the out degree of vertex i. If $d(V_i) = 1$, then we follow the only edge $(i, j) \in E$ out of V_i, which means the edge is either a temporal edge or a transition edge close to a dead end. If $d(V_i) > 1$, the next vertex in the path V_j is chosen randomly. In general, one can define purely random traversals by choosing edges with a uniform probability $p(i, j) = \frac{1}{d(V_i)}$, where $(i, j) \in E$. Even though random walks result in plausible transitions, we propose two methods to constrain this traversal and offer a better control to the animator.

Traversal with temporal bias. In this traversal, the user controls the likelihood of following a temporal edge instead of being purely random. This traversal provides more control when the animator desires a more structured fire. We introduce a free parameter $0 < \gamma < 1$, which represents the probability of following a temporal edge. The probability of following an edge $(i, j) \in E$ is now:

$$p(i,j) = \begin{cases} \frac{1-\gamma}{d(V_i)-1} & (i,j) \in E, i+1 \neq j \\ \gamma & (i,j) \in E, i+1 = j \\ 0 & \text{otherwise} \end{cases} \qquad (12)$$

A lower value of γ leads to frequent transitions while a higher value of γ produces fires with more stable states. Note that $\gamma = \frac{1}{d(V_i)}$ results in the purely random case. Fig. 6 shows a few animations for three settings. On top, $\gamma = \frac{1}{d(V_i)}$. In the middle, $\gamma = 0.875$ and $B = 1$. At the bottom, $\gamma = 0.875$ and $B = 32$. Note that the results with temporal bias and a larger block size B tend to produce smoother transitions.

Traversal with target states. In this traversal, we allow the user to steer the synthesis towards a target physical state, defined as a subset $W \subset V$. The edge probabilities are computed depending on how far the current visited node is from the target state. Let us assume that the current visited node is

© The Eurographics Association 2011.

Figure 6: *Frames from a synthesized animation using different traversal strategies. Top: Random transitions and $B = 32$, Middle: Biased transitions, $\gamma = 0.875, B = 1$. Bottom: Biased transitions with larger block size, $\gamma = 0.875, B = 32$. Even though transitions are visually plausible, temporal bias and a larger block size B produce smoother transitions.*

Case	States	Simulation Grid	Output Steps	Unsaved Vol. Data	Pathlines/Node	Pathline Data	Avg. Proc. Time/Step	Syn. Time/Step
Campfire	3	$128 \times 64 \times 64$	868	27GB	3217	5GB	407ms	35ms
Firewall	2	$400 \times 200 \times 100$	356	114GB	39375	40GB	2470ms	144ms
Fire Blade	1	$200 \times 200 \times 150$	300	36GB	32360	13GB	1691ms	121ms
Fire Plume	3	$200 \times 400 \times 200$	300	288GB	70686	85GB	3516ms	271ms

Table 1: *Detailed statistical results of our experiments.*

V_i. When we are out of reach of the target state, i.e., $V_i \notin W$ and $V_j \notin W, \forall (i,j) \in E$, then we perform random traversal as defined above. When we are at the boundary or the interior of the target state, i.e., there is at least one edge (i,j) such that $V_j \in W$, then the probability of an edge is defined uniformly as $p(i,j) = \frac{1}{k}$, if $V_j \in W$, or 0, otherwise, where k is the number of neighbors of V_i in set W. Note that once a target state is reached, the traversal will remain there until a new target state is defined. It is easy to achieve transitions between different states where the average number of steps before transition depends on the block size of the cost matrix which is described in Section 4.3.

Algorithm 1 Graph-based Fire Synthesis

```
Load configuration
Load flow graphs
Allocate particle buffers
Start streaming pathline data
while synthesizing = true
    Read user state constraint of flow graph
    Let node be next random node under constraint
    Emit particles associated with node
    Update positions of existing particles
    Sample particle attributes from pathline data
    Remove particles which exceed lifetime
    Send current particles to the preview renderer
end
```

5.2. Flow Integration and Deformation

For each visited node in the synthesized flow, we generate a group of particles following the node's pathline, each with physical attributes such as velocity and temperature. The velocity information is used to integrate the displacement of the particles at each time step

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \int_0^{\Delta t} [\mathbf{u}(t) + \mathbf{u}_{ext}(t)] dt \qquad (13)$$

where $\mathbf{x}_k$ is the position of certain particle, $\mathbf{u}$ is the velocity sampled from the pathline data and $\mathbf{u}_{ext}$ is the velocity introduced by (optional) external forces, useful for fire deformation. Scalars such as density and temperature are used for rendering. In general, traversing the flow graph and integration happen in parallel, as summarized in Algorithm 1.

