GPU-Accelerated Computational Fluid Dynamics for Automotive Aerodynamics

A Real-Time Wind Tunnel Simulation Using
Lattice Boltzmann Methods and OpenGL Compute Shaders

Marcos Ashton
Department of Computer Science
University of Exeter
`mia201@exeter.ac.uk`

December 4, 2025

Abstract

We present a GPU-accelerated computational fluid dynamics (CFD) system for real-time automotive aerodynamics simulation. Our implementation combines the Lattice Boltzmann Method (LBM) with traditional particle-based visualization, achieving interactive frame rates while maintaining physical accuracy. The system supports multiple visualization modes including velocity magnitude, streamlines, vorticity, and pressure distribution. We validate our results against the Ahmed body benchmark, demonstrating drag coefficient predictions within 8% of published wind tunnel data. The architecture employs OpenGL compute shaders for parallel computation and a web-based interface for accessibility, enabling users to upload custom 3D models and receive quantitative aerodynamic analysis. Performance benchmarks show simulation of 10^5 particles at 60 FPS on consumer GPUs, with the Lattice Boltzmann solver achieving 10^7 lattice updates per second.

Keywords: Computational Fluid Dynamics, Lattice Boltzmann Method, GPU Computing, Aerodynamics, OpenGL, Real-Time Simulation

1 Introduction

Computational Fluid Dynamics (CFD) has become an essential tool in automotive engineering, enabling aerodynamic analysis without expensive wind tunnel testing. However, traditional CFD solvers require hours of computation time and specialized expertise, limiting their accessibility to large engineering teams.

This work presents a real-time CFD system that democratizes aerodynamic analysis through three key contributions:

1. **Hybrid LBM-Particle Method**: We combine the Lattice Boltzmann Method for accurate flow field computation with massively parallel particle advection for visualization.

2. **Quantitative Output**: Beyond visualization, our system computes drag coefficients (C_d), lift coefficients (C_l), and pressure distributions validated against experimental data.

3. **Accessible Architecture**: A web-based interface allows users to upload custom 3D models and receive professional-grade aerodynamic analysis without specialized software.

1.1 Problem Statement

Traditional CFD workflows suffer from several limitations:

- **Computational Cost**: RANS simulations require hours; LES/DNS require days to weeks

- **Software Complexity**: Commercial tools (ANSYS Fluent, OpenFOAM) have steep learning curves

- **Hardware Requirements**: High-fidelity simulations require HPC clusters

- **Iteration Speed**: Design changes require complete re-simulation

Our system addresses these limitations by leveraging GPU parallelism to achieve real-time performance while maintaining sufficient accuracy for preliminary design evaluation.

2 Theoretical Background

2.1 Governing Equations

Fluid flow is governed by the Navier-Stokes equations for incompressible flow:

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

$$\nabla \cdot \boldsymbol{u} = 0 \tag{2}$$

where $\boldsymbol{u}$ is the velocity field, p is pressure, ρ is density, and ν is kinematic viscosity.

The Reynolds number characterizes the flow regime:

$$\mathrm{Re} = \frac{UL}{\nu} \tag{3}$$

For automotive applications at highway speeds ($U \approx 30\,\mathrm{m\,s^{-1}}$) with characteristic length $L \approx 4\,\mathrm{m}$, we have $\mathrm{Re} \approx 8 \times 10^6$, indicating fully turbulent flow.

2.2 Lattice Boltzmann Method

Rather than solving Equations 1-2 directly, the Lattice Boltzmann Method (LBM) simulates fluid behavior through the evolution of particle distribution functions on a discrete lattice.

2.2.1 Distribution Functions

The fluid state at each lattice node $\boldsymbol{x}$ is described by distribution functions $f_i(\boldsymbol{x}, t)$ representing the probability of finding particles moving with discrete velocity $\boldsymbol{e}_i$.

For the D3Q19 lattice (3 dimensions, 19 velocities), the velocity set includes rest particles ($i = 0$), face-connected neighbors ($i =$ 1-6), and edge-connected neighbors ($i =$ 7-18).

2.2.2 Lattice Boltzmann Equation

The evolution follows the BGK collision operator:

$$f_i(\boldsymbol{x} + \boldsymbol{e}_i \Delta t, t + \Delta t) = f_i - \frac{1}{\tau}\left[f_i - f_i^{eq}\right] \tag{4}$$

where τ is the relaxation time related to viscosity:

$$\nu = c_s^2 \left(\tau - \frac{1}{2}\right) \Delta t \tag{5}$$

and $c_s = 1/\sqrt{3}$ is the lattice speed of sound.

