Directable, High Resolution Simulation of Fire on the GPU

As proposed by Christopher Horvath and Willi Geiger

Presented by Ben Guitreau

Difficulties in simulating fire

- High detail
- High velocity due to flickering nature of fire
- Turbulence created by large gradients of temperature
- Large grid-size in traditional grid-based simulations
- Long simulation times

Previous methods, Film/Extraction of fire and smoke

- Film and extract fire and smoke elements that are then rendered as camera-facing sprites
  - Compositing may look believable
  - Sprites need to be large to be visually effective
  - Unfortunately, no interaction with environment that is not a part of the original filming
  - Balrog in “The Lord of the Rings”

Previous methods, 3D Grid with Navier-Stokes equations

- Three-dimensional grids using Navier-Stokes equations for velocity, density, and temperature
  - Augmented with texture details and volume-rendered
  - Able to interact with environment and simulation is easily directable
  - Due to large grid-sizes, large amounts of resources and render-time are needed
  - Hellboy, Hellboy 2, and The Matrix: Reloaded

Proposed method

- Makes use of techniques used in both of the previous methods
- Particle simulation that stores scalar attributes
  - Fuel
  - Mass
  - Impulse
- Allow any user-defined particle behavior as input
- Minimum physical criteria enforced by coarse stage
- Utilizable by consumer-grade GPUs

Stage One

- Coarse grid
  - Alters the final position and velocity of each particle at the end of the time-step
- Enforce incompressibility
- Vorticity may be amplified
  - Axial vector that describes the local rotation of a fluid near a point as it would be seen by an observer located at that point and traveling along the fluid

Stage Two

- Evenly-spaced, two-dimensional slices
  - Particles are projected onto the slices
  - Represent independent 2D grids on which Navier-Stokes equations are performed on the GPU
  - Slices are computed across GPU farm

Input Particle System

- Requires a visually plausible fluid-like motion
- Fully directable, easy to control, and fast enough for many iterations
- Attributes:
  - Vectors: position ($\vec{P}_i$), velocity ($\vec{V}_i$)
  - Scalars: radius (R_i), mass (M_i), fuel (K_i), age (A_i), impulse (J_i)
- Creating obstacles
  - Non-zero value for impulse
  - Zero values for fuel and mass
- Time-step
  - user-defined rules are executed
  - New velocity computed and integrated to create a new position

