Real-time 3D Smoke Simulation with Convolutional Neural Network-based Projection Method

Valentin Miu
vmiu@bournemouth.ac.uk
Centre for Digital Entertainment,
Bournemouth University

Oleg Fryazinov
ofryazinov@bournemouth.ac.uk
The National Centre for Computer Animation,
Bournemouth University

1 Introduction

During the last decade, machine learning methods have become very popular for a wide range of applications including those of computer graphics. In this work we consider the problem of real-time fluid simulation, which is a core task for modern physically-based graphics and computer animation. We base our work on an 3D fluid simulation using the FluidNet projection scheme described in [4]. The scheme uses a convolutional neural network to infer the pressure in the projection step, and has been shown to produce better results than a latency-matching Jacobi method. However the method presented in [4] and its 3D implementation was only capable of offline simulations, while we aim for interactive real-time simulation with addition of moving obstacles.

In this work we present the interactive real-time method, which allows for variable inflows and outflows and moving obstacles. It is based on a raycasting Jacobi smoke simulator [3], with modified advection and buoyancy force application, and uses a FluidNet-based convolutional neural network to make the simulation real-time.

2 Method and Results

To train the network for smoke simulation, we use the method shown in [4]. This uses a staggered MAC grid [2], in which the velocity grid samples are at the center of the cell faces, while scalar values such as density are sampled at the center of the cells. One of our main improvements to the original scheme is the addition of a temperature grid in the inference phase, which allows use of the Boussinesq approximation [1] for the bouyancy force update:

$$\mathbf{v}^*_{i,j,k} = \mathbf{v}_{i,j,k} + (\alpha * c - \beta * (T_{i,j,k} - T_{amb})) * \mathbf{g}, \quad (1)$$

Here c is the smoke concentration, and $\mathbf{v}$ and $\mathbf{v}^*$ are the velocity values before and after the buoyancy update, respectively. This is opposed to the constant buoyancy force used in [4] and allows for better simulation of fluid domains with large temperature variations, i.e. hot smoke and fire.

Another improvement is the addition of inflows in the inference stage, in the form of variable density, velocity and temperature grids to be added to the existing domain field values on each update loop. This takes advantage of the robustness of the original training scheme, in which random density variations were added to prevent overtraining and increase generality, enabling it to now handle user-driven perturbations.

The advection scheme is semi-Lagrangian, using a settable limited amount of sub-timesteps in case of backtracing out-of-bounds or into obstacles. This involves halving the timestep until an unoccupied and within-boundary cell is found. This cell becomes the new point to be backtraced from using the remainder of the timestep, the process repeating if this leads again to an invalid cell. For reasonable velocity values (respecting the CFL condition) this is more precise than a single-attempt backtracing and faster than the full line trace method used by the FluidNet implementation in case of failure.

The implementation is written in C++, using OpenGL, CUDA and cuDNN. The OpenGL interoperability of CUDA allows for the advection, inflow and buoyancy force update steps to be handled with higher-performance GPU kernels rather than OpenGL shaders. After these steps, the divergence, previous pressure and obstacle grids (the latter in the form of Boolean flags) are taken as input for the cuDNN neural network. This network performs convolutions at three different grid resolutions (full, half and quarter), and outputs the inferred pressure, whose divergence is subtracted from the velocity to provide a zero-divergence velocity grid. Through interop, OpenGL can access the density and obstacle 3D texture to perform voxel rendering through raycasting, and complete the loop (as in Figures 1 and 2).

As only the smoke density 3D texture is shared with the OpenGL side, the CUDA side could easily be attached to other rendering methods. This also allows all grids except smoke concentration to be stored as buffers, which avoids copying memory from textures to GPU buffers that cuDNN can operate on.

A limited number of wrapper classes around low-level cuDNN operations and custom CUDA kernels were written to allow for simpler higher-level construction of various versions of the FluidNet network, with a variable number of multiresolution branches and convolution operations. By default, 3 resolution branches are used, with a total network depth of 5 convolution operations.

The convolution operations are by far the highest-latency operations. Even without using the most efficient convolution algorithm offered by cuDNN (which requires more memory), the simulator still achieves about 60 frames per second for a 64 by 64 by 64 grid. The simulation was benchmarked on a Nvidia GTX 1070. The results of simple smoke simulation with our method are shown in Figure 1. Obstacles in a form of voxelised volumes can also be added to the simulation, as shown in Figure 2.

Figure 1: Three frames of the smoke simulation with a single user-controlled noisy spherical inflow.

Figure 2: The smoke simulation with additional moving obstacles.

The results show that our method allows for smoke simulation in real-time with interactive input. However, the new advection, buoyancy and inflow schemes require a more suitable training scheme for proper long-term divergence removal. Furthermore, the implementation could be improved with respect to obstacle handling and simulation behaviour control. These can be seen as further work of this research.

Acknowledgments

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

[1] R. Bridson and M. Müller-Fischer. Fluid simulation. In *ACM SIGGRAPH 2007 courses*, pages 1–81. ACM, 2007.

[2] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.

[3] P. Rideout. fluidsim. `https://github.com/prideout/fluidsim`, 2012.

[4] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating Eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3424–3433. PMLR, 2017.