5.3. Rendering

Although the synthetic results can be sent to a high quality renderer with global illumination and multiple scattering ef-

© The Eurographics Association 2011.

fects, we adopt a simple slice-based renderer for interactive preview purpose using the G3D graphics engine [McG10]. The synthesized fluid particles and their attributes are first projected onto a set of slicing planes which are orthogonal to the camera direction. These slicing planes are then blended to the screen in back-to-front order. A 1D color texture is used as transfer function to convert flow attributes to colors and opacities. In our examples, we use 128 slicing planes of size 640×480 which can achieve interactive frame rates when the number of particles is in the scale of 10^5. Results show that the rendering quality is sufficient for previewing.

5.4. Results and Discussion

We performed a number of tests including a small campfire, a thin fire wall, an animated fire blade and a high resolution fire plume. The campfire consists of three configurations with different levels of wind strength. The fire wall and fire plume transit from a low turbulence state to a high turbulence state. The animated fire blade demonstrates the use of extended particle control where the fire source is attached on an dynamic mesh. All the tests are performed on a PC with an Intel Core2 3.0GHz CPU (4 cores) and an NVIDIA Quadro 6000 GPU. Table 1 summarizes these results.

We compared the cost of simulation vs. synthesis for different problem scales. Results show that the cost of simulation grows nonlinearly with the problem scale while the cost of synthesis grows linearly. Figure 7 is a comparison of the cost between simulation and synthesis. Our method reduces the length of time-consuming simulation towards the generation of visually similar results. Because of our low-cost synthesis method, performance increases greatly.

Rendering results are shown in Figure 1 and the accompanying video. The quality of the results depends on the spatial and temporal resolution of the simulation data, assuming that particles are sampled at the Nyquist rate. Note that the synthetic fires are not exactly the same as the original simulations due to the random traversals, but the transitions appear smooth and visually plausible. Smooth transitions can be achieved without special treatment through our traversal methods. When the synthesizer reaches a transition edge, it enters the transition state. The next particle group to be emitted will follow a different set of pathlines from the original simulation. All the existing particles still follow their paths and the synthesizer will not leave the transition state until these existing particles are dissipated. This space-time transition mechanism makes the flow smoother than changing the entire motion state at a single time step (i.e. using the motion graph), since it is relatively easier to find similar pathline groups instead of entire similar motion states in limited simulation steps. However, the quality of transitions still depend on the original data. In some cases where the input samples are sparse (i.e. with less similar/overlapping pathlines), although our method can find best transitions with higher costs, small flickering artifacts may be produced. It is possible to reduce such effects by applying smooth pathline blending techniques.

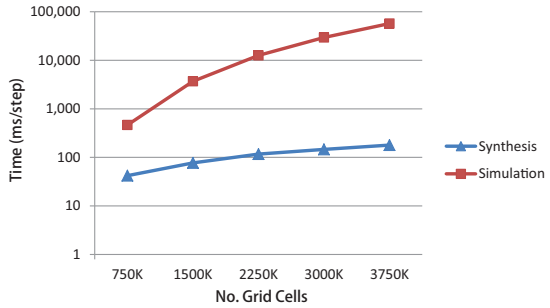

Figure 7: *Comparison of the time cost between simulation and synthesis on different problem scales. The data are measured from a campfire synthesis test with various resolutions.*