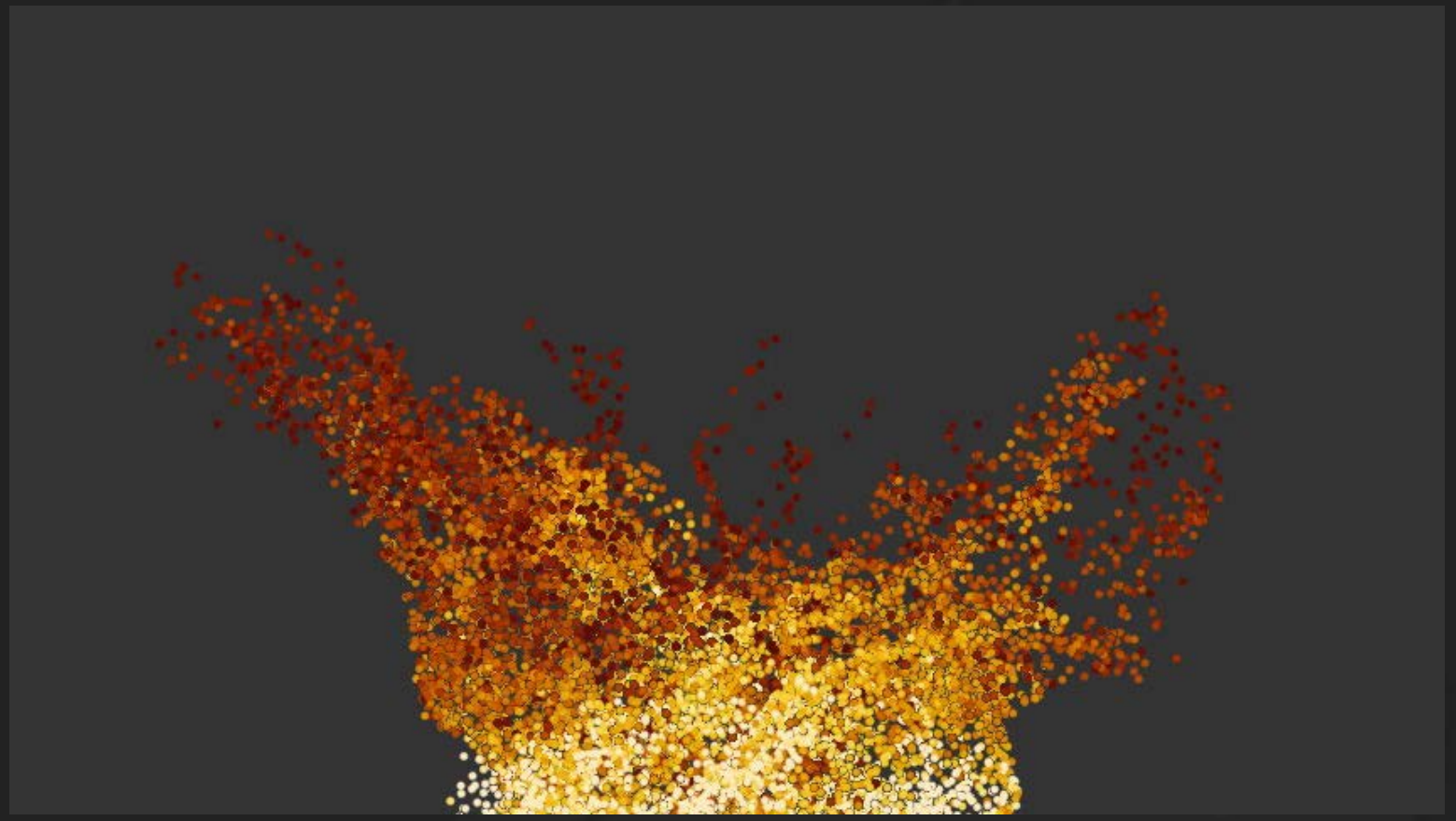
Coarse Grid Step

- Grid size usually 50 x 50 x 50
- Inserted into the update step between calculation of new velocity and position
- Augment PIC/FLIP (Particle-In-Cell/Fluid-Implicit-Particle)
  - Using wavelet decomposition on velocity grid producing multiple levels of detail
    - Amplifies vorticity
    - Accelerates convergence of iterative incompressible solution
- Incompressibility enforced at each detail level starting at the lowest level and moving upwards
- Use Poisson solver with simple Jacobi iterations for solving the non-divergence equation $(\nabla * \vec{U}^* = 0)$, where $(\vec{U}^* = (u, v, w))$ is the grid velocity in three dimensions

Coarse Grid Step Algorithm

Computes approximately non-divergent velocity $\vec{V}_i^*$ from hypothetical $\vec{V}_i^{**}$

