

Directable, High-Resolution Simulation of Fire on the GPU

Presentation Date: Sep 22nd

Chrissie Chi Cui

Outline

- Introduction
- Related work
- Highlights
- Input Particle System
- View-Specific Fire Refinement
- Fire Volume Rendering
- Experiment Results
- Conclusions & Future Work

Introduction

- Popular techniques in special movie effects
- Challenging visual characteristics for simulation
 - **Flickering nature, rapid time-evolution**
 - **Large gradients of temperature, turbulent**
 - **Vortical behavior in comparatively calm systems**
- Very fine spatial and temporal resolution
- Large Scale combustion Simulation for artist-driven, visual effects production: intractable
- Previous methods:
 - Compositing large amount of filmed, extracted fire:
Lack details, not adaptive, E.g. “The Lord of the Rings”
 - Solving 3D Navier-Stokes Equations for velocity, density and temperature: *Intractable for large spatial temporal resolution, E.g. “The matrix: Reloaded”*

Related Work

- Particle based:
 - *Model individual flames with chain of particles*
 - *Lack realistic details and coherence structures*
- Grid Simulation:
 - Solving 3D NV Equation
 - Hard to model the highly turbulent detail of real fire
- Combination of Particle and grid simulation:
 - A view dependant way
 - Limited to phenomena that can be decomposed into one or more 2-dimensional slices
 - Problematic when problems of rendering coherent structures from individual particles domain

Highlights

- Produce extremely detailed and can be integrated seamlessly into film-resolution images.
- A novel combination of coarse particle grid simulation with very fine, view-oriented refinement simulations performed on a GPU.
 - Split the refinement stage into separable parallel tasks by considering. perceptive limitations likely viewing behavior
 - Employ multiple independent GPUs for final simulations refinement
- A simple, GPU-based volume rendering scheme.
- Directability:
 - Allowing virtually any user-defined particle behavior as an input to the initial coarse simulation.
 - Minimum physical criteria enforced by the coarse stage
 - Utilizable on consumer-grade GPUs

Input Particle System

- A visually plausible fluid-like motion
- The particle simulation: fully directable, easy to control and fast
- Particle counts: from *20,000 to 100,000* per frame
- Particle Attributes:
 - Vectors: position(\vec{P}_i), velocity(\vec{V}_i)
 - Scalars: radius(R_i), fuel(K_i), mass(M_i), impulse(J_i), Age(A_i)
- Implement obstacles or disturbances as “impulse-only” particles
 - non-zero values for impulse,
 - zero values for fuel and mass,

Coarse Grid Step

- Small grid size: no larger than $50 \times 50 \times 50$.
- Efficiently obtaining an approximate of non-divergent particle velocity
- Inserted into the update step between the calculation of the new velocity and the calculation of the new position
- Augment PIC/FLIP by using wavelet decomposition on the velocity grid to produce multiple levels of detail.
 - Amplify vorticity at multiple scales
 - Accelerating convergence of the iterative incompressibility solution.
- Enforce incompressibility at each detail level from the lowest level
- Solving pressure Poisson Equation using Jacobi iterations

Coarse Grid Step Algorithm

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

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

View Specific Slice Refinement Overview

- Goal:
 - Add high resolution details
 - Add fire-like physics
- Observation Grounds for view specific refinement:
 - Fire is very fast-moving and turbulent, even in calm situations.
 - Fine details are short-lived and tend to change quickly.
 - The turbulent, fine details do not have a large visual impact on the overall motion of the flame flow.
 - From a particular viewpoint, fine variations of detail and movement perpendicular to the projection plane are not individually visible so they do not affect the large-scale flow very much.
- To-do-list:
 - Create a set of independent image planes parallel to a camera viewing plane;
 - Project attributes, e.g. forces and fuel, from the Coarse Grid Step particles onto these planes;
 - Solve the two dimensional, incompressible Navier-Stokes equations for each image plane on GPUs independently.

