Physically Simulating Burning of Rigid Objects

Rohan Prinja

November 24, 2014

Contents

Chapter 1

Introduction

1.1 Photorealistic Simulation

Photorealism in the context of computer graphics means creating renderings or animations that look as similar to real life as possible. This project is about using physically-based methods to achieve photorealism in the context of simulating the behaviour of fire and its interaction with arbitrary objects in three dimensions. Simulating fire is an instance of a more general problem - we have materials which are known to follow some physical laws and we want to animate them over some time frame. The hard part is doing this realistically. Physically-based simulation is a way to do so. Before physical simulation was a viable option, photorealism in simulating materials was traditionally achieved in one of two ways:

1. Skilled animators, designers and artists would manually draw or adjust each frame of the animation in order to make it look real. The obvious problem with this approach is that it is slow. A 120-minute with 24 frames per second has 1,72,800 frames! A lot of manpower and time is needed if we want to make an animation longer than just a few minutes. Some amount of automation is necessary.

2. We can model the materials to be animated by simple heuristics whose parameters can be tweaked to give the illusion of realistic behaviour. Flames, for example, can be simulated by Perlin noise, a relatively simple catch-all model (used in many diverse areas) that does not take into account any of the physical properties of fire. Although this in practice very easy to implement, there are two problems with this approach.

   (a) A large amount of tweaking is required to get the heuristics just right, and in any case, we are not really simulating the behaviour of the material as much as simply using "hacks" to get the job done. This limits the extent to which our results can be photorealistic.

(b) Another problem is that in the real world, interactions between objects can become quite complicated. When a white sheet of paper is burnt, fire propagates across the sheet. The paper becomes discolored, going from white to brown and then black. The paper deforms at the point of burning, and this burning is often directly proportional to the strength of the flame at that point. The deformation itself affects the precise way in which the flame will spread from the currently burning portion to the rest of the paper. Little bits of ash may fall off as the burning progresses. Smoke is emitted by the flame. A heuristic model to account for all these complex interactions would be complicated and unwieldy. Our only option then is to sacrifice accuracy for simplicity of implementation.

Given these disadvantages, why is it only recently that physically-based methods have become popular? The answer is that we do not have closed-form solutions for simulating many physical phenomena. Instead, we usually have a system of differential equations that are solved by numerical methods, which are often slow. This means that computing the scene description of even a single frame of an animation on a personal computer can take a large amount of time, perhaps as many as a few seconds to a few minutes or more. Until recently, the hardware to efficiently process these computations had not yet caught up with software demands. Today, however, with the advent of more powerful methods of computation (parallel programming, graphics processing units, render farms, etc.) running time is no longer as big a barrier, and it is easier to achieve realism via physically-based animation.

1.2 About this project

1.2.1 Aim of the project

In this project we aim to develop a framework for simulating the burning of an arbitrarily-shaped object. The inputs to this program will be a 3D object (described as either a mesh of triangles or tetrahedra) with certain parameters, like calorific value, rate of flame propagation, and so on. Along with this, we will have a list of coordinates corresponding to points on the surface of the object which are on fire at time $t = 0$. The output of this program will be a series of frames which can be stitched together into an animation. The animation will depict the propagation of flame across the surface of the object (for thin-shell objects) or the body of the object (for volumetric objects). As the flame progresses, the object will heat up and then burn, in the process deforming and losing mass. The object may burn directly or first melt and then burn (this is parametrizable). Burning byproducts like ash and smoke will be emitted.

1.2.2 Difference from heuristic methods

Like the heuristic methods described in 1.1, we are using parameters to differentiate the behaviour of one type of object from another. However, the difference here is that we are the parameterization corresponds to actual physical properties of the object, for example, does the object melt or directly burn, what is the rate at which fuel escapes from the object surface and so on. In heuristic-based methods there may or may not be any direct relation between a simulation parameter and the physical properties of the materials involved in the simulation.

1.3 Structure of this Report

TODOTODO

Chapter 2

Literature Survey

In this section, we discuss the features we would like to have in a unified model of burning objects, and discuss the literature on the implementation of these features.