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



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

View-Specific Fire Refinement

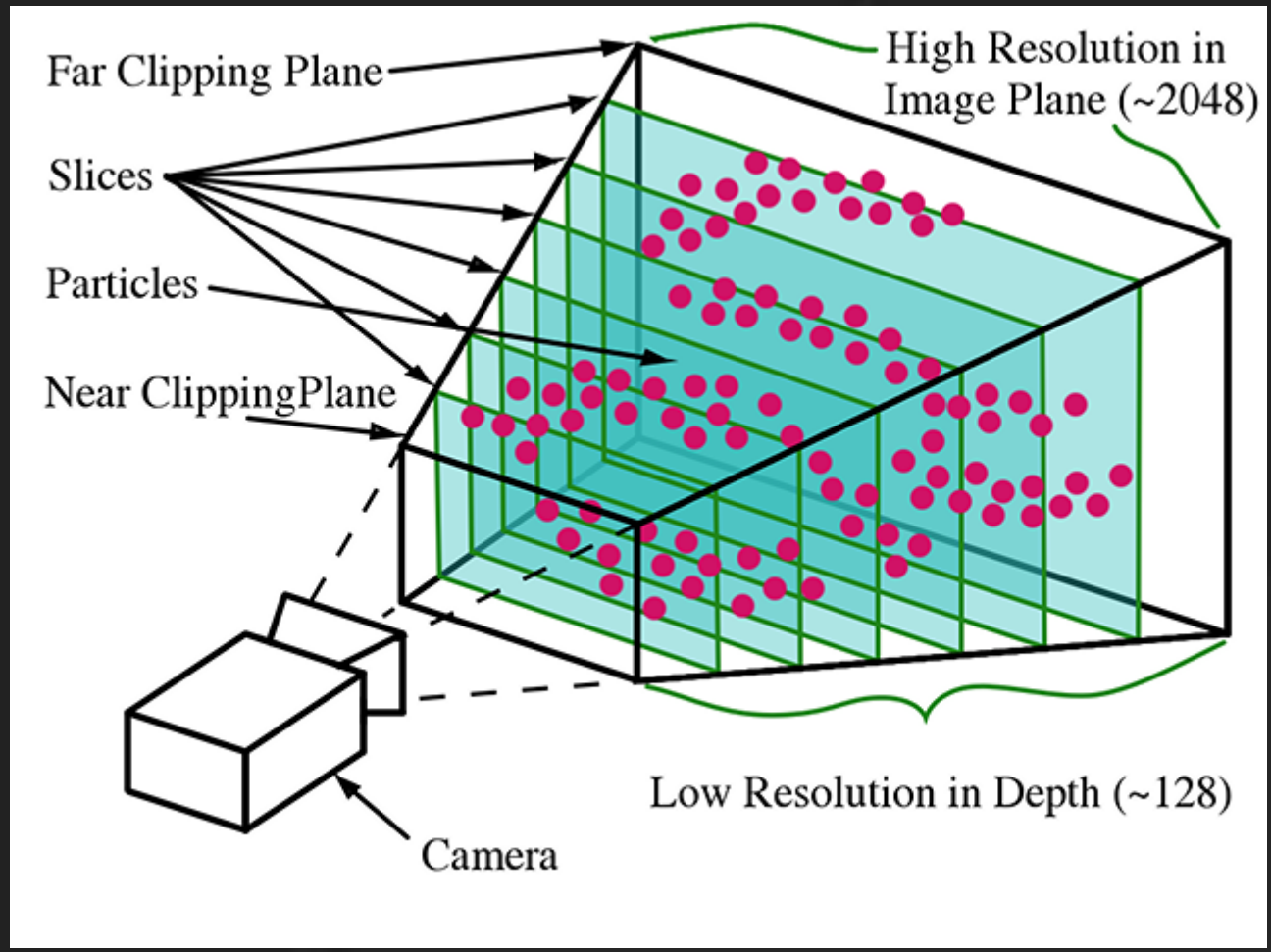
- Conclusion of coarse grid step has produced a computed particle set consisting of position, velocity, radius, fuel, mass, impulse, and age
- GPU configuration
- Slice Refinement Overview
  - Particle projection
  - Cooling, Dissipation, Heating, Density
  - Boundary Conditions
  - Semi-Lagrangian Advection
  - Detail Texture Synthesis
  - Velocity Damping
  - Vorticity and Turbulence
  - Thermal Buoyancy
  - Enforce Incompressibility

Refinement Simulation Process Algorithm

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

- Two-dimensional refinement simulation computed on GPU
  - Geometry-based graphics processing
  - GPGPU
    - NVidia – Cuda
    - ATI – OpenCL
- System built on OpenGL 2.1 using GLSL



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

Slice Refinement Overview I

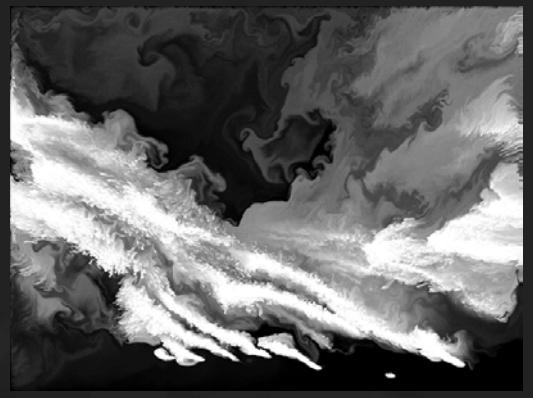
- Time-stepped, two-dimensional fluid simulation iterating across a user-defined time-range and solving the incompressible Navier-Stokes equations
- Simulation state consists of identical rectangular planes of data