6. Conclusion and Future Work

Our graph-based synthesis technique provides a cost-effective way to generate fire animations of arbitrary length from examples, at a lower storage and computational cost. The synthesized particle results offer better interactivity than volumes for operations like external deformations. The performance gained by our method comes from the reduced simulation time which is used for generating similar flow behaviors. It is possible to apply the technique towards fire effects in animations and games. Due to the nature of data-driven methods, one of the limitations of our method is that we cannot do physically accurate fluid-fluid/fluid-solid interaction without additional simulations during the synthesis process. We do not allow motion blending in our system to prevent breaking important flow structures such as vortices, although it can be achieved by interpolating between different pathlines for laminar flows. Our method also requires that fire emanates from a source region and dissipates in a limited time. Our method can be extended to other dissipative gaseous effects such as smoke and dust. We will study hybrid methods that combine simulation and synthesis to achieve higher flexibility and improved quality.

Acknowledgement

This work has been sponsored in part by the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777, and by the U.S. National Science Foundation through grants OCI-0749217, CCF-0811422, CCF-0850566, OCI-0749227, and OCI-0950008. We are also grateful for NVIDIA's equipment donation.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Trans. Graph. 21* (2002), 483–490. 2

© The Eurographics Association 2011.

[ANSN06] ANGELIDIS A., NEYRET F., SINGH K., NOWROUZEZAHRAI D.: A controllable, fast and stable basis for vortex based smoke simulation. In *Proc. of SCA '06* (2006), pp. 25–32. 2

[BSHK04] BHAT K. S., SEITZ S. M., HODGINS J. K., KHOSLA P. K.: Flow-based video synthesis and editing. *ACM Trans. Graph. 23* (2004), 360–363. 2

[But87] BUTCHER J. C.: *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience, New York, NY, USA, 1987. 4

[CBL*09] CHANG Y., BAO K., LIU Y., ZHU J., WU E.: A particle-based method for viscoelastic fluids animation. In *Proc. of VRST '09* (2009), pp. 111–117. 2

[CLC*09] CHA M., LEE J., CHOI B., LEE H., HAN S.: A data-driven visual simulation of fire phenomena. In *Proc. of SIGGRAPH '09: Posters* (2009), pp. 1–1. 2

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *Proc. of SIGGRAPH '01* (2001), pp. 23–30. 2

[FL04] FATTAL R., LISCHINSKI D.: Target-driven smoke animation. *ACM Trans. Graph. 23* (2004), 441–448. 2

[FOA03] FELDMAN B. E., O'BRIEN J. F., ARIKAN O.: Animating suspended particle explosions. *ACM Trans. Graph. 22* (2003), 708–715. 2, 3

[FSJ01] FEDKIW R., STAM J., JENSEN H. W.: Visual simulation of smoke. In *Proc. of SIGGRAPH '01* (2001), pp. 15–22. 2, 3

[GBO04] GOKTEKIN T. G., BARGTEIL A. W., O'BRIEN J. F.: A method for animating viscoelastic fluids. *ACM Trans. Graph. 23* (2004), 463–468. 2

[Har03] HARRIS M. J.: *Real-time cloud simulation and rendering*. PhD thesis, University of North Carolina at Chapel Hill, 2003. Director-Lastra, Anselmo. 2

[HG09] HORVATH C., GEIGER W.: Directable, high-resolution simulation of fire on the gpu. *ACM Trans. Graph. 28* (2009), 41:1–41:8. 2

[HSF07] HONG J.-M., SHINAR T., FEDKIW R.: Wrinkled flames and cellular patterns. *ACM Trans. Graph. 26* (2007). 2

[JTCW07] JAMES D. L., TWIGG C. D., COVE A., WANG R. Y.: Mesh ensemble motion graphs: Data-driven mesh animation with constraints. *ACM Trans. Graph. 26* (2007). 2, 4

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Trans. Graph. 21* (2002), 473–482. 2, 4

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph. 22* (July 2003), 277–286. 2

[KTJG08] KIM T., THÜREY N., JAMES D., GROSS M.: Wavelet turbulence for fluid simulation. *ACM Trans. Graph. 27* (2008), 50:1–50:6. 2

[LCF05] LAI Y.-C., CHENNEY S., FAN S.: Group motion graphs. In *Proc. of SCA '05* (2005), pp. 281–290. 2