2.1 A Wishlist for a Unified Model of Burning

As mentioned above, a feature-complete burning simulation would include most or all of the following phenomena

1. Flames and smoke. The flames spread across the surface of the object as the simulation progresses.
2. Mass loss. The object should lose mass as the simulation progresses, since it is being converted to fuel which is used to feed the flame.
3. The object might deform. By deformation we mean that certain parts of the object change the angles they make with other parts of the object. For example, if we burn the Stanford bunny, we might expect to see its ears droop. Or, as we burn a sheet of paper, it should crumple in on itself.
4. Melting, possibly. An object need not necessarily burn directly, it may first melt and then the liquid will catch fire.

All these phenomena have been well-studied. Below, we discuss some of these papers. However, there hasn't been much investigation into creating a model that unifies all of these features into a single simulation model. All of the following papers concentrate on implementing only a subset of these features, often with great results.

2.2 Survey of previous work

In [1], the authors present a way to simulate smoke. This paper used a standard stable fluids approach as introduced in Stam's seminal paper [2], and introduced vorticity confinement as a way to model the turbulence associated with smoke.

This paper was used in [3], in which the authors presented a framework for animating fire along with smoke. Flames are considered as the product of burning a gaseous fuel. The authors implement a coupled system consisting of two fluids - the fuel undergoing burning, and the hot gaseous product. A level-set is used to track the implicit surface that separates the fuel from the gaseous product. The paper also discusses burning of solid fuels. Points on the surface of the solid are assumed to be emitting gaseous fuel at a certain rate. In this way the solid burning case is reduced to the gas burning case. However, the paper is not concerned with the effects on the solid as the simulation progresses. Mass loss, deformation and residue formation are thus not dealt with, since the focus of the paper is rendering and animating flames.

In [4], the authors implement burning of the surface of volumetric objects. The main contribution of this project is a very fast GPU-based implementation of surface burning. However, flames are not simulated, and only mass loss is dealt with - object deformation is not modeled. Further more, the paper only deals with melting and not burning.

In [5], the authors describe a method to burn thin-shell rigid objects like paper. They depict flames, and also model the object's deformation and discoloration as it burns. However, the scope of the paper does not extend to volumetric objects, or smoke simulation.

[6] is the most feature-complete of them all. It describes a refined meshing procedure for solids and demonstrates melting of solids into liquids and deformation of thin-shell objects. However, it does not demonstrate melting alongside burning or deformation of volumetric objects. Thus there is room for improvement here as well.

[7] discusses a unified model of burning that involves smoke emission, combustion, heat distribution, mass loss, deformation and residue formation. However, it is interactive and real-time, and thus less photorealistic than other burning simulation attempts.

Chapter 3

An Introduction to Fluid Simulation

Flames in a burning model are simulated using fluid simulation techniques. In our implementation we follow the approach outlined by [3], in which fire is treated as a coupled system of two fluids. The first fluid is the fuel being burnt. The second is the gas emitted when the first is burnt. The two fluids are separated by a roughly conical surface called the blue core. The fluid being burnt is inside the cone. The coupling comes from the fact that the blue core has to, at every step of the simulation accurately represent the boundary between the two fluids. To do this a level-set technique is used, and the level-set values are periodically corrected to avoid numerical inaccuracies.

In this chapter, we discuss the general problem of fluid simulation, and then explain how [3] builds on top of fluid simulation to simulate fire.

3.1 The Navier-Stokes equations

There are many ways to simulate fluids. The literature in computer graphics usually focuses on incompressible fluids. The techniques relevant to us are those which simulate the Navier-Stokes equations, sometimes referred to as the Euler equations. They treat the fluid as a velocity field satisfying:

$$\nabla \cdot \mathbf{u} = 0 \tag{3.1}$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\nabla \cdot \mathbf{u})\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{F} \tag{3.2}$$

Here, $\mathbf{u}$ is the fluid velocity, t is time, ρ is the fluid density (assumed uniform everywhere), p is the pressure, ν is the viscosity of the fluid (also assumed

constant), and **F** is the external force field on the fluid (gravity, buoyancy, wind, etc.).

