Fire rendering with fluid simulation and ray casting

Hanna Björgvinsdóttir* Robin Seibold†

Lund University
Sweden

Abstract

This paper aims to describe a solution of real-time fluid simulation in general, and fire simulation in particular, as well as ray-casting.

1 Introduction

Fluid and fire animations are present in most games nowadays. Graphics cards are getting more and more powerful, and yet the improvement often seems to bypass these animations. Creating fluid and fire animations that look real is challenging, and with realism comes the price of render time.

To take on these challenges, the objective of this project is to render an authentic fire in real time.

2 Algorithms

To achieve the visual effect of a three dimensional fire, simulation is needed, and implicitly a rendering approach to display the simulation result. Below is a description of the algorithms used to achieve this.

2.1 Fire simulation

The algorithm used for simulating the density and movement of the fire is based on Jos Stam's real-time fluid dynamics solver [Stam 2003]. In his article, Jos Stam explains the elements needed to simulate a density and velocity driven 2D fluid. Expanding this stable fluid solver to a three dimensional space is straightforward.

The structure of the fluid is represented by a three dimensional matrix where each cell holds the velocity magnitude and direction, as well as the current density for that cell, as illustrated in figre 1. The three main steps of the fluid solver algorithm are the diffusion, advection and projection steps.

Diffusion is the process of direct neighbouring cells exchanging density, and in the advection step density is moved, depending on the velocity pattern. The last step, projection, forces the velocity to be mass-conserving.

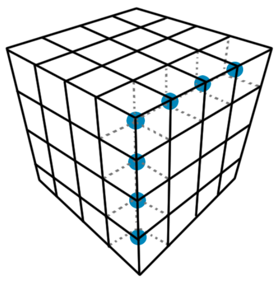

Figure 1: An illustration of the fluid representation

Sources of density and velocity are constantly added to the simulation, which along with diffusion and advection parameters controls the outcome of the simulation.

*dat11hbj@student.lu.se, hanna.bjorgvinsdottir@gmail.com
†dat11rse@student.lu.se, robinbobseibold@gmail.com

In this implementation, the simulation is run on a separate thread on the CPU. In each iteration of the simulation, the density of every cell is encoded into the corresponding voxel of a 3D texture.

The resulting 3D texture is passed to a shader, where ray-casting is performed.

2.2 Ray-casting

In order to visualise the fire simulation, ray-casting is used. The ray-casting is achieved by placing two identical cubes, one back face culled and the other front face culled, where the fire is to be positioned, and render them into separate buffers, using each fragment's normal as color, as described in subsection 30.3.1 in [Crane et al.].

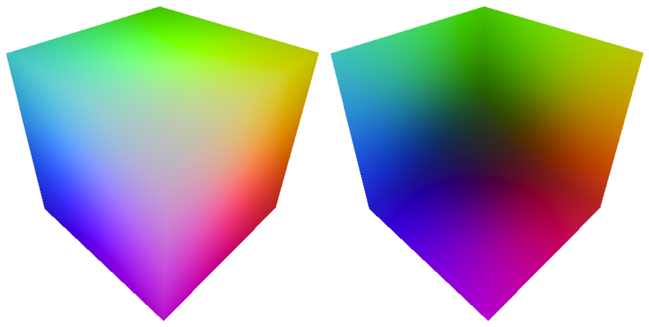

Figure 2: The enclosing cube, back- and front face culled, used to calculate entry, exit, and direction of rays

The colours stored in the buffers are used as coordinates representing the entry- and exit points for the rays, and subtraction between the two values results in the direction of the ray. A step-size approximately the size of one voxel is used to step through the cube along the ray path, from entry point to exit point, gradually accumulating density from the density texture. The final density value is in turn used to fetch an appropriate colour from the texture in figure 3, which is then added to the final output colour.

Adding the colours from all rendering stages in the final render pass gives the effect of transparency, well suited for the fire.

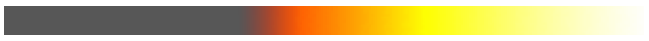

Figure 3: Texture used for setting fire colour

3 Result

The resulting fluid simulator and ray-casting implementation were integrated into a pre-existing scene in RenderChimp. Images of the fire are shown in figure 4 and 5.

The simulation was run on a MacBook Pro, with a 2.4GHz Intel Core i7 processor and a 1024 MB AMD Radeon HD 6770M graphics card. Using a fluid grid of size $32 \times 32 \times 32$ the frame rate reached 60fps, while a size of $64 \times 64 \times 64$ reduced it dramatically.

4 Discussion

During the development of the simulation the result was not rendered, which made it extremely hard to validate the work being done. The inability to test the fire simulation in the early stages turned out to be one of the most challenging aspects of this project.

Another challenge was relating the fire to all of the geometry. Placing the fire in a pot much smaller than the cube used in the ray-casting initially created horrible artifacts. The culling problem was partly solved by placing a cone bottom up in the pot, and rendering only fire inside the cone. This resulted in some sharp edges however, which were removed by placing a square plane on top of the pot, always facing the camera's x- and z-coordinates, and make sure the fire was covered by either the cone or the plane before being rendered.

