Efficient, Scalable Traffic and Compressible Fluid Simulations Using Hyperbolic Models

Jason Douglas Sewall

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2011

Approved by:

Ming C. Lin, Advisor

Dinesh Manocha, Reader

Jatin Chhugani, Reader

Anselmo Lastra, Reader

Sorin Mitran, Reader

Abstract

Jason Douglas Sewall: Efficient, Scalable Traffic and Compressible Fluid Simulations Using Hyperbolic Models.
(Under the direction of Ming C. Lin.)

This thesis presents novel techniques for efficiently animating compressible fluids and traffic flow to improve virtual worlds. I introduce simulation methods that recreate the motion of coupled gas and elastic bodies, shockwaves in compressible gases, and traffic flows on road networks. These can all be described with mathematical models classified as *hyperbolic* — models with bounded speeds of information propagation. This leads to parallel computational schemes with very local access patterns. I demonstrate how these models can lead to techniques for physically plausible animations that are efficient and scalable on multi-processor architectures.

Animations of gas dynamics, from curling smoke to sonic booms, are visually exciting. Existing computational models of fluids in computer graphics are unsuitable for properly describing compressible gas flows — I present a method based on a truly compressible model of gas to simulate two-way coupling between gases and elastic bodies on simplicial meshes that can handle large-scale simulation domains in a fast and scalable manner.

Computational models of fluids used so far in graphics are inappropriate for describing supersonic gas dynamics because they assume the presence of smooth solutions. I present a technique for the simulation of explosive gas phenomena that addresses the challenges found in animation — namely stability, efficiency, and generality. I also demonstrate how this method is able to achieve near-linear scaling on modern many-core architectures.

Automobile traffic is ubiquitous in modern life; I present a traffic animation technique that uses a hyperbolic continuum model for traffic dynamics and a discrete representation that allows visual depiction and fine control. I demonstrate how this approach outperforms agent-based models for traffic simulation.

Additionally, I couple discrete agent-based vehicle simulation to continuum traffic. My hybrid technique captures the interaction between arbitrarily arranged regions of a road network and dynamically transitions between the two models. I present an analysis

of the impact my hybrid technique on the ability of simulation to mimic real-world vehicle trajectory data.

The methods presented in this dissertation use hyperbolic models for natural and man-made phenomena to open new possibilities for the efficient creation of physically-based animations.

Dedication

This work is dedicated to the Apple IIe computer in my Uncle Fred and Aunt Joanne's den; thanks for all the Robot Odysseys...

Acknowledgments

I would like to thank all of the many people who helped make this thesis possible. Foremost, my advisor, Ming Lin, who has supported my work through all of these years and helped me, sometimes gently and sometimes firmly, sort through the good ideas and the bad. I also owe a great debt to all of my committee members: Jatin Chhugani for his incisive mind and expertise on modern architectures, Anselmo Lastra for always asking the right questions, Dinesh Manocha for pushing me to keep striving for the bigger and better, and Sorin Mitran for making sure that the math checked out (and for helping me find the remainder when it didn't). Without the encouragement — and yes, criticism — from these people, I shudder to think of the state of this thesis.

The word 'I' appears frequently in this thesis, but my numerous collaborators and coauthors have my heartfelt thanks for their help in wrangling the code and pushing the pixels: Paul Mecklenburg, Georgi Tsankov, Nico Galoppo, Paul Merrell, and David Wilkie.

I also owe a great debt to my office-mates over the years: Jeff Schoner, Nikunj Raghuvanshi, Anish Chandak, Liangjun Zhang, and Rahul Narain, for always being there to try to make sense of something I drew on the whiteboard.

Those who I have been fortunate to call friend these past six years are too many to list here, but they are friends indeed for always sharing a laugh and for making sure that I left Sitterson at intervals to recall the outside world that I have striven so hard to recreate within those walls.

To my parents and grandparents, my gratitude knows no bounds; no matter how far afield I have gone from their own experiences, they have encouraged and supported me to do what I love.

Finally, surely the endless hours in the lab and the frantic deadlines would have burned me out of school if it were not for my dear wife Sarah, who has been a SIGGRAPH widow many more times than she deserves.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Computers have a tremendous ability to let us experience virtual worlds, from the sometimes abstract — like today's ubiquitous social networks — to those more grounded in the physical world, such as film productions, games, and tools like Google Earth, Second Life, and Microsoft's Live Maps. Realism plays an important part in these more "tangible" virtual worlds; we would like to see trees sway in the breeze, water to splash and ripple, and clouds to drift overhead. In addition to these familiar natural phenomena, we also wish to see evidence of people — as we walk down the streets of a synthetic city, we expect to be confronted by thronging crowds and the crawl of urban traffic. Virtual worlds that lack these hallmarks of our daily lives are conspicuously uncanny; Figure 1.1 shows a screenshot from Google Earth that contains motionless geometry and images from the real world.