The first equation says that any point inside the fluid, the divergence of the velocity field is zero. This means that at any point inside the volume occupied by the fluid, mass is neither being accumulated nor lost. This is a consequence of the fact that we are dealing only with incompressible fluids.

The second equation is Newton's second law in the context of a (not necessarily incompressible) fluid element, minus the mass term. It says that the change in the velocity field over time is due to four things.

1. The first term is the convection term. The momentum of the fluid is advected ("carried along") by the velocity field.
2. The second term is the pressure term. Pressure differences within the fluid create forces on a fluid element.
3. The third term is the diffusion term. The Laplacian operator measures the extent to which the neighbouring fluid elements differ from the current fluid element. The greater the difference, the greater the force exerted on the current fluid element. The viscosity of the fluid "damps" this force - the larger it is, the less likely the fluid element is to respond to a velocity difference between itself and its neighbours.
4. The fourth term is an external force field. This varies depending on the fluid we are simulating. For water and other liquids, it is typically gravity, which is implemented as a constant downward force field. For the gaseous fuel and the hot gaseous product in the flame simulation, it is buoyancy which varies from fluid element to fluid element depending on the temperature.

There is no closed-form solution for the Navier-Stokes equations yet. So we have to actually update the velocity field by manually calculating the appropriate gradients. We now describe, at a high-level, a way to solve these equations.

3.2 Solving the Navier-Stokes equations

In this section, we describe Stam's stable fluids method, since it is the one used by the paper on which our fire implentation is based. A fluid is continuous, but in practice we simulate it on a discrete grid called the **MAC grid** (marker-and-cell grid). Each cell has a pressure variable p and a density variable ρ defined at its center, and velocity components u_x, u_y and u_z defined at its centre. These values are updated in each simulation step. The main program looks like this:

```
void main()
{
  intializeGrid();
  while (true)
  {
```

```
    updateGrid();
    renderGrid();
  }
}
```

The high-level algorithm for updateGrid() is:

```
void updateGrid()
{
  addExternalForces(); //add gravity, buoyancy, etc. to each MAC cell
  advectParticles(); //advect the velocity field along itself
  diffuse(); //spread the velocity field around to represent velocity
  project(); //remove divergence created above (Navier-Stokes #1)
}
```

The main contribution of Stam's stable fluids paper was that advection was done in a semi-implicit way. Previously, velocity was advected by taking velocities from adjacent grid cells. This led to unstable simulations which were controlled by carefully adjusting the timestep. Stam used the method of characteristics, which is unconditionally stable. To find the new velocity at a point x (located at the center of a grid cell), we walk backwards along the velocity field $\mathbf{u}$ for time δt. Wherever we land, we obtain the new velocity by interpolating the current velocities. This new velocity is the velocity at point x after time δt.

3.3 Flame simulation using stable fluids

3.3.1 Extensions to Stam's paper

In [3], the authors take the ideas presented in Stam's paper and apply it to the problem of simulating fire. In addition to the above-mentioned simulation variables, we also store the unit normal at each grid cell for which the level-set value is non-positive, that is, we store the unit normal at each point in the region occupied by the gaseous fuel as well as the implicit surface.

We also have two balance equations used to conserve mass and momentum across the implicit surface:

$$\rho_h(V_h - D) = \rho_f(Vf - D) \tag{3.3}$$

$$\rho_h(V_h - D)^2 + p_h = \rho_f(Vf - D)^2 + p_f \tag{3.4}$$

where p_h and p_f are the pressures of the hot gaseous products and the gaseous fuels. Similarly, V_h and V_f are the components of the velocities of the gases in the normal direction (normal with respect to the implicit surface), and D is the speed of the implicit surface in the normal direction. We have $D = V_f - S$, where S is a fuel parameter proportional to the rate at which the fuel burns i.e. the rate at which the gaseous fuel is converted to the hot gaseous product.

The implicit surface itself is represented by a level-set defined at the centre of each grid cell, which is updated at each time step. We also store a scalar representing temperature at each grid cell. Every few time steps, numerical error will build up so a fast-march technique (see [8]) is used to correct the level-set values.