View Specific Slice Refinement Demo

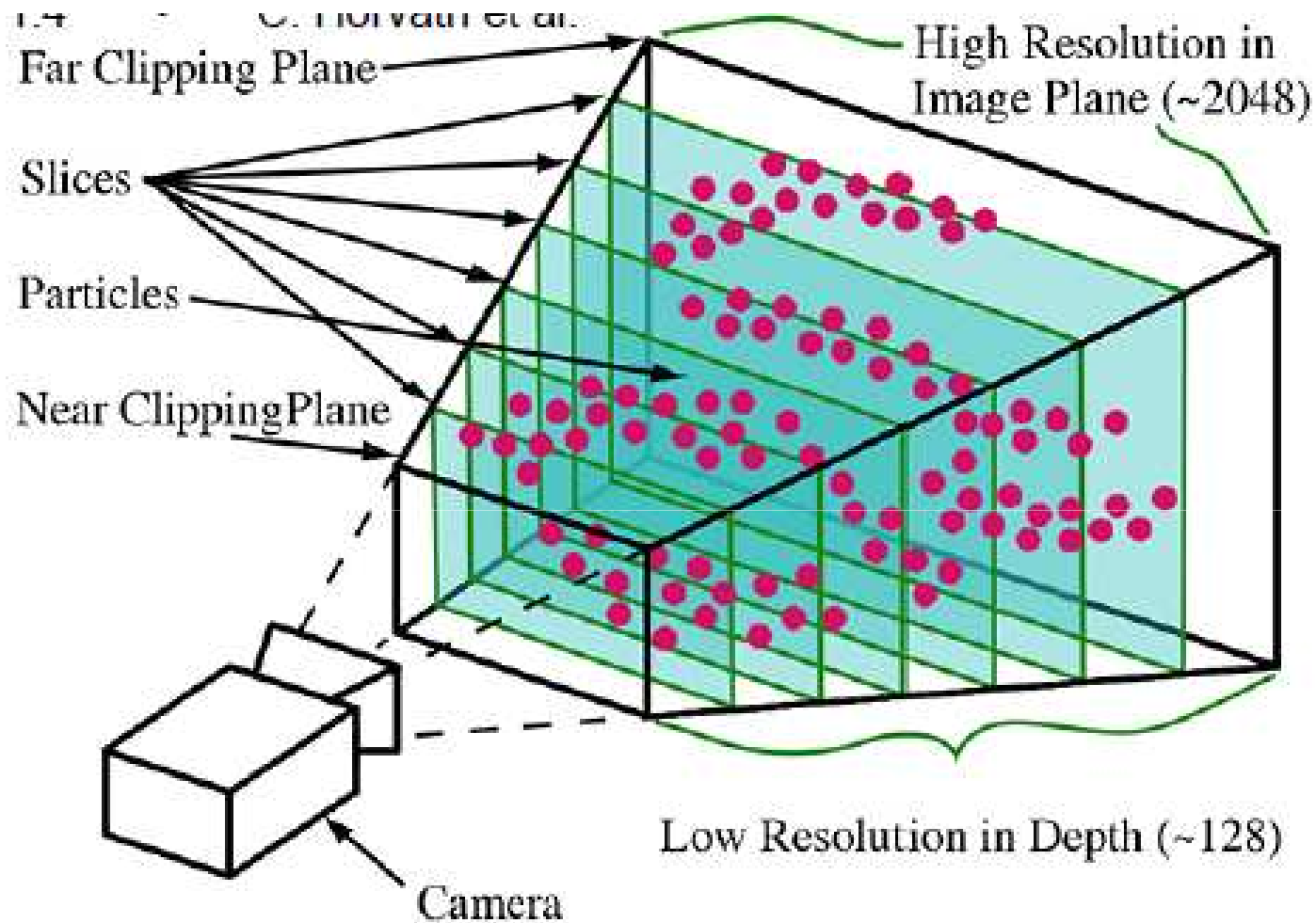


Figure. Refinement simulation slices are aligned to the projection plane. They are spaced coarsely and evenly along the projection axis. Slices have very high resolution in the dimension parallel to the projection

View Specific Slice Refinement steps

Algorithm 2 Fire Refinement Slice Simulation, Single Time Step

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

GPU Configuration

- The 2D refinement simulations are computed entirely on the GPU
- Built in OpenGL2.1 using GLSL

Slice Refinement Details I

- Compare to the work done by Harris:
 - **New ordering of the major sub-steps in the update step**
 - **The addition of temperature, cooling, and thermal buoyancy**
- Inputs from Particles: Particle Fuel, Particle Mass, Particle Weighted Velocity
- Simulation State: Density, Temperature, Texture, Fuel, Velocity, Artificial Pressure
- Output per Slice:
 - **Image I: simDensity (Ch.I), simTemperature (Ch.II), textureDetail (Ch. III) and fluidFuel (Ch. IV)**
 - **Image II: simVelocityU (Ch. I) and simVelocityV (Ch. II)**
- Particle Projection onto refinement slices
 - **Create three planes of data: *Fuel(adding temperature), Mass(adding density), WeightedVelocity(an impulse, ultimately adding velocity).***
 - **Used additively with the existing simulation**
 - **Each particle's contribution to the slice is weighted based on its perpendicular distance from the slice, using a Gaussian kernel.**

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

d_i **Perpendicular distance between particle i and this slice**
 Δz **The uniform distance between slices**