Bringing these features to the computer is not straightforward. There are demanding expectations, as we want the world around us to behave reasonably even if given unreasonable input and domains to operate in, and there are stringent resource requirements, as the computer is already taxed by communication, rendering, and competing simulation tasks. Even if we have a good understanding of the physics of a phenomena, it is a challenge to reproduce a virtual environment that satisfies these opposing requirements of performance and realism.

This dissertation proposes new techniques for the creation of animations of three

Figure 1.1: In Google Earth, Hollywood Boulevard is a 'ghost town'; without animations of natural and man-made phenomena such as traffic, the world seems lifeless

natural phenomena for virtual worlds, specifically compressible gas phenomena — including fluid/solid interactions, shockwaves, and traffic flow on large networks. I explore a common thread that runs through these simulation problems — their ability to be expressed via nonlinear *hyperbolic* models — and demonstrate that this can be exploited to achieve efficient and scalable solutions that are realistic.

1.1 Dynamic systems and differential equations

Many incredibly complex phenomena can be described with remarkably compact mathematical formulas. In particular, *partial differential equations* have impressive descriptive power — these equations (or systems of equations) relate various partial derivatives of quantities; together with boundary conditions and frequently, initial conditions, these describe how a particular phenomena behaves.

Some examples of particularly simple, 'archetypal' partial differential equations (PDEs):

Advection Motion of a wave in q in with speed κ:

$$q_{tt} - \kappa \nabla^2 q = 0 \tag{1.1}$$

Heat Diffusion of a quantity q (with diffusion factor κ):

$$q_t - \kappa \nabla^2 q = 0 \tag{1.2}$$

Laplace Potential field of a quantity q:

$$\nabla^2 q = 0 \tag{1.3}$$

In each of these, q is a scalar quantity and subscripts denote partial differentiation. Equations (1.1) and (1.2) have q_t in them; problems formulated with this type of equation require boundary conditions and initial conditions — these are initial value boundary problems. Equation (1.3) has no time component; problems formulated with these equations need just boundary values — boundary value problems.

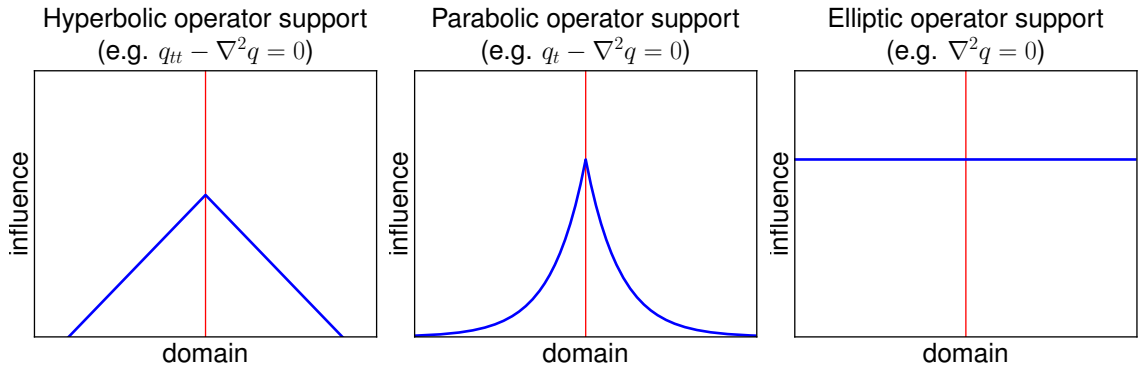
These simple equations often admit analytic solutions for certain boundary conditions, but real-world phenomena are typically described by more complicated equations — Equations (1.1), (1.2), (1.3) combined together in some fashion, often with yet other components. Many of the most important natural phenomena are described by *nonlinear* systems of equations. Initial boundary-value and boundary-value problems for many nonlinear systems of equations admit closed-form solutions for only the simplest configurations. Indeed, a major open problem — one of the million-dollar 'Millennium Prizes' offered by the Clay Mathematics Institute — is simply *to prove or disprove the*

existence of solutions to the Navier-Stokes equations of fluid dynamics in 3-dimensions.

In the absence of closed-form solutions, numerical methods allow us solve complicated PDEs — these too are challenging to handle efficiently and correctly. This thesis focuses on using characteristics of certain types of PDEs to achieve efficient results for visual applications.