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

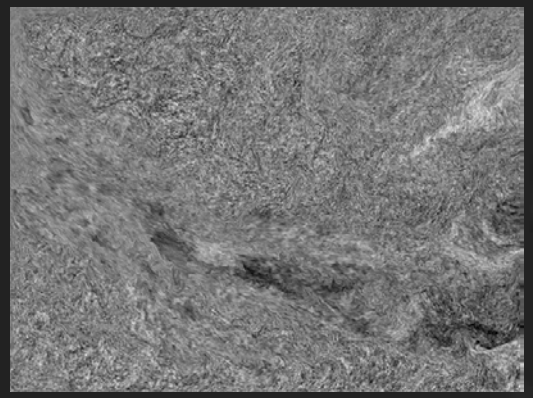
Slice Refinement Overview II

- Each frame produces two four-channel floating-point images per slice
  - First slice: Density, Temperature, Texture Detail, and Fuel
  - Second slice: VelocityU and VelocityV
- 20 substeps per frame to handle a normal range of velocities

Density

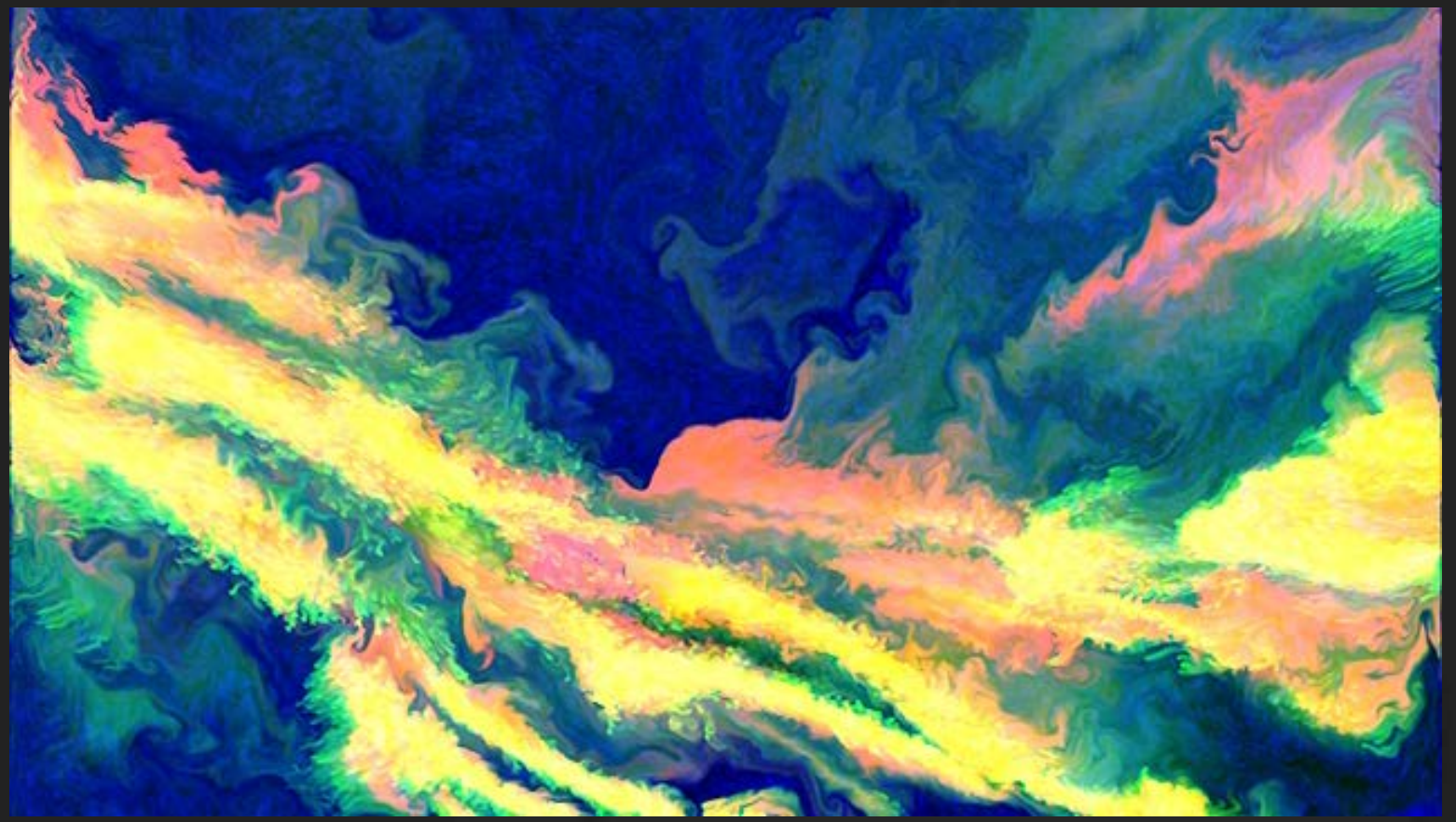


Temperature



Texture

Fuel



Primary slice refinement output.

