

A simulation is obtained by iterating these steps

**Step 0:** Initialize grid

- Set up 3D grid with u, v, w (velocity), p (pressure), ρ (density), T (temperature)

|  |  |
| --- | --- |
| **10–20% of grid size** | Enough to create a visible plume but not overwhelm domain |
| Define a compact initial smoke/temperature region near the bottom center (~10–20% domain width), and let it rise/spread due to buoyancy + advection. | |

|  |  |
| --- | --- |
| **Centered horizontally** | Symmetrical evolution and balanced rising flow |

|  |  |
| --- | --- |
| **Near bottom in z (or y)** | Simulates natural rising smoke due to heat/buoyancy |

|  |  |
| --- | --- |
| **Optionally add noise** | Breaks symmetry, produces natural-looking rollups and curls |

**Real-Time Fluid Dynamics for Games" – Jos Stam (2003)**

* Stam introduced the idea of semi-Lagrangian advection and pressure projection.
* Demonstrates compact source regions (e.g. 10–20% of grid width).

**Stable Fluids" – Jos Stam (1999)**

* Introduces the stable fluid solver on a grid.
* Smoke source regions are always placed near the bottom and take up a small central portion (e.g., 8×8 or 16×16 blocks in a 64×64 grid).

**Smoke Simulation for Large Scale Phenomena" – Rasmussen et al.**

* While using 2D slices, they initialize smoke plumes in localized regions (~1/8 or 1/16 of grid domain).
* Showcases how narrow source regions expand and fill larger space via advection and turbulence

**Step 1:** Add external forces (buoyancy, wind, user input, etc.)

* Influencing smoke’s motion using physical effects like **buoyancy**, **gravity**, and optionally **vorticity confinement** (for swirls).
* In smoke simulation, **hotter and denser regions rise**, simulating fire plumes, steam, etc.

**Equation to implement** f(buoyant force) = α . ρ · Z

**Equation to implement** u += Δt \* f

* Compute the buoyant force
* Add it to the **vertical velocity component**: u(i,j,k) += dt \* f\_buoy

If needed voracity can be included to make smoke swirl

Step 2: Advect quantities using Semi-Lagrangian method

Advection moves **quantities (velocity, density, temperature)** through the fluid

- Advect u, v, w → gives u\*

**Equation to implement:** **u***t* = *-*(**u** *.* ∇)**u** *-* ∇*p*+**f**

- Advect ρ (smoke density)

**Equation to implement:** ρ*t* = *-*(**u** *.* ∇)ρ

- Advect T (temperature)

**Equation to implement:** *Tt* = *-*(**u** *.* ∇)*T*

semi\_lagrangian\_advection(u, u, v, w, dt);

semi\_lagrangian\_advection(v, u, v, w, dt);

semi\_lagrangian\_advection(w, u, v, w, dt);

semi\_lagrangian\_advection(rho, u, v, w, dt);

semi\_lagrangian\_advection(T, u, v, w, dt);

Step 3: Compute divergence of u\*

**Equation to implement** b = ∇ · u\*

Step 4: Solve Poisson equation ∇²p = b

**Equation to implement** Iterative solver (CG)

Step 5: Correct velocity to be divergence-free

**Equation to implement** u = u\* - Δt ∇p

Step 6: Apply boundary conditions

- Walls, inflow, outflow, slip/no-slip

Step 7: Render smoke from ρ (density)

- Use ray marching or OpenGL volume rendering

Step 8: Repeat for each time step

smoke\_sim/

├── include/

│ ├── advection.hpp

│ ├── pressure.hpp

│ ├── boundary.hpp

│ ├── forces.hpp

│ └── render.hpp

├── src/

│ ├── main.cpp

│ ├── advection.cpp

│ ├── pressure.cpp

│ ├── boundary.cpp

│ ├── forces.cpp

│ └── render.cpp

|  |  |  |
| --- | --- | --- |
| **Module** | **File(s)** | **Functions/Responsibility** |
| Advection | advection.hpp | Semi-Lagrangian advection |
| Pressure | pressure.hpp | Pressure Poisson solver |
| Forces | forces.hpp | Buoyancy, vorticity confinement |
| Boundary | boundary.hpp | Apply wall/inlet/outlet conditions |
| Rendering | render.hpp | Render density field (OpenGL or write to file) |