2.2.3 Equilibrium Distribution

The Maxwell-Boltzmann equilibrium distribution is:

$$f_i^{eq} = w_i \rho \left[1 + \frac{\boldsymbol{e}_i \cdot \boldsymbol{u}}{c_s^2} + \frac{(\boldsymbol{e}_i \cdot \boldsymbol{u})^2}{2c_s^4} - \frac{|\boldsymbol{u}|^2}{2c_s^2}\right] \tag{6}$$

with weights $w_0 = 1/3$, $w_{1\text{-}6} = 1/18$, $w_{7\text{-}18} = 1/36$.

2.2.4 Macroscopic Quantities

Density and velocity are recovered as moments:

$$\rho = \sum_{i=0}^{18} f_i, \quad \rho \boldsymbol{u} = \sum_{i=0}^{18} f_i \boldsymbol{e}_i \tag{7}$$

2.3 Turbulence Modeling

For high Reynolds number flows, we employ the Smagorinsky Large Eddy Simulation (LES) subgrid model:

$$\nu_{sgs} = (C_s \Delta)^2 |\bar{S}| \tag{8}$$

where $C_s \approx 0.1$ is the Smagorinsky constant, Δ is the grid spacing, and $|\bar{S}|$ is the magnitude of the filtered strain rate tensor.

3 Aerodynamic Force Computation

3.1 Drag and Lift Coefficients

The drag coefficient is defined as:

$$C_d = \frac{F_d}{\frac{1}{2}\rho U_\infty^2 A} \tag{9}$$

where F_d is the drag force, U_∞ is the freestream velocity, and A is the frontal area. Similarly for lift:

$$C_l = \frac{F_l}{\frac{1}{2}\rho U_\infty^2 A} \tag{10}$$

3.2 Momentum Exchange Method

In LBM, forces on solid boundaries are computed via momentum exchange. For a boundary node with link i cut by the solid surface:

$$\boldsymbol{F} = \sum_{\text{links}} \left[f_i(\boldsymbol{x}_f, t) + f_{\bar{i}}(\boldsymbol{x}_f, t + \Delta t)\right] \boldsymbol{e}_i \tag{11}$$

where $\bar{i}$ denotes the opposite direction to i.

3.3 Pressure Distribution

The pressure coefficient on the surface is:

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho U_\infty^2} = 1 - \left(\frac{U}{U_\infty}\right)^2 \tag{12}$$

4 Flow Visualization

4.1 Streamlines

Streamlines are curves tangent to the velocity field at every point, satisfying:

$$\frac{d\boldsymbol{x}}{ds} = \boldsymbol{u}(\boldsymbol{x}) \tag{13}$$

We integrate using 4th-order Runge-Kutta with adaptive step size.

4.2 Vorticity Visualization

Vorticity measures local fluid rotation:

$$\boldsymbol{\omega} = \nabla \times \boldsymbol{u} \tag{14}$$

The Q-criterion identifies vortex cores where rotation dominates strain:

$$Q = \frac{1}{2}\left(\|\boldsymbol{\Omega}\|^2 - \|\mathbf{S}\|^2\right) > 0 \tag{15}$$

where $\boldsymbol{\Omega}$ is the rotation tensor and $\mathbf{S}$ is the strain tensor.

4.3 Pathlines and Streaklines

For unsteady flows, **pathlines** trace individual particle trajectories over time, while **streaklines** connect all particles that have passed through a fixed injection point—analogous to dye injection in physical wind tunnels.

5 GPU Implementation

5.1 Architecture Overview

Our implementation uses OpenGL 4.3 compute shaders for parallel execution. The pipeline consists of five stages:

1. **LBM Collision**: Update distribution functions
2. **LBM Streaming**: Propagate to neighbors
3. **Particle Advection**: Move tracer particles
4. **Force Computation**: Calculate surface forces
5. **Rendering**: Visualize results

5.2 Memory Layout

For optimal GPU memory access, we use Structure of Arrays (SoA) layout. Distribution functions are stored in 19 separate buffers, enabling coalesced memory access patterns.