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23* (2004), 457–462. 2

[LLX*01] LIANG L., LIU C., XU Y.-Q., GUO B., SHUM H.-Y.: Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph. 20* (2001), 127–150. 2

[LR09] LONG B., REINHARD E.: Real-time fluid simulation using discrete sine/cosine transforms. In *Proc. of I3D '09* (2009), pp. 99–106. 2

[LZF10] LENTINE M., ZHENG W., FEDKIW R.: A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Trans. Graph. 29* (July 2010), 114:1–114:9. 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proc. of SCA '03* (2003), pp. 154–159. 2

[McG10] MCGUIRE M.: *G3D Innovation Engine*, 2010. http://g3d.sourceforge.net/. 7

[MTPS04] MCNAMARA A., TREUILLE A., POPOVIĆ Z., STAM J.: Fluid control using the adjoint method. *ACM Trans. Graph. 23* (2004), 449–456. 2

[MWGZ09] MA C., WEI L.-Y., GUO B., ZHOU K.: Motion field texture synthesis. *ACM Trans. Graph. 28* (2009), 110:1–110:8. 2

[ND01] NISHITA T., DOBASHI Y.: Modeling and rendering of various natural phenomena consisting of particles. In *Proc. of CGI '01* (2001), pp. 149–156. 2

[NFJ02] NGUYEN D. Q., FEDKIW R., JENSEN H. W.: Physically based modeling and animation of fire. *ACM Trans. Graph. 21* (2002), 721–728. 2

[NSCL08] NARAIN R., SEWALL J., CARLSON M., LIN M. C.: Fast animation of turbulence using energy transport and procedural synthesis. *ACM Trans. Graph. 27*, 5 (2008), 1–8. 2

[NVI07] NVIDIA C.: *Compute Unified Device Architecture Programming Guide*, 2007. 3

[PCS04] PIGHIN F., COHEN J. M., SHAH M.: Modeling and editing flows using advected radial basis functions. In *Proc. of SCA '04* (2004), pp. 223–232. 2

[PTSG09] PFAFF T., THUEREY N., SELLE A., GROSS M.: Synthetic turbulence using artificial boundary layers. *ACM Trans. Graph. 28*, 5 (2009), 1–10. 2

[SB08] SCHECHTER H., BRIDSON R.: Evolving sub-grid turbulence for smoke animation. In *Proc. of SCA '08* (2008), pp. 1–7. 2

[SP09] SOLENTHALER B., PAJAROLA R.: Predictive-corrective incompressible sph. *ACM Trans. Graph. 28* (2009), 1–6. 2

[SSSE00] SCHÖDL A., SZELISKI R., SALESIN D. H., ESSA I.: Video textures. In *Proc. of SIGGRAPH '00* (2000), pp. 489–498. 2, 4

[Sta99] STAM J.: Stable fluids. In *Proc. of SIGGRAPH '99* (1999), pp. 121–128. 2, 3

[Sta01] STAM J.: A simple fluid solver based on the fft. *J. Graph. Tools 6*, 2 (2001), 43–52. 2

[TLP06] TREUILLE A., LEWIS A., POPOVIĆ Z.: Model reduction for real-time fluids. *ACM Trans. Graph. 25* (2006), 826–834. 2

[WP10] WEISSMANN S., PINKALL U.: Filament-based smoke with vortex shedding and variational reconnection. *ACM Trans. Graph. 29* (July 2010), 115:1–115:12. 2

[WST09] WICKE M., STANTON M., TREUILLE A.: Modular bases for fluid dynamics. *ACM Trans. Graph. 28* (2009), 1–8. 2

[YOH00] YNGVE G. D., O'BRIEN J. F., HODGINS J. K.: Animating explosions. In *Proc. of SIGGRAPH '00* (2000), pp. 29–36. 2

[ZZJ*07] ZHANG L., ZHANG Y., JIANG Z., LI L., CHEN W., PENG Q.: Precomputing data-driven tree animation. *Comput. Animat. Virtual Worlds 18*, 4-5 (2007), 371–382. 2, 4

© The Eurographics Association 2011.