The conversion from fluid simulation to fire was simplified in this implementation. More advanced methods include temperature, light scattering, and algorithms for deciding the durability of particles[Nguyen et al. 2002]. The simplification consists of deciding colour solely based on the fire's density, and using a cooling constant to make particles further from the flame origin extinguish faster. A thick layer of density was added towards the bottom of the cube, and velocity placed underneath, resulting in an aggressive flame.

When the camera is placed inside the cube used in the ray-casting, no entry-point is registered, and therefore no fire is shown. This is an unpleasant artifact that could not be removed due to time limits.

Instead of using ray casting for volume rendering, another viable option is view-aligned slicing. This was the initial plan for this project, ray-casting was chosen instead, since it felt more comprehensible.

The fact that the performance for a fluid simulation volume of size $64 \times 64 \times 64$ decreased rapidly, invites the possibility for implementing the simulation on the GPU instead of the CPU. This is a viable option since the computations in the simulation are many, but basic.

5 Conclusion

In conclusion, the result is satisfying, however not as much in screenshots as in motion. The project turned out to be more complicated than expected, and there is plenty of room for improvements.

References

CRANE, K., LLAMAS, I., AND TARIQ, S. Gpu gems 3 chapter 30. real-time simulation and rendering of 3d fluids.

NGUYEN, D. Q., FEDKIW, R., AND JENSEN, H. W. 2002. Physically based modeling and animation of fire. Tech. rep., Stanford University.

STAM, J. 2003. Real-time fluid dynamics for games. Tech. rep., Alias — wavefront.

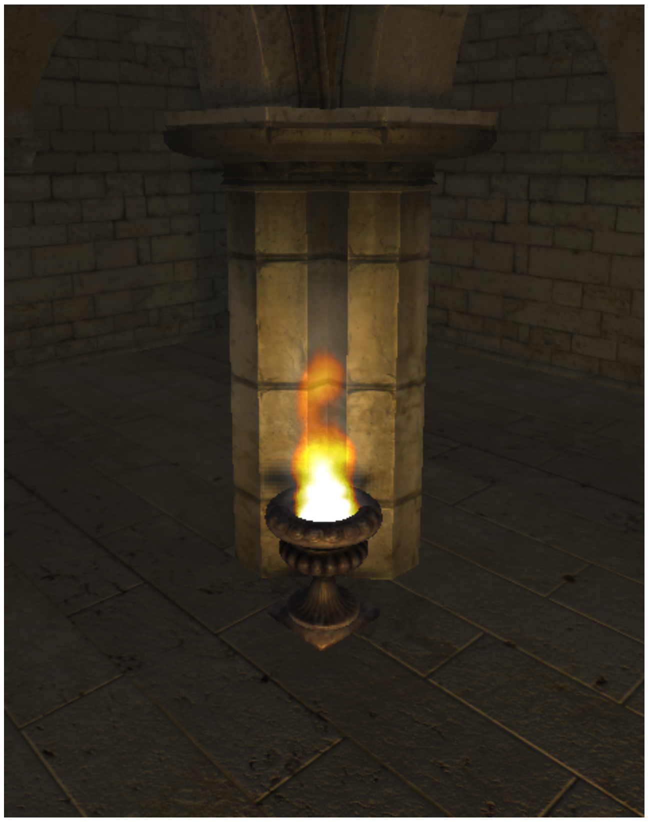

Figure 4: The resulting fire, placed in a pre-existing scene in RenderChimp

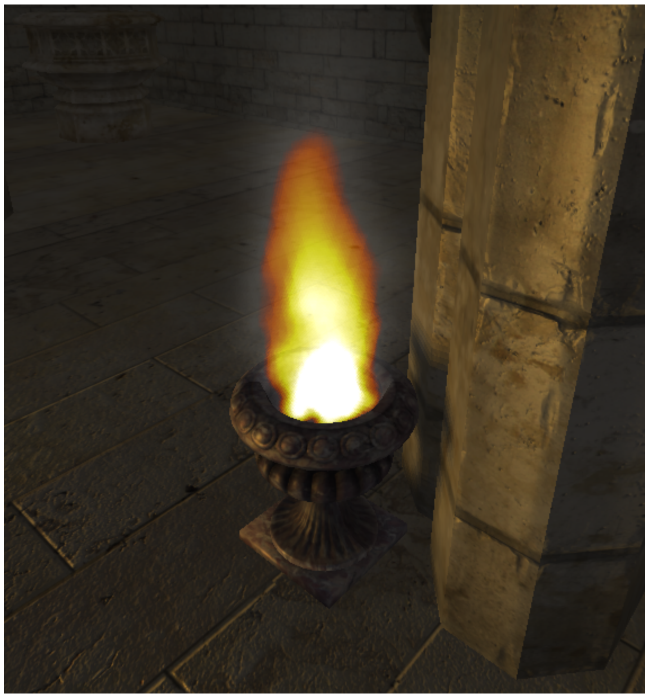

Figure 5: The resulting fire, from another point of view