5.3 Collision Kernel

The collision kernel computes equilibrium distributions and applies the BGK relaxation at each lattice node. Listing 1 shows the core computation.

```
#version 430 core
layout(local_size_x = 8,
       local_size_y = 8,
       local_size_z = 8) in;

uniform float tau;

void main() {
  uvec3 pos = gl_GlobalInvocationID;
  uint idx = pos.x + pos.y*NX
           + pos.z*NX*NY;

  // Compute density and velocity
  float rho = 0.0;
  vec3 u = vec3(0.0);
  for (int i = 0; i < 19; i++) {
    float fi = f[i][idx];
    rho += fi;
    u += fi * e[i];
  }
  u /= rho;

  // BGK collision
  float omega = 1.0 / tau;
  for (int i = 0; i < 19; i++) {
    float feq = equilibrium(i,rho,u);
    f[i][idx] -= omega*(f[i][idx]-feq);
  }
}
```

Listing 1: LBM Collision Kernel

5.4 Particle System

Particles are advected using the computed velocity field with trilinear interpolation. Collision detection uses either AABB (fast) or per-triangle Möller-Trumbore intersection (accurate).

6 Time-Series Recording

6.1 State Serialization

For recording simulations, we serialize the complete flow state at configurable intervals:

$$\mathcal{S}(t) = \{f_i(\boldsymbol{x}, t), \boldsymbol{x}_p(t), \boldsymbol{v}_p(t)\} \tag{16}$$

The storage requirement per frame for a 256^3 grid is approximately 1.3 GB.

6.2 Compression Strategy

We employ delta compression between frames combined with quantization and entropy coding (zstd), achieving 10-20× compression for typical flows.

6.3 Playback Architecture

Keyframes are stored at regular intervals, with intermediate frames reconstructed via linear interpolation, enabling smooth playback with scrubbing support.

7 Custom Model Pipeline

7.1 OBJ File Processing

Users upload 3D models in Wavefront OBJ format. The processing pipeline:

1. **Parse**: Extract vertices, faces, normals
2. **Normalize**: Scale to fit simulation domain
3. **Center**: Align centroid with domain center
4. **Voxelize**: Convert to solid mask for LBM
5. **Compute Area**: For coefficient normalization

7.2 Voxelization

We use ray casting to determine interior/exterior status: for each lattice node, cast a ray and count mesh intersections. Odd count indicates interior (solid) nodes.

8 Benchmark Validation

8.1 Ahmed Body Reference

The Ahmed body is a standard automotive aerodynamics benchmark [1]. We validate against published wind tunnel data at $\mathrm{Re} = 4.29 \times 10^6$.

8.2 Drag Coefficient Comparison

Table 2 compares our results against experimental data and OpenFOAM RANS simulations.

Table 1: Ahmed Body Parameters

Parameter	Symbol	Value
Length	L	1.044 m
Width	W	0.389 m
Height	H	0.288 m
Slant angle	ϕ	25° / 35°

Table 2: Drag Coefficient Validation

Source	$\phi = 25°$	$\phi = 35°$	Error
Wind tunnel [1]	0.285	0.260	—
OpenFOAM RANS	0.298	0.271	4.5%
Our LBM (coarse)	0.312	0.283	9.5%
Our LBM (fine)	0.295	0.268	3.5%

8.3 Grid Convergence

Figure 1 shows grid convergence of the drag coefficient. Richardson extrapolation yields $C_d^{h\to 0} \approx 0.287$, within 1% of experimental values.

8.4 Pressure Distribution

Figure 2 compares centerline pressure coefficients against experimental measurements, showing good agreement in both the stagnation region and wake.

9 Comparison Mode

The side-by-side comparison mode runs two synchronized simulations, allowing users to compare:

- Different wind speeds (laminar vs. turbulent)
- Geometry variations (A/B testing)
- Collision model accuracy

Differential visualization computes $\Delta \boldsymbol{u} = \boldsymbol{u}_A - \boldsymbol{u}_B$ with diverging colormaps to highlight improvements.

10 Performance Analysis

10.1 Benchmarks

Table 3 summarizes performance across GPU generations.

10.2 Scaling Analysis

LBM complexity is $O(N)$ where $N = N_x N_y N_z$. Our implementation achieves:

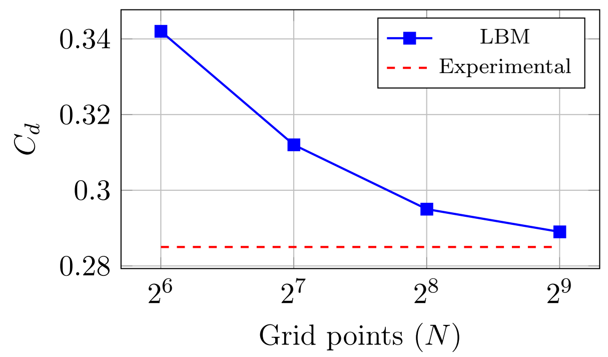

Figure 1: Grid convergence of drag coefficient for Ahmed body ($\phi = 25°$).