3.3.2 The ghost fluid method

The other problem to take care of in a coupled system is that the advection step involves moving a particle 'backwards' in time. For a particle in the gaseous product fluid, tracing backwards might take us to a position in the gaseous fuel region. We would then be interpolating velocities from one fluid to assign a new velocity for the other fluid, which is wrong. To solve this problem, the authors make use of the ghost fluid method introduced in [9]. Say we are tracing back a fuel particle and we end up in the region occupied by the gaseous product. Instead of interpolating the gaseous product velocities we interpolate the ghost fluid velocities, where the ghost fluid velocity is given by

$$\mathbf{u}_h^G = V_h^G \mathbf{n} + \mathbf{u}_f - (\mathbf{u}_f \cdot \mathbf{n})\mathbf{n} \tag{3.5}$$

Here, V_f is the normal velocity of the fuel computed as $V_f = \mathbf{u}_f \cdot \mathbf{n}$, where $\mathbf{n}$ is the unit normal. The velocity V_h^G is computed as

$$V_h^G = V_f + (\frac{\rho_f}{\rho_h} - 1)S \tag{3.6}$$

3.3.3 Evolving temperature and density

For temperature, we evolve a scalar variable Y which takes values in the range $[0, 1]$. This variable is then mapped to the temperature using a temperature curve identified by the authors. The authors also identify a computationally cheap alternative to mapping to the temperature falloff region of the curve (in which the temperature has reached its peak value and now only decreases). For smoke, we simulate another fluid representing smoke. We evolve a scalar variable representing the density of the smoke.

Chapter 4

Modeling Objects

In this section we discuss how we model the objects on which burning is being simulated. We do this by classifying the objects as 2D or 3D, and if 3D, thin-shell or volumetric. In each case, the solid needs to be meshed. Our flame propagation algorithm operates on this mesh.

4.0.4 Meshing two-dimensional objects

Two-dimensional objects are simulated as a mesh of triangles. For this, the `triangle` tool is used. It generates the DeLaunay triangulation of a surface.

4.0.5 Meshing three-dimensional objects

We consider two types of 3D objects - **thin shell objects** and **volumetric objects**. Thin-shell objects have no thickness, for example, paper, cloth and hollow spheres. We only need to mesh the surfaces of such objects using triangles. Volumetric objects by contrast are solid. We model these objects as a mesh of tetrahedra. For this we use the `tetgen` tool. Like `trigen`, it generates a DeLaunay tetrahedralization of a given volume. `tetgen` accepts various parameters which affect the process of tetrahedralization. For example, passing the `-a` option with a numerical argument V when invoking `tetgen` ensures that limit the maximum volume over all tetrahedra in the output is V. Given below is an example of tetrahedralizing a L-shaped volumetric block.

