Animating Smoke Using Particle Systems And Fluid Dynamics

Andre Philipp[*] Andy Spencer[†] Haakon Garseg Mørk[‡]
University of California, Los Angeles
CS274C Computer Animation

Abstract

In this project we explored the possibilities of using particle systems and fluid dynamics to animate smoke.

Particle systems are often used in computer animation to animate different types of natural phenomenons like fire, fluids, fog, smoke, and many times a combination that can for instance be an explosion. Particle systems is also use to animate a huge amount of other phenomenons. In these examples given above, the particle system typically have an emitter, which is a source for the particles. The particles itself holds some parameters that is used to simulate the behaviour of each particle (e.g. velocity vector, color/image, lifetime, etc.).

In computer animation fluid dynamics is used to deal with fluid flow. This is necessary to simulate a realistic behaviour of fluids. Fluid dynamics is a subdiscipline of fluid mechanics, and has several sub disciplines itself like hydrodynamics and aerodynamics.

Keywords: partical systems, fluid dyamics, smoke, fire, webgl

1 Software Architecture

WebGL is a JavaScript API for rendering 3D graphics in a web browser, released in 20011. It is supported by all of the most popular web browsers. We choose to use WebGL, because it is very accessible and requires no plugins, or software other that the web browser.

Three.js is a cross-browser JavaScript API that provides a simple interface for 3D graphics in web browsers. It uses WebGL as an underlying API, and is one of the most popular libraries for WebGL programming. Particle System support is included in the original version of the library. We chose to use Three.js because we wanted to focus on the animation part of the project rather than the graphics part, and Three.js provides a level of indirection that suited our needs.

2 Fluid Dynamics Simulation

Our fluid dynamics simulation is based on the work "Real-Time Fluid Dynamics for Games" by Jos Stam. We extended the algorithms to three dimensions and modified the data structures to be more suitable for implementation using JavaScript. In the density function was implemented using an explicit method due to our use of particle systems for smoke simulation.

2.1 Velocity function

Our velocity functions consists of diffuse, project, advect, and project steps. Projection is performed twice, once before and once after the advect step.

During diffuse the velocity field is modified by viscus forces. However, air can generally be considered inviscus (todo: cite), therefore the diffuse step is not needed during a smoke simulation.

During advection, the location of each velocity cell is back propagated in time to determine the new velocities at each point. This step causes the velocity field to lose the mass conceiving property which could lead to a convergence or divergence of the smoke particles and an unrealistic simulation.

As air at relatively slow velocities can be treated as incompressible (todo: cite), the mass conceiving property of the velocity must be maintained. This is done by the project step which modifies to velocity field to remove divergences and convergences.

2.2 Density function

Normally fluid simulations involve computing a density function on rectangular grid. The density at each point represents the number of particles contained within that space. This computation is similar to velocity function computations but does not require a projection step. Furthermore, the diffusion step cannot be left out of the density function.

However, for our simulation, instead of computing a density function we computed the position of a large number of particles directly. This eliminates a large amount of complexity in the algorithm and also simplifies rendering because the particle positions are already known.

In place of the advection step move each smoke particle individually. Because the resulting particles are not arranged on a rectangular gird we can perform a simpler forward propagation rather than the reverse propagation used by advection. Additionally we add random motion to each particle to simulation diffusion of the smoke particles.

2.3 Linear interpolation

Linear three dimensional interpolations were used whenever velocity are calculated for points in the velocity field. These include: the velocity of each particle while moving the particles forward in time, starting location of the velocities during velocity advection, and the position of the cursor when interacting with the user.

2.4 Boundary conditions

Our simulation does not implement boundary conditions on the velocity field, and smoke particles that move outside the simulated area are removed from the simulation.

3 Rendering

In the real world, smoke is a collection of airborne solid and liquid particulates and gases. It would not be feasible to simulate and render a realistic number of particles for a WebGL real time animation like we did. Fortunately, a lot can be done in the rendering stage to improve the final result that ends up on the screen.

[*]e-mail:philipp.andy@gmail.com
[†]e-mail:andy753421@ucla.edu
[‡]e-mail:haakongmork@gmail.com

To deal with the high number of particles in the composition of real smoke, we added a texture to the particles that makes each particle look like a set of particles rather than a single particle, that way we are able to generate fewer particles and still get a rich look.

We found that when the forces acting on particles are not too complex, the assumption that groups of particles will stick together does not result in obvious unrealistic movement of the smoke. The particles opacity is adjusted to be below 100%, and a blending effect is applied to avoid a fragmented look, and make the particles blend smoothly together.

4 User Interaction

Several forms of user interaction were included in our simulation. The user interface provides the ability to start and extinguish fires on the ground, create and stop wind in the air, and clear the smoke created by the fires.

4.1 Fires

Fires can be started by the user anywhere on the ground of the simulation. Each fire consists of the fire location and a heat value. The heat from the fire is added as an external force into the fluid dynamics simulation. Additionally, at each timestep the fire will produce some number of smoke particles. The smoke particles cleared from the simulation at the users request.

todo: describe smoke particle hidingh

4.2 Wind

Wind can be added by the user though mouse interactions. Wind is a temporary change in the velocity field and is propagated by the fluid dynamics equations and interact with the heat and some particles produced by the fires. The velocity field can also be reset to zero at any time.

5 Conclusion

Gas, is one of the most interesting states of matter in the context of Computer Animation. The complexity and diversity of smoke, makes it hard to do accurate physics based simulation.

6 Future Work

The are many ways to improve the smoke simulation, the rendering could be improved with shaders, better textures and more details in the smoke material. Considering improvements on the animation parts only, we would like to improve the following:

6.1 Collision Detection

We would like to add collision detection to the particle system to animate smoke interacting with other objects in the environment. We do not think collision detection between particles is a natural next step for a smoke animation as we see overlapping particles a feature to simulate different densities of smoke.

6.2 Fire/flames

The focus of the project was on smoke and smoke animation, but smoke often occurs in the context of a fire, and it would be an interesting next step to include a fire animation as the source of the smoke.

6.3 Better Equations?

7 References

Smoke	http://en.wikipedia.org/wiki/Smoke
Three.js	https://github.com/mrdoob/three.js/
WebGL	https://www.khronos.org/webgl/

`http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/GDC03.pdf`

`http://www.autodeskresearch.com/pdf/ns.pdf`

`http://nerget.com/fluidSim/`