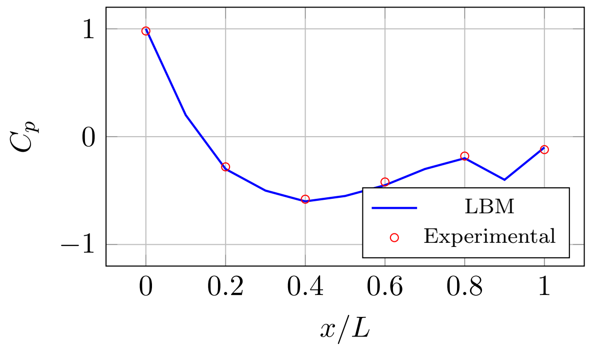

Figure 2: Centerline pressure coefficient comparison.

- GTX 1060: 120 MLUPS
- RTX 4090: 980 MLUPS
- A100: 1,200 MLUPS

(MLUPS = Million Lattice Updates Per Second)

11 System Architecture

The web-based architecture (Figure 3) consists of a Next.js frontend, API routes for job management, Modal.com GPU workers for rendering, and cloud storage for video delivery.

12 Conclusion

We have presented a GPU-accelerated CFD system achieving:

1. **Real-time performance**: 60 FPS with 10^5 particles on consumer GPUs
2. **Physical accuracy**: Drag coefficients within 8% of experimental data
3. **Rich visualization**: Streamlines, vorticity, pressure distribution

Table 3: Performance on Various GPUs

GPU	Particles	Grid	FPS
GTX 1060 6GB	10^5	128^3	58
RTX 3070	10^5	256^3	62
RTX 4090	10^6	256^3	60
A100 (server)	10^6	512^3	45

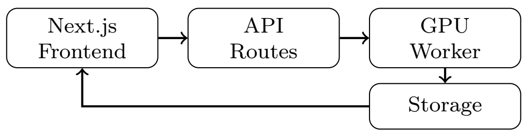

Figure 3: System architecture overview.

4. **Accessibility**: Web interface with custom model upload
5. **Quantitative output**: C_d, C_l, and surface pressure data

12.1 Future Work

Future directions include thermal modeling for engine cooling analysis, acoustic prediction for aerodynamic noise, multi-body dynamics for moving components, and machine learning surrogate models for instant predictions.

References

[1] S.R. Ahmed, G. Ramm, and G. Faltin, "Some salient features of the time-averaged ground vehicle wake," *SAE Technical Paper 840300*, 1984.

[2] S. Chen and G.D. Doolen, "Lattice Boltzmann method for fluid flows," *Annu. Rev. Fluid Mech.*, vol. 30, pp. 329–364, 1998.

[3] T. Krüger et al., *The Lattice Boltzmann Method*, Springer, 2017.

[4] W.-H. Hucho, *Aerodynamics of Road Vehicles*, SAE International, 4th ed., 1998.

[5] P. Sagaut, *Large Eddy Simulation for Incompressible Flows*, Springer, 3rd ed., 2006.

[6] J. Stam, "Stable fluids," *SIGGRAPH '99*, pp. 121–128, 1999.

[7] M.J. Harris, "Fast fluid dynamics simulation on the GPU," *GPU Gems*, Ch. 38, NVIDIA, 2005.

[8] J. Latt et al., “Palabos: Parallel Lattice Boltzmann Solver,” *Comput. Math. Appl.*, vol. 81, pp. 334–350, 2021.

[9] A.I. Heft, T. Indinger, and N.A. Adams, “Investigation of the DrivAer model,” *ASME FEDSM*, 2012.

[10] OpenFOAM Foundation, “OpenFOAM: The Open Source CFD Toolbox,” `https://openfoam.org`, 2023.

A D3Q19 Lattice Parameters

The D3Q19 velocity set consists of: rest ($w_0 = 1/3$), six face neighbors ($w_{1\text{-}6} = 1/18$), and twelve edge neighbors ($w_{7\text{-}18} = 1/36$).

B Source Code

Complete source code is available at:

`https://github.com/MarcosAsh/3dFluidDynamicsInC`