Combined density, temperature, and texture.

Slice Refinement Overview III

- Particle Projection
  - Particles projected onto refinement slices creating planes
    - Fuel – adds temperature to system
    - Mass – adds density
    - WeightedVelocity – treated as an impulse adding velocity
  - These planes are used additively
  - Vertex shader handles matrix transformations
  - Gaussian weighting kernel
    - $w_i(d_i) = e^{(-d_i^2/(4\Delta z)^2)}$

      where:

      d_i =Perpendicular distance between particle i and slice

      Δz =Uniform distance between slices

Slice Refinement Overview IV

- Cooling, Dissipation, Heating, and Density
- First of “slab” operations: two-dimensional planes operated on to create an output data plane
- Parallelizable on the 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 $Mass^{**} = maximum(Mass^*, ParticleMass * K_{density})$
  - Where $C_{cool} =$ cooling coefficient
    $T_{max} =$ theoretical maximum system temperature
    $\gamma =$ decay
    $T_{fuel} =$ compbustion temperature
    $K_{density} =$ user-defined density

Slice Refinement Overview V

- Boundary Conditions
  - Collisions taken care of in input particle simulation
  - Neumann boundary conditions
- Semi-Lagrangian Advection
  - Allows for fluid grid advection steps performed unconditionally and stably for any size time-step
  - Temperature, Detail Texture, Fuel, and Velocity

Slice Refinement Overview VI

- Detail Texture Synthesis
  - Replace highest frequencies of lost detail
  - Texture plane is initialized to black
    - At each time-step, rapidly evolving fractal added to advected texture plane
  - Noise functions have average value of zero
  - Set frequency of finest octave of noise in (x,y,t) to Nyquist limit of $(\frac{\Delta x}{2}, \frac{\Delta y}{2}, \frac{\Delta t}{2})$
  - Fractal noise has maximum frequency of

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

  - For GPU-based noise: 4D Simplex Noise

Slice Refinement Overview VII

- Velocity Damping
  - User-defined damping prior to vorticity, turbulence, and incompressibility calculations
    - For velocity $\vec{V}$, timestep Δt, velocity damping coefficient K_{damp},
      $\vec{V}^* = \vec{V} * (1 - K_{damp})^{\Delta t}$
- Vorticity and Turbulence
  - Vorticity confinement and curl-noise turbulence combination for a single step
    $\Psi = \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}$
    $\Phi = \text{userDefinedScalarTurbulenceField}()$
    $K = \Phi + (K_{vort} * \Psi)$
    $\vec{J} = curl(K) = (\frac{\partial K}{\partial y}, -\frac{\partial K}{\partial x})$
    $\vec{V}^* = \vec{V} + \Delta t * \vec{J}$
  - User-defined noise function uses same Nyquist limits and same style of 4D Simplex noise as in Detail Texture Synthesis

Slice Refinement Overview VIII

- Thermal Buoyancy
  - Update velocity with an upward impulse
    - For temperature T, ambient room temperature T_{room}, user-defined constant K_{buoy}, and direction of user-supplied unit vector $\vec{V}_{up}$:

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

- Enforce Incompressibility
  - Step One: Divergence calculated on new velocity
  - Step Two: Artificial pressure term is iteratively solved using a Jacobi solver

