EG UK Theory and Practice of Computer Graphics (2007)
Ik Soo Lim, David Duce (Editors)

Level of Detail for Physically Based Fire

Odd Erik Gundersen and Lars Tangvald

Norwegian University of Science and Technology, Trondheim, Norway

Abstract

In this paper, we propose a framework for implementing level of detail for a physically based fire rendering running on the GPU. The physics of the fire is simulated using a fluid solver and combustion modelling, and the fire is visualised using a particle system. Our preliminary results indicate that by adjusting the simulation domain and particle system, performance can be increased without noticeably degrading the fire visually when it is far from the camera.

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

1. Introduction

Computer games push the limits of real-time graphics, and current titles like *Gears of War*™ and *The Elder Scrolls: Oblivion*™ have stunning graphics. Still, there is a long way to go until virtual environments are mistaken for real ones. Generally, games that take place in cities look more realistic than games that take place in natural environments. Contrary to human-build structures, natural phenomena are very complex, and thus hard to visualise in a realistic manner, especially in real-time. Until natural phenomena are rendered in a convincing way, virtual environments will look exactly like that; virtual. Our belief is that rendering of natural phenomena should be based on the laws of physics. However, there are problems related to this philosophy as solving equations describing the laws of physics often are computationally very demanding. This is because either lots of small equations need to be computed or that the solutions only can be approximated numerically using methods needing several iterations.

Our focus is on realistic rendering of fire. As many other natural phenomena, like smoke, water, and explosions, fire can be simulated using computational fluid dynamics (CFD). CFD methods used for creating realistic flames have generally been too computationally demanding for real-time implementations [TOT*03] [LF02]. But as hardware is becoming increasingly more powerful, this is changing. In recent years, numerous methods for rendering realistic fire that run in real-time have been published [ZWF*03] [AH05] [BF06].

Although the papers referenced above describe real-time techniques for rendering physically based fire, they are not easily utilised in a virtual environment. This is because they use large amounts of the resources available to render the fire only, even when the fires are far away from the camera and are not important to the scene. Similar problems related to object geometry are solved using level of detail (LOD) algorithms. LOD algorithms seek to reduce the detail of object geometry without creating a visual difference. Reducing the detail of the objects leads to shorter processing time, which again leads to higher frame rates. However, there is a lack of research on LOD algorithms for fluid dynamics system in the literature. We aim to develop a framework that enables the inclusion of a physically based fire into a virtual environment.

Our goal is to enhance our previous work on real-time fires presented in [RSG06] and [GRS06] with a LOD algorithm that reduces the computational cost without creating a notable difference in visual quality. As both the simulation and visualisation is completely executed on the GPU, this is a requirement for our LOD algorithm too. The framework is still under development, and the results presented in this paper are preliminary. In spite of that, the results indicate that we are on the right track.

Our contribution. The collection of methods presented in this paper is a first step to towards a fully GPU implemented LOD framework for rendering physically based real-time fires. Solutions for both the simulation part and visualisation part of the fire rendering process are presented. Dy-

namically resizing the resolution of the simulation domain together with simulation step skipping are the methods proposed for reducing the computational cost of the simulation, while particle size adjustment and particle count resizing are used for reducing the cost of the visualisation step. The framework can easily be extended to work for animating both smoke and explosions.

This paper is organised as follows. After a brief overview of related work, an introduction of the fire rendering algorithm is presented. Then, the LOD framework is proposed followed by preliminary results. The paper concludes with summary and future work.

2. Related Work

As physically based fires traditionally have been too computationally demanding for real-time applications, several non-physical methods have been developed. The most prominent non-physically based method is presented in [Ngu04]. They use video-textured sprites for creating believable raging fires with smoke in real-time. In order to add variety to the flames, two flame animations are combined in various ways. Another non-physical approach is presented in [KCR00]. They use volume rendering in combination with a set of textures to visualize animated amorphous materials such as fire, smoke, and dust. Dynamics and illusion of motion are created through cycling the textures in each voxel. In [FMF06], a method that uses a photometric solid defining luminous intensities for a set of zentihal and azimuthal directions is presented. The intensities are stored in a 2D texture and by rotating this texture the fire is animated.