1.2 Hyperbolic models

With the ongoing move in computing towards commodity parallel hardware, techniques that can effectively take advantage of parallel hardware are becoming essential. Hyperbolic partial differential equations (PDEs) are characterized by the finite support of their operators; other classes of PDEs have infinite support and thus require, at one level or another, that each element of a spatial discretization communicate with each other element of the discretization at every iteration of the simulation. In contrast, hyperbolic PDEs require only local information to update the solution at each point — essentially, all information in a hyperbolic PDE travels with a finite speed. The spatially small operators naturally associated with these problems minimize communication, and can be leveraged to achieve a high level of parallelism. Figures 1.2a, 1.2b, and 1.2c show



(a) Hyperbolic **(b)** Parabolic **(c)** Elliptic

Figure 1.2: Representative operator support for 3 classes of PDEs

representative operator support for common classes of PDEs — these correspond to the example equations in Equations (1.1), (1.2), (1.3), respectively.

Additionally, many phenomena are naturally expressed with hyperbolic systems; there are situations where familiar equations and numerical schemes have been adopted for ease of implementation and analysis. These can be dramatically improved in quality of results (in addition to scalabiltiy and efficiency) by using the more natural expression of the phenomena as a hyperbolic system.

1.2.1 Hyberbolic models and trends in computer architecture

As mentioned above, hyperbolic models have limited regions of dependency for each point in the domain, which limits the amount of communication necessary when computing their solutions. This is a useful property in any computational context, but it is particularly relevant in the context of modern developments in computation. Computational power has steadily increased; the prediction of 'Moore's Law' — that the number of transistors in CPUs will double every 18 months — has continued to hold. Translating transistors to performance is not straightforward, however, and these benefits have come in a number of different forms;

Increasing clock speeds Throughout the 1980s, 1990s, and the early 2000s, processor manufacturers were able to steadily increase CPU clock speeds from the single-digit megahertz range to the multi-gigahertz one. Voltage and heat limitations proved themselves to be barriers to this process, and the increase in clock speeds stopped around four gigahertz in the mid-2000s. This type of improvement is easily realized by most software; the processor simply executes instructions more quickly.

Architectural improvements Processor manufacturers have been able to deliver higher throughput in many applications through sophisticated techniques for

instruction-level parallelism — notably out-of-order execution, pipelined execution, and multiple arithmetic units. These improvements are also fairly transparent to software, by effectively increasing the number of 'in flight' instructions that can be executed by the processor at once.

Data- and thread-level parallelism The most recent trend in using additional transistors to deliver performance has been in parallelism. Data-level parallelism — in particular, single-instruction, multiple-data units — provide a dramatic performance improvement in certain applications for a modest amount of processor die area. Thread-level parallelism allows the CPU to run multiple processes and/or 'threads' of execution at once; processor manufacturers place multiple 'cores' on a die, each with their own instruction register files, arithmetic units, and instruction counters. These types of performance require specialized programming and software; data-level parallelism usually requires careful layout of data and special instructions to be utilized effectively, and thread-level parallelism requires algorithms to be rewritten and careful attention paid to synchronization and race conditions.

The above discussion of trends in computer performance and architecture does not take storage into account. Any meaningful computation has inputs and outputs, and must read data from memory and write data back out, and the speed with which this data can be accessed — more precisely, the latency and bandwidth — is tremendously important to computation. The overall throughput of high-performance processors can be limited by the performance of the memory system associated with it, and memory performance has not kept pace with the rapid rise in computing power.

To mitigate the effects of processors exceeding the performance of their associated memory, processors operate with a memory hierarchy — a series of caches of decreasing latency and capacity that sit ever closer to the processor. The result is that large problems are typically broken up into smaller 'working sets' to take advantage of these

small areas of high-speed memory. The specifics vary for different models, but in multi-core CPUs, at least one level of the memory hierarchy is partitioned and private to each core.

As the number of cores in a CPU increases, it becomes increasingly difficult to keep these different, partitioned levels of the memory hierarchy coherent and to move data between them. Communication-intensive methods — such as elliptic or parabolic methods that need all-to-all communication to find solutions — are greatly penalized by these limitations, which continue to grow as larger numbers of cores are added to CPUs and architectural changes improve processor performance. Hyperbolic methods, with their limited regions of dependence, enable small working sets based on local regions of the simulation domain that are able to take advantage of CPU performance while minimizing communication.

1.2.2 Conservation laws

The phenomena examined in this thesis are governed by systems that can be written as *conservation laws*; that is, equations of the form:

$$\frac{\partial \mathbf{q}}{\partial t}(\mathbf{x}, t) + \nabla \cdot \mathbf{F}(\mathbf{q}(\mathbf{x}, t), \mathbf{x}, t) = 0 \tag{1.4}$$

Here $\mathbf{x}$ is position in space, t time, $\mathbf{q}$ the vector of unknowns (such as density, momentum, and energy) and $\mathbf{F}$ the vector-valued *flux function* that characterizes the system.