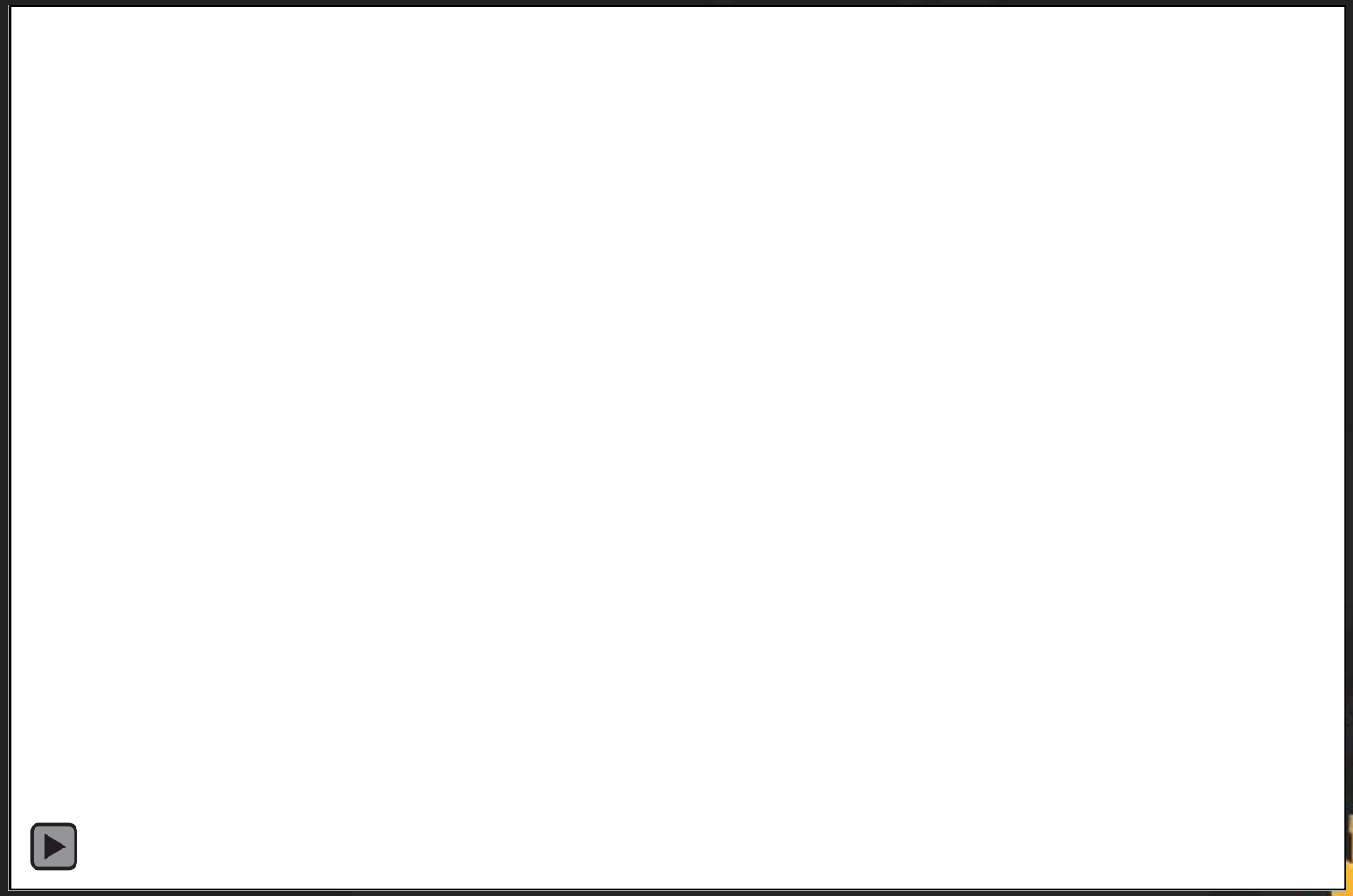
Fire Volume Rendering

- Volume rendering is performed using a GPU-based renderer
  - After each slice is computed fully and saved to disk for every frame of every slice
  - Each slice is already camera oriented and aligned to frustum
  - Convert temperature to light using Planck Blackbody Radiation
  - Convert opacity as exponential function of density and spacing between slices
  - Texture channel selectively used to modulate temperature and density
  - Fuel not used
- Velocity images used in conjunction with semi-Lagrangian advection slab operations
  - Achieves motion-blur
- Redraw stack of slice images continuously into accumulation buffer
  - Render time for 2000 images approximately 30-40 seconds to load, render, and save
- Fire renders composited fully by incorporating holdout mattes during near-to-far rendering
- Heat distortion effect is a post-process