We have not been able to find any previous work on LOD algorithms used with physically based fires. There are however a plethora of literature published on LOD. LOD for graphics is generally divided into three different types of methods according to [LRC*03]. These are discrete, which uses different versions of objects generated before starting the application, continuous, which generates new versions of the object during run-time, and view-dependant, which generates multiple detail levels of the same object during run-time.

[HD04] presents a list of of conditions that determines the importance of an object to the current frame. Among these conditions are velocity of the objects, distance from camera, size, and whether the objects are completely or partially visible. LOD has been applied to object geometry [CCSS05] [DHDS05] including terrain [Hop98] [RHSS98], physics [HC97] [FC97], and autonomous behaviour of objects not controlled by the user, like computer played characters and effect from weapons [OCV*02]. [OFL01] presents a method for clustering particles together to increase calculation efficiency of particle systems.

3. Rendering Physically Based Fire

The fire rendering process is divided into two parts. First, the fire is simulated, and then the simulation is visualised. Simulation is the most computationally demanding process because it solves a fluid system. This fluid system evolves four fields controlling the temperature, the amount of exhaust gas, the amount of fuel, and the velocity in the simulation domain. The simulation domain is the limited volume where the fire can burn.

After simulation, the state of the fluid system is visualised using a particle system of textured particles. The particles flow through the simulation domain guided by the velocity field. For each voxel in the simulation domain, a fire colour is computed and stored in a table called the fire colour field. The particle's texture colour is looked up in the fire colour field based on the particle's position.

The rest of this chapter gives a brief overview of the fire rendering process. For a detailed description of the complete method, see [RSG06].

3.1. Simulating Fire

The fire is simulated by evolving a fuel gas field, an exhaust gas field, and a temperature field in co-evolution with a velocity field. These fields are governed by the Navier-Stokes equations and the combustion process, which converts fuel gas to exhaust gas and heat when the temperature exceeds a certain threshold. Buoyancy due to heat then causes the hot exhaust gas to rise, which in combination with vorticity confinement, cause the characteristic fire-like motion.

3.1.1. The Simulation Domain

We use a voxel data structure to represent the simulation domain and will refer to each unit as a cell. The simulation domain limits the volume where a fire is simulated. There are two different kinds of cells in the simulation domain; interior cells and boundary cells. Each cell contains a corresponding field value. When discretizing the fields into cells, the field values are defined in the centre of the cells and assumed to be uniform inside each one. As for boundary conditions, the boundary cells are set to 0 in the density fields and to the wind vector for the velocity field.

3.1.2. Velocity field

The velocity field $\mathbf{u}$ is governed by the Navier-Stokes equations for incompressible flow with zero viscosity, also known as the Euler equations:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\,\mathbf{u} - \nabla p + \mathbf{F} \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{2}$$

The first term on the right-hand side of equation 1 is the self-advection of the velocity causing velocity to move along itself. The second term, $-\nabla p$, is the pressure gradient causing velocity to move from areas of high pressure to areas of low pressure. The pressure field is used as a correcting term ensuring that equation 2 holds. Equation 2 is the non-divergence condition, which states that the velocity field should be mass conserving. The last term on the right hand side of equation 1 is the external force acting on the velocity field. The external force actually consists of several separate forces as shown in equation 3:

$$\mathbf{F} = f_{vorticity} + f_{gravity} + f_{buoyancy}, \tag{3}$$

where $f_{vorticity}$ is the vorticity confinement force, $f_{gravity}$ is the gravity force due to fuel and exhaust gases, and $f_{buoyancy}$ is the buoyancy force due to heat.

3.1.3. Fire density fields

The three separate scalar fields specifying the amount of fuel gas, exhaust gas, and heat distributed throughout the simulation domain are collectively referred to as the fire density fields. These three scalar fields are evolved by the same equation:

$$\frac{\partial d}{\partial t} = -\mathbf{u} \cdot \nabla d + \kappa_d \nabla^2 d - \alpha_d d + S_d + C_d \tag{4}$$

The parameter d is a scalar quantity that represents either the amount of fuel gas, exhaust gas, or temperature in a cell in the simulation domain; denoted by g, a or T respectively. Equation 4 describes the evolution of a scalar field over time in the simulation domain as the velocity field $\mathbf{u}$ affects the scalar field. We use a slightly modified form of the equations described in [Sta99].