Slice Refinement Details II

- Cooling, dissipation, heating and density step
 - the first of the “slab” operations: several two-dimensional data planes are operated upon to create an output data plane of exactly the same rectangular size
 - Highly parallelizable on GPUs
 - Cooling: $T^* = T - \Delta t * C_{cool} * (T/T_{max})^4$
 - Dissipate: $Fuel^* = Fuel * (1 - \gamma_{fuel})^{\Delta t}$ $Mass^* = Mass * (1 - \gamma_{mass})^{\Delta t}$
 - Heating: $Fuel^{**} = maximum(Fuel^*, ParticleFuel * T_{fuel})$
 - Density: $Mass^{**} = maximum(Mass^*, ParticleMass * K_{density})$
- Boundary Conditions
 - Collision only resolved in the input simulation
 - “impulse only particles” still affect simulation velocity in the refinement stage
 - Neumann BC
- Semi-Lagrangian Advection
 - Unconditional stable for any size of time step
 - Temperature, Detail Texture, Fuel and Velocity
- Detail Texture Synthesis
 - Goal: replace the highest frequencies of lost detail
 - A rapidly evolving fractal consisting of several octaves of time-evolving noise is added to the advected texture plane per time step
 - Zero average noise function across all spatial and temporal dimensions
 - Spatial and temporal maximum frequency of the fractal noise
 - GPU Generation: 4D Simplex Noise

$$\vec{v}_{max} = \left(\frac{\Delta x}{2}, \frac{\Delta y}{2}, \frac{\Delta z}{4}, \frac{\Delta t}{2} \right)$$

Slice Refinement Details III

- Velocity Damping:

- Applied to the velocity prior to vorticity, turbulence and incompressibility calculations

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

- Vorticity and Turbulence

- Combine the vorticity confinement and curl-noise turbulence techniques into a single simulation step

- Procedural turbulence scalar field Φ

$$\Psi = \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}$$

$$\Phi = \text{userDefinedScalarTurbulenceField}()$$

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

$$\vec{J} = \text{curl}(K) = \left(\frac{\partial K}{\partial y}, -\frac{\partial K}{\partial x} \right)$$

$$\vec{V}^* = \vec{V} + \Delta t * \vec{J}$$

- User-defined noise function here is subject to the Nyquist limits $\vec{v}_{max} = \left(\frac{\Delta x}{2}, \frac{\Delta y}{2}, \frac{\Delta z}{4}, \frac{\Delta t}{2} \right)$
- 4D Simplex noise

- Thermal Buoyancy

- The velocity is updated by adding an “upward” impulse

$$\vec{V}^* = \vec{V} + (\Delta t * K_{buoy} * (T - T_{room})) * \vec{V}_{up} \quad T_{room} \text{ Ambient “room” temperature}$$

$$\vec{V}_{up} \text{ the direction of a user supplied unit vector } K_{buoy} \text{ user-defined constant}$$

- Enforce Incompressibility

- Step I: Divergence is calculated based on the new velocity
- Step II: An artificial pressure term is iteratively solved for using a Jacobi solver

Fire Volume Rendering

- Volume rendering using a simple GPU based renderer
 - **Calculate light from temperature using the Planck Blackbody Radiation function**
 - **Compute opacity as an exponential function of density and the spacing between slices**
 - **The texture channel is selectively used to modulate temperature and density (do not use the fuel information)**
- Velocity images used together with the semi Lagrangian advection slab operation to distort the slice images for the purposes of creating motion-blur.
- Redraw the stack of slice images over and over again into an accumulation buffer
- Holdout mattes are incorporated during the near-to-far rendering traversal
- The heat distortion effect
 - **A post process in a compositing application**
 - **Warp the background outwards by a blurred version of the brightness of the fire render.**
 - **Standard compositing technique for simulating heat distortion**

Experiment Results



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

- Three simulations differ mostly in the direction of the input particle systems and the settings for exposure and density gain in the final rendering stage.
- Approximately 25,000 particles in the coarse grid simulation
- Refinement slice resolution of 2048×1556
- NVidia Quadro 5600 GPUs running on 2.2 GHz AMD Dual-Core Opteron (SUSE Linux, version 9.3.1)
- An average of 10 seconds per frame, per slice
- Distributed across 10 GPUs
- The simulations generally took between 2 to 4 hours to complete
- Renders taking an hour to two.

Conclusions & Future Work

- **Success Story**

- employed on three feature films so far
- A wide variety of application: from fireballs to walls of angry flame
- Short artist learning curve
- Save the turnaround times for gigantic, high-resolution simulations from days to hours

- **Limitation**

- One limitation of our current system is that collision objects are not explicitly handled during the refinement stage.
- Current solution for the system's handling of rapidly moving cameras is tedious and imperfect workaround

- **Future Work**

- Rendering from locked-off cameras and then projecting onto geometry in the scene
- Allow for a more complete communication between slices, while still maintaining the many aesthetic advantages of the view-oriented detail that characterizes our technique

Thanks for your attention. 😊

Questions???