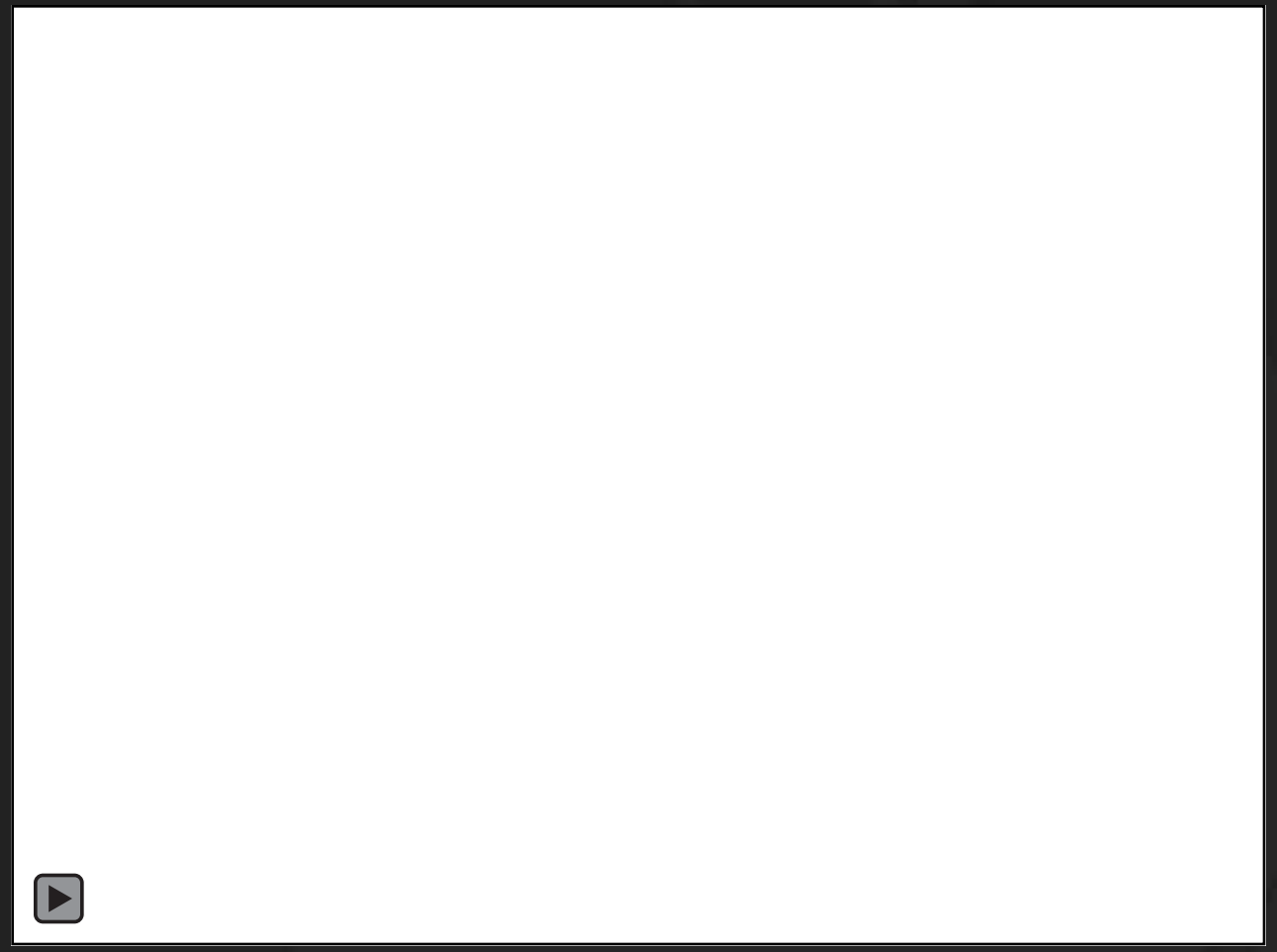
Simulation

- Three simulations
- 25000 particles in coarse grid simulation
  - Between 50 and 200 frames
  - Took no longer than 10 minutes to execute and save to disk
- Refinement slice resolution of 2048 x 1556
- Farm:
  - NVidia Quadro 5600 GPUs
  - 2.2 GHz AMD Dual-Core Opteron
  - SUSE Linux 9.3.1
- Timing:
  - 10 seconds per frame per slice on 10 GPUs
  - 2 to 4 hours per simulation plus 1 to 2 hours for rendering