The first term on the right-hand side in equation 4 governs the advection of the scalar quantity d by the velocity field $\mathbf{u}$, while the second term governs the diffusion of the scalar quantity d. κ_d is the diffusion constant controlling the amount of diffusion associated with each of the density fields. Furthermore, the third term governs the dissipation of the scalar quantity d where α_d denotes the dissipation rate. The dissipation rate ensures that fuel gas, exhaust gas, and temperature will decrease over time. S_d denotes a source term used for increasing the scalar quantity d. Only the fuel gas field has a source, which is used for injecting fuel, while temperature and exhaust gas are produced solely in the combustion process. C_d is the combustion term that controls the effect of the combustion process on a specific density field.

3.2. Visualising Fire

Using a precomputed black-body radiation lookup table, a fire colour field is computed based on the exhaust gas and temperature fields. The fire is visualised using a particle system, and the particle positions are updated based on the velocity field, and the particle colours are read from the fire colour field. Smoke is implemented in a separate particle system, and the light intensity is based on the fire colour field.

3.2.1. Computing the fire colour field

We use Planck's formula for black-body radiation (equation 5) in order to calculate the intensity radiated by the hot exhaust gas.

$$B_\lambda(T) = \frac{2\pi hc^2}{\lambda^5 \left(e^{\frac{hc}{\lambda kT}} - 1 \right)} \tag{5}$$

By using the wavelengths of red, green, and blue light and the temperature of the gas, we calculate the three intensities B_{red}, B_{green}, and B_{blue}. These intensities have a very high dynamic range whereas the resulting colour should have a limited dynamic range suitable for display on traditional computer monitors. To map the given intensities between 0 and 1, we use the exponential mapping function from [Mat97]:

$$n = 1 - e^{\frac{-L}{L_{average}}} \tag{6}$$

L is the original intensity, and $L_{average}$ is a constant controlling the overall brightness. The resulting intensity n will be in the range $[0, 1\rangle$.

Equations 5 and 6 are used to precompute black-body radiation colour values for a user specified range of temperatures, which are stored in a one dimensional lookup table.

At the beginning of each visualization step, the exhaust gas and temperature fields are used in combination with the black-body radiation lookup table in order to compute the fire colour field. This is done for each cell in the simulation domain. Equation 7 shows how the colour $\mathbf{c}$ in the fire colour field is computed based on the temperature T, exhaust gas a, and a temperature scaling factor T_{scale}, which is used to control the resulting brightness of the fire. *lookup* is the black-body radiation lookup table.

$$\mathbf{c} = a \times lookup\,(T_{scale} T) \tag{7}$$

3.2.2. Visualisation using two particle systems

We visualize the fire and the smoke using separate particle systems defined in the simulation domain. By computing a separate smoke field instead of trying to incorporate the smoke into the fire colour field, we get more control over the appearance and the amount of smoke produced in the fire. Each particle represents a small element of the fire or the

smoke and has a set of associated variables: spawn position, current position, initial spawn delay, current velocity, and colour. Spawn position and initial spawn time are given at the beginning of the simulation, whereas the other variables are dynamically updated. A particle's colour is specified by an RGBA colour value. The amount of smoke is computed based on the temperature at a given cell and the amount of exhaust gas, quite similar to how the fire colour field is generated.

Initially, after the given spawn delay, a particle's position is set to the spawn position of the particle. A simple Euler step is later used to update a particle's position:

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i \delta t, \tag{8}$$

where $\mathbf{x}_i$ and $\mathbf{v}_i$ are the position and velocity of particle i respectively and δt is the timestep. Based on the particle position, the particle's velocity and colour are found by interpolating samples from the discretized simulation velocity and fire colour fields.

When a particle's intensity drops below a certain threshold, it is respawned by resetting its position to its spawn position. A minimum initial lifetime ensures that the particle is not respawned before it has had a chance to enter the fire. Particles are textured to create more low-level detail.

3.3. Additional Properties

Fires interact with their surroundings. Our fire implementation reacts to dynamic wind, looks realistic when moved, and illuminates surrounding objects. Wind is generated dynamically by utilising Perlin noise curves [Per85]. The wind vector operates on the simulation domain. Simulation domain advection is used for moving the fire around. The simulation domain and the particles are advected using the distance vector, which is the distance between the new and old position.