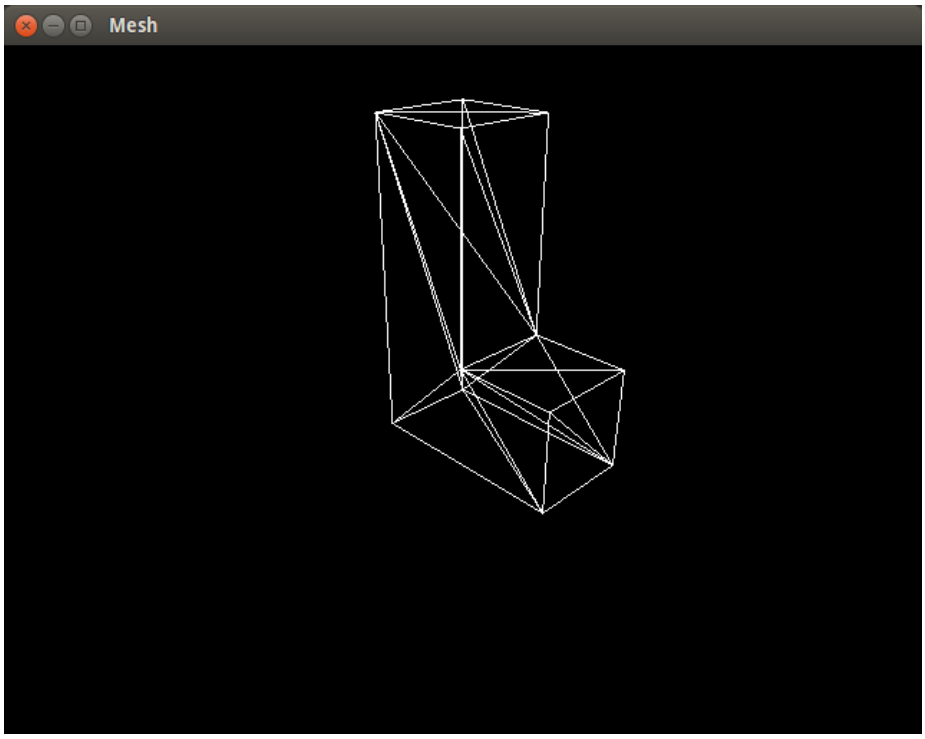


Figure 4.1: Meshing a 3D volumetric L block

4.0.6 Input format

`tetgen` takes as input a file in the `.smesh` format. `smesh` stands for surface mesh. It is a format for representing Piecewise Linear Complexes. Each facet of the input object is considered to be made up of one polygon. We assume no holes or points inside the object. When passed the `-k` flag, it outputs a `.vtk` file. The VTK format is a convenient way to store meshes. To visualize these meshes, we can use the **Paraview** mesh viewer and editor.

Chapter 5

Phenomena associated with burning

In the preceding chapters we described how objects are modeled and how fire and smoke are simulated. Now we look at how objects react to being burnt.

Chapter 6

Acknowledgements

Figure 4.1 is a screenshot of a VTK visualizer I wrote in OpenGL to test `TetMesh`, a class I wrote for representing tetrahedral meshes. The code was originally written by Tom Dalling for a series of blogs explaining OpenGL. I modified the code to incorporate the `TetMesh` class.

For the fluid simulation framework, I originally used **Mantaflow**, a fluid simulation framework. Due to difficulties with the library, I abandoned it in favour of modifying my classmate Harshavardhan's framework for simulating water using stable fluids.

Chapter 7

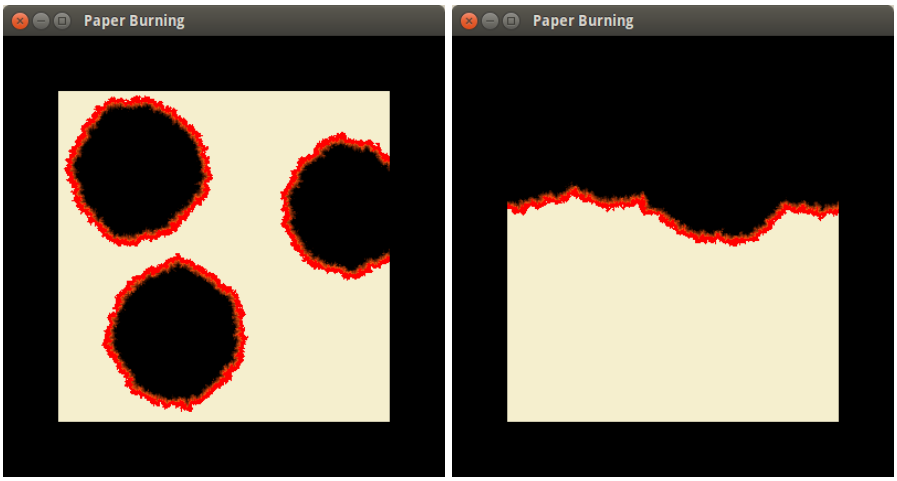
Results and future work

7.1 Heuristic Burning

I implemented a heuristic method for interactively burning a 2D object like paper as a proof-of-concept. In this section we explain the algorithm and evaluate it in the context of this project.