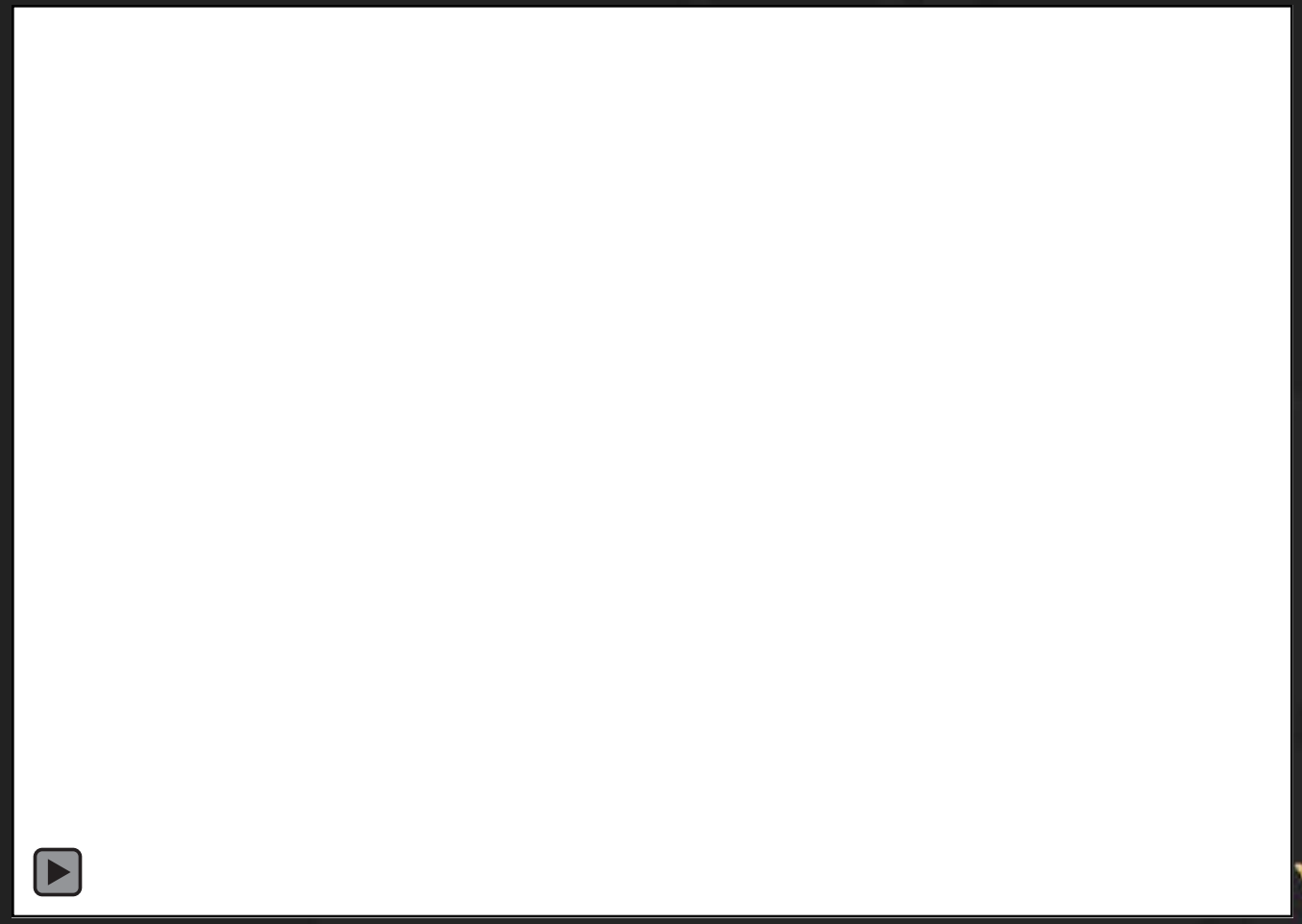
Fast moving fireball with sparks



Twisting campfire with rotation



Wall of fire



Conclusions

- Used on three feature films as of paper authoring
- Artist learning curve is small
- Short turnaround time for large, high-resolution simulations
- With increasing GPU storage and speed, simulations should be able to be completed using 1 machine and 1 GPU