In addition, lights with dynamically set intensities are implemented. One or more point light sources are placed inside the simulation domain, and based on their position in the simulation domain the light intensities are computed from the amount of exhaust gas and temperature at their respective position. The dynamic lighting produces the flickering light often associated with fires.

4. Level of Detail Framework

As shown above, rendering of fire is a complex task with several steps utilising different technologies. Thus, there are several possible ways to reduce the computational load. We propose four different strategies, and they are:

Simulation domain resizing: The resolution of the simulation domain is changed according to the distance between the fire and the viewer. At close range, the simulation is done at maximum resolution, while reduced with prolonged range.

Simulation step skipping: By skipping simulation steps when the fire is not significant to the view, lots of resources can be saved. The visualisation steps are *not* skipped though.

Particle count adjustment: The amount of particles in the particle system is reduced when the camera moves away from the fire and increased when moving towards it.

Particle resizing: The size of the particles is increased when reducing the particle count and decreased when the particle count rises.

The following sections describe in detail the different strategies used in our LOD algorithm for physically based fire rendering.

4.1. Overall LOD

The overall LOD of an object is a value describing how detailed the object should be rendered. Two LOD conditions determines the overall LOD, and these are view culling and distance.

View culling determines whether the fire is inside the view or not. The angle between the camera's view direction and the line from the camera to the fire is calculated. If the angle exceeds a certain value, the fire is determined to be outside the view, and no further calculations are performed.

The distance between the camera and the fire is calculated. The detail level of the fire decreases linearly as the distance increases. This strategy is based on the assumption that a fire seen at a distance need less details to be visually convincing. The LOD value for distance is calculated by the formula:

$$lod = \frac{b}{d}, \tag{9}$$

where b is the maximum distance the camera can be away from the fire while still rendering the fire at full detail, and d is the distance between the fire and the camera.

4.2. Simulation Domain Resizing

The most computational demanding step in our fire rendering algorithm is solving the fluid dynamics system. Therefore, reducing the amount of computations associated with solving the fluid dynamics system is most important. Our solution is to resize the simulation domain according to how important the fire is to the scene. By reducing the size of the simulation domain, the Navier-Stokes equations are solved fewer times for each simulation step, and thus the fire need less resources to render.

The resolution of the simulation domain is changed according to the distance between the fire and the viewer. At

close range, the simulation is done at maximum resolution, while it is reduced when prolonging the range.

To transfer values from the old simulation domain to the new, a form of bilinear filtering for three dimensions is used. Each cell in the new simulation domain is given the average value of the eight closest cells from the old simulation domain.

4.3. Simulation Step Skipping

In [RSG06], the simulation is done at each time step. However, when the flame is not significant to the view, it is not necessary to perform these calculations for every frame. Thus, simulation steps may be skipped while letting the particles used for visualisation travel through the simulation domain fetching old values from the velocity and fire density fields. This may, however, lead to a static looking fire if too many steps are skipped as the simulation in practice is slowed down.

There are other possible variants to this strategy. One is to extend the interval between each simulation step while visualising at constant intervals. This will probably lead to a more flickering fire, but the simulation is not slowed down and therefore the fire will not look static. The computational savings will not be as good as for step skipping as long as the intervals between simulation runs are less than a full step. However, the visual quality of the fire animation might be better. It might, however, be hard to implement a fully working version of this variant for the GPU.

The other variant of this strategy is a more low-level one. In stead of skipping or extending the interval between simulation steps, it is possible to lower the quality of the simulation. The velocity field simulation may be skipped every other simulation step, and the velocity field computed last simulation step may be used to evolve the fire density fields in the current simulation step. This will, for each skipped velocity field simulation, save the computational cost of solving twenty iterations of the Jacobi method, which are used for solving the advection step in the velocity field. Still, the Jacobi method is used to solve the diffusion step in all three density fields. Informal tests we have done show that reducing the diffusion approximation from twenty to four iterations may give satisfying visual results. This strategy will not save as many GPU cycles as the other two variants, but the visual quality should be better.

4.4. Particle Count Adjustment

In [RSG06], the size of the particles and the particle count is constant. When a particle is no longer visible it respawns at the base of the fire. When the fire is small or viewed briefly, a lower particle count is needed as the details of the fire is less important for the visual quality of the fire. There are several possible ways of adjusting the particle count.