7.1.1 Algorithm

We divide the surface of the object into square cells and probabilistically propagate a flame from a cell to its four neighbours. Initially some points are chosen as sources of burning. Once the `space` key is pressed the flame propagates outwards from each burning source. We have no concept of temperature. Each cell on the object is in one of three states: *unburnt*, *burning* and *burnt*. All cells are initially *unburnt*. In each simulation step, every burning cell has a chance to spread the flame to each of its four neighbours with a certain probability (this is a parameter of the simulation). Once a cell enters the *burning* state, it stays in that state for a fixed number of simulation steps (this is also a parameter), and then enters the *burnt* state. For rendering, we color *burning* cells depending on the amount of time they have remained in the burning state. *burnt* cells are black.



(a) 3 initial points in the interior (b) 5 initial points near the top

Figure 7.1: A heuristic-based method for simulating paper burning

7.1.2 Evaluation

We observe that a lot of fine-tuning is needed to get a good simulation. The probability value needs to be tweaked. If it is too low, the flame dies out too quickly. If it is two high, the flame begins to resemble an outward-growing rectangle, which looks completely unrealistic. Furthermore, deformation, actual flame simulation and smoke are not covered by this simplistic model. Photorealism is absent. This proof-of-concept program verifies the need for physically-based animation as an alternative to naïve heuristic-based methods that only approximate reality instead of describing it.

At the same time, there are some features of this model that we will use in our actual physically-based simulation, namely, propagating flame from nodes to neighbour nodes (except not probabilistically), and transitioning cells from unburnt to burning to the burnt state.

7.2 Tetrahedral Meshing

I implemented a class for representing tetrahedral meshes and integrated it into Mantaflow. To test it out, I wrote a VTK visualizer which loads a `.vtk` file and parses it into a tetrahedral mesh object.

7.3 Flame simulation

Currently, I am implementing a 2D smoke-and-fire simulation which is based on Harsha's code for simulating water in 2D. Once this is complete, I will add

interaction with triangulated 2D meshes to the extent that the flame treats the object just as an obstacle. After this, I plan to incorporate a burning model. Once this is complete, I will extend the program to 3D.

Chapter 8

Bibliography

[1] Fedkiw, Stam, and Jensen. Visual simulation of smoke. pages 17–22, 2001. http://graphics.ucsd.edu/~henrik/papers/smoke/smoke.pdf. 2.2

[2] Jos Stam. Stable Fluids. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999. http://www.autodeskresearch.com/pdf/ns.pdf. 2.2

[3] Nguyen, Fedkiw, and Jensen. Physically based modeling and animation of fire, 2002. http://physbam.stanford.edu/~fedkiw/papers/stanford2002-02.pdf. 2.2, 3, 3.3.1

[4] Jiao, Yonggan, and Aimin. GPU Based Burning Process Simulation, 2012. 2.2

[5] Liu, Liu, An, and Sun. Physically based simulation of thin-shell objects' burning. *The Visual Computer: International Journal of Computer Graphics*, pages 687–696, 2009. http://www.cs.rochester.edu/u/qliu/homepage_data/TVC2009_burning.pdf. 2.2

[6] Losasso, Irving, Guendelman, and Fedkiw. Melting and burning solids into liquids and gases. *IEEE Transactions on Visualization and Computer Graphics*, pages 343–352, 2006. http://physbam.stanford.edu/~fedkiw/papers/stanford2005-08.pdf. 2.2

[7] Zeki Melek. *Interactive Simulation of Fire, Burn and Decomposition*. PhD thesis, 2007. http://research.cs.tamu.edu/keyser/Papers/MelekDissertation.pdf. 2.2

[8] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, pages 1591–1595, 1995. https://www.ljll.math.upmc.fr/frey/ftp/M5105/Sethian%20J.A.,%20A%20fast%20marching%20level%20set%20method%20for%20monotonically%20advancing%20fronts.pdf. 3.3.1

[9] Fedkiw, Aslam, Merriman, and Osher. A Non-Oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (The Ghost Flowsluid Method). *Journal*

of Computational Physics, pages 457–492, 1998. http://public.lanl.gov/aslam/cam98-17.pdf. 3.3.2