As in [OFL01], particles with similar position and speed may be clustered together when computing the motion of the particle and visualised individually. Clustering particles may be a good solution when the motion of the particles are computed individually. The particles in our particle systems fetch their velocity from the velocity field, and their new positions are found from their velocity and current position. This is done in parallel for several particles at once (how many depend on your specific GPU). Another reason for gains in the frame rate when rendering fire on the GPU is that the fire particles are not connected in any way. The particles need not to know anything about other particles and thus only need to read from its own location in memory. We have not implemented this strategy.

Another option is to cluster neighbouring particles into one particle that behaves as one both for simulation and visualisation. As with the other method mentioned above, this would require knowledge about other particles, which would decrease the gain of performing the calculations on the GPU. Therefore this method has not been investigated further either.

We use particle respawning together with the cameras distance from the fire to control the particle count of the particle system. We define a target respawning variable that is set based on the distance between the fire and the camera. The target value is set to all particles in the particle system when the camera is close to the fire. The farther away the camera is from the fire, the lower target value. Particles are respawned until the target value is reached. If the particle count of the particle system is higher than the target value, no particles are respawned.

Halting respawning would create a region of the flame with few or no particles followed by a wall of flame as respawning restarts. To counteract this, spawning can be done according to the ratio of current and desired particle count. For example, if the particle count is twice the target count only every other particle will respawn until the desired count is reached.

The target particle count decreases linearly with the fire's overall LOD value. The target count t is determined by the formula

$$t = basecount * lod, \tag{10}$$

where *basecount* is the base particle count used at the highest detail level and *lod* is the overall LOD value. Particles are added to or removed from the fire by altering the respawn rate of particles instead of adding or removing particles instantly.

4.5. Particle Resizing

When reducing the number of particles in the particle system, the fire may look less dense as the particles are quite

small. Also, strange looking holes may appear in the flame. To counterbalance this, we adjust the particle size according to particle count. If the number of particles is lowered, the size of each particle is increased in scale to compensate. Altered particle size only affects newly spawned particles.

Particle size increases logarithmically as the fire's overall LOD value decreases. The size of particles s is determined by the formula:

$$s = \log d, \tag{11}$$

where d is the distance from the camera.

4.6. Other Considerations

For many physical simulations it is important to approximate the behaviour when not in view. For example, you expect a ball or a heat-seeking missile to have a certain behaviour when not in your view. If a ball is thrown out of your view and bounces on a wall, you expect it to return in the same direction. Also, if a heat-seeking missile is aimed and launched at you, you will not stop running just because you cannot see it any longer. Because of the turbulent behaviour of a fire, you will not have any visual expectations of how the flames have evolved when not looking at them. Thus, visual expectations of the viewer will not cause any problems.

There is a problem connected to illumination, though. The fire illuminates its surroundings and the light intensity is set based on the fire's properties. Something will have to be done to approximate the light intensity of the fire as it may be possible to look at the illuminated surroundings and not the fire. The fire intensity may be stochastically generated based on the maximum, minimum and mean temperature of the fire. In transitions between simulated light intensity and randomly generated, the light intensity will be interpolated between the last random generated and current simulated value for a short time interval. The light intensity will be generated by a proxy simulation when the fire is not in the view.

4.7. The Complete Algorithm

The LOD algorithm shown in Figure 1 is run for each frame. The algorithm first checks if the fire is visible. If not, the visualization is disabled, and a simple stochastic simulation is used to maintain the flickering of the dynamic lights. If the fire is visible, a single LOD value is calculated based on the fires distance from the camera as well as a factor determined by the scale of the scene. The LOD value is used to calculate a new size for the simulation domain and a new particle target count and spawn ratio. The simulation domain resizing is done by using a three dimensional form of bilinear filtering. When a fire particle is set to respawn the spawn ratio determines whether no, one or several particles are spawned.

```
calcLOD()
   if fire object does not
   intersect view frustum
      enableProxySimulation();
      disableParticleVisualization();
      return;

   oldLOD = newLOD;
   newLOD = min(distance(), 1);
   if newLOD == oldLOD
      return;
   newGridSize = maxGridSize * newLOD;
   resizeGrid(newGridSize);
   stepSkipping = 1 / newLOD;

   targetParticleCount =
      maxParticleCount*newLOD;

   particleSpawnRatio =
      targetParticleCount
      /currentParticleCount;

distance()
   dist = abs(cameraPosition
          - firePosition);
   return LODFactor / dist;

resizeGrid(newGridSize)
   for each newcell in new grid
      val = 0;
      find intersection in old grid closest
         to newcell's position

      for each oldcell
      in old grid bordering intersection
         val += oldcell;
      newcell = val / 8;

particleRespawn()
   if (currentParticleCount
   != targetParticleCount)
      counter = counter + particleSpawnRatio;
      while (counter > 1)
         spawn single particle;
         counter = counter - 1;
         currentParticleCount++;
      Adjust particle parameters;
   else
       respawn single particle;
   currentParticleCount--;
```

Figure 1: *Pseudo code for the LOD framework for our physically based fire implementation.*

5. Results and Evaluation

All tests were run on an intel 1.83GHz Core Duo with 1GB RAM and an NVIDIA Geforce 7600 Go with 512 MB VRAM. Three different tests have been implemented. The first one focused on possible LOD methods and ran without a proper fluid simulation. Simulation domain resizing, step skipping, particle resizing, and particle count adjustment was implemented. The performance gains indicated by these results did convince us that we were on the right track.

Table 1 shows the result of the second test, which was a 2D fire rendering running on the CPU using different grid dimensions. As can be seen, the performance increases lin-

early with the size of the grid, with performance roughly quadrupling when the dimensions are halved. The resulting flame is shown i figure 3.

5.1. Results

Grid size	Frame rate
128x128	12.76
64x64	51.25
32x32	202.54
16x16	798.68

Table 1: *Performance results for the second test: two-dimensional flame using different simulation domain sizes.*

Table 2 shows the performance result from the third test. A 3D fire that ran completely on the GPU with different grid dimensions and particle counts was tested, see figure 2 for the visual result. The tests were run with the camera a constant distant from the fire. As expected, the impact from altering the particle count is greater when the grid dimensions are small, as the simulation dominates the calculation complexity when the dimensions are larger. This indicates that altering particle count needs to be combined with altering the simulation dimensions to be effective.

	Grid size		
Particle count	16x24x16	24x36x24	32x48x32
512	43.27	20.87	6.28
1024	37.12	18.19	6.12
2048	29.20	14.19	5.94
4096	20.11	10.07	5.54
8192	12.37	7.86	4.87
16384	7.64	5.78	3.93

Table 2: *Performance results (frames per second) for the third test with a three-dimensional flame using different simulation domain sizes and particle counts.*

6. Summary and Future Work

We have presented an algorithm for combining dynamic LOD with physically based fire rendering on the GPU. The algorithm is based on changing the size of the simulation domain and altering the particle system to increase performance of the fire rendering when the fire is far away or not visible. While the work is still incomplete, the preliminary results presented indicate that the algorithm should give good performance gains without significantly degrading the visual appearance of the fire when it is far away from the camera.

Future work will include implementing the complete framework for execution on the GPU to get conclusive results.We will also investigate additional ways to calculate the relative importance of the fire, for instance determining whether other objects obscure the camera's view of the fire.

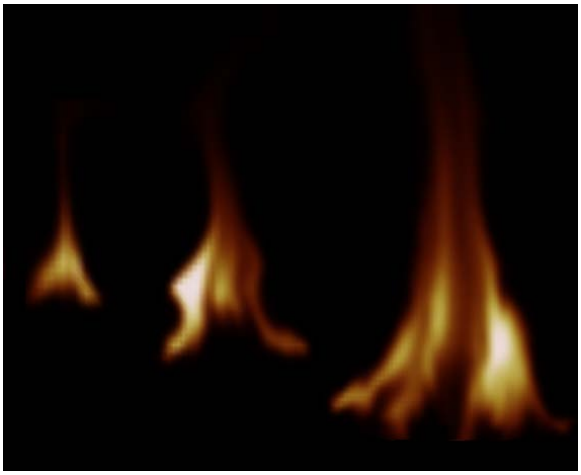

Figure 3: *Screen captures from the second test. The figure shows three different 2D fire renderings with simulation domain dimensons 32x32, 64x64 and 128x128.*

References

[AH05] ADABALA N., HUGHES C. E.: Grid-less controllable fire. *Game Programming Gems 5 (K. Pallister, Ed.), Charles River Media* (2005), 539–549.

[BF06] BALCI M., FOROOSH H.: Real-time 3d fire simulation using a spring-mass model. *Multi-Media Modelling Conference Proceedings, 2006 12th International* (2006), 8pp.

[CCSS05] CALLAHAN S. P., COMBA J. L. D., SHIRLEY P., SILVA C. T.: Interactive rendering of large unstructured grids using dynamic level-of-detail. *Visualization, 2005. VIS 05. IEEE* (October 2005), 199–206.

[DHDS05] DUGUET F., HERNANDEZ C., DRETTAKIS G., SCHMITT F.: Level of detail continuum for huge geometric data. *SIGGRAPH 2005* (2005).

[FC97] FORSYTH D., CHENNEY S.: View-dependent culling of dynamic systems in virtual environments. *Proceedings 1997 Symposium on Interactive 3D Graphics* (1997), 55–58.

[FMF06] F. B.-L., M. L., F R.: Afigraph Õ06: Enhanced illumination of reconstructed dynamic environments using a real-time flame model. In *Proceedings of the 4th international conference on Computer graphics, virtual reality, and interaction in Africa* (Aire-la-Ville, Switzerland, Switzerland, 2006), ACM Press.

[GRS06] GUNDERSEN O. E., RØDAL S., STORLI G.: Physically based simulation and visualization of fire in real-time using the gpu. In *Eurographics UK Chapter*

Figure 2: *Screen captures from the third test. Fire rendering at different distances using 20, 200, and 4000 particles and simulation domain dimensions 16x24x16.*

Proceedings: Theory and Practice of Computer Graphics 2006 (Aire-la-Ville, Switzerland, 2006), Eurographics Association, pp. 13–22.

[HC97] HODGINS J., CARLSON D.: Simulation levels of detail for real-time animation. *Graphics Interface '97* (1997), 1–8.

[HD04] HEOK T. K., DAMAN D.: A review on level of detail. *Computer Graphics, Imaging and Visualization, 2004* (2004), 70–75.

[Hop98] HOPPE H.: Smooth view-dependent level-of-detail control and its application to terrain rendering. *Proceedings of the conference on Visualization '98* (1998), 35–42.

[KCR00] KING S. A., CRAWFIS R. A., REID W.: Fast volume rendering and animation of amorphous phenomena.

[LF02] LAMORLETTE A., FOSTER N.: Structural modeling of flames for a production environment. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), 729–735.

[LRC*03] LUEBKE D., REDDY M., COHEN J. D., VARSHNEY A., WATSON B., HUEBNER R.: *Level of Detail for 3D Graphics*. Morgan Kaufmann Publishers, 2003.

[Mat97] MATKOVIC K.: *Tone Mapping Techniques and Color Image Difference in Global Illumination*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 1997.

[Ngu04] NGUYEN H.: Fire in the "vulcan" demo. In *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Fernando R., (Ed.). Addison-Wesley Professional, 2004, pp. 87–105.

[OCV*02] O'SULLIVAN C., CASSELL J., VILHJÁLMSSON H., DINGLIANA J., DOBBY S., B. MCNAMEE C. PETERS T. G.: Levels of detail for crowds and groups. *Computer Graphics Forum, Volume 21 number 4* (2002), 733–741.

[OFL01] O'BRIEN D., FISHER S., LIN M. C.: Automatic simplification of particle system dynamics. *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings* (2001), 210–257.

[Per85] PERLIN K.: An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1985), ACM Press, pp. 287–296.

[RHSS98] RÖTTGER S., HEIDRICH W., SLUSALLEK P., SEIDEL H.-P.: Real-time generation of continuous levels of detail for height fields. *Proc. 6th Int. Conf. in Central Europe on Computer Graphics and Visualization* (1998), 315–322.

[RSG06] RØDAL S., STORLI G., GUNDERSEN O. E.: Realistic 2d fire in real-time. In *Norsk Informatikkonferanse NIK 2006* (Trondheim, Norway, 2006), Tapir Forlag, pp. 189–200.

[Sta99] STAM J.: Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 121–128.

[TOT*03] TAKESHITA D., OTA S., TAMURA M., FUJIMOTO T., MURAOKA K., CHIBA N.: Particle-based visual simulation of explosive flames. *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on* (2003), 482–486.

[ZWF*03] ZHAO Y., WEI X., FAN Z., KAUFMAN A., QIN H.: Voxels on fire. *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (2003), 36.