Equation (1.4) has a simple, intuitive interpretation: given any connected subset V of the domain governed by such an equation, the total value of $\mathbf{q}$ in V does not change over time except for what passes through the boundary ∂V; the quantities represented by $\mathbf{q}$ are *conserved*. In fact, Equation (1.4) tells us precisely the *flux* through the boundary ∂V at a time t: $\mathbf{F}(\mathbf{q}(\mathbf{x}, t), \mathbf{x}, t)\ \forall \mathbf{x} \in \partial V$. See Figure 1.3 for a visual depiction of this for a 1-dimensional problem.

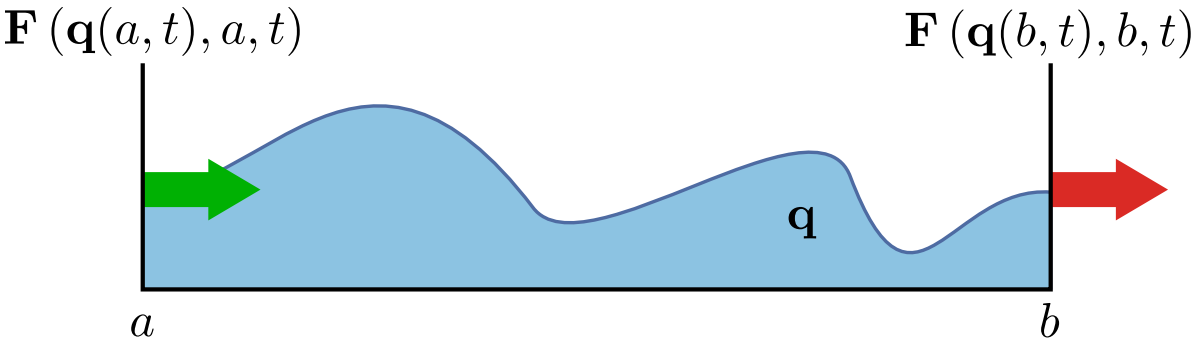


Figure 1.3: A 1-dimensional conservation law and fluxes

Formally, the above statement is substantiated by the divergence theorem; see Leveque [Leveque, 2002] for a lucid discussion of conservation laws.

1.2.3 Hyperbolicity in Conservation Laws

It is the flux function $\mathbf{F}$ in Equation (1.4) that characterizes the system; problems like passive advection and linear acoustics have flux functions that are linear in $\mathbf{q}$, whereas the equations governing compressible fluid flow and traffic dynamics have nonlinear functions $\mathbf{F}$. The flux function also determines the classification of the system; for simplicity, if one assumes a 1D conservation law of the form

$$\frac{\partial \mathbf{q}}{\partial t}(x,t) + \frac{\partial \mathbf{F}}{\partial x}(\mathbf{q}(x,t)) = 0 \tag{1.5}$$

then this can manipulated, without loss of generality, to:

$$\frac{\partial \mathbf{q}}{\partial t}(x,t) + \left[\frac{\partial \mathbf{F}_i}{\partial \mathbf{q}_j}\right](\mathbf{q}(x,t))\frac{\partial \mathbf{q}}{\partial x}(x,t) = 0 \tag{1.6}$$

which is called the *quasi-linearization* of the system. The matrix $\left[\frac{\partial \mathbf{F}_i}{\partial \mathbf{q}_j}\right]$ is the Jacobian $J_{\mathbf{F}}$ of $\mathbf{F}$; the system is called *hyperbolic* if the eigenvectors of the matrix form a basis of the space of $\mathbf{q}$ for all 'physical' values of $\mathbf{q}$. Briefly, said eigenvalues correspond to the speeds of propagation in the system — for these speeds to correspond to finite

propagation, the corresponding eigenvalues must be real (c.f. [Leveque, 2002]).

1.2.4 Hyperbolic problems in animation

In developing this thesis, I have identified a number of problems in graphics and simulation to which hyperbolic equations can be applied to achieve superior results and efficiency:

1. **Fluid and rigid body interaction**: Use of incompressible Navier-Stokes for compressible phenomena: it has been common practice in graphics to use solvers for the familiar incompressible Navier-Stokes equations for fluid simulation of all types, including *compressible* fluids such as air. This allows large simulation time steps but is difficult to parallelize and precludes compressible phenomena. Additionally, coupling an implicit solver with suspended solid objects can be challenging due to the differences in simulation methodology.

2. **Shockwave propagation**: Methods have been developed for high-energy gas and explosion simulations that use naïve numerical techniques like finite difference or require circuitous modifications to the base equations and numerical scheme to function. Such methods suffer from numerical difficulties, a limited range of phenomena, and difficulty to parallelize.

3. **Traffic simulation**: The use of strictly discrete techniques for traffic simulation eschews opportunities to gain efficiency from macro-scale behaviors, while purely continuum methods are unable to account for individual vehicles in a satisfying way.

This thesis addresses these problems using hyperbolic models that achieve a broad range of physically-based phenomena while enhancing their performance and scalability.

1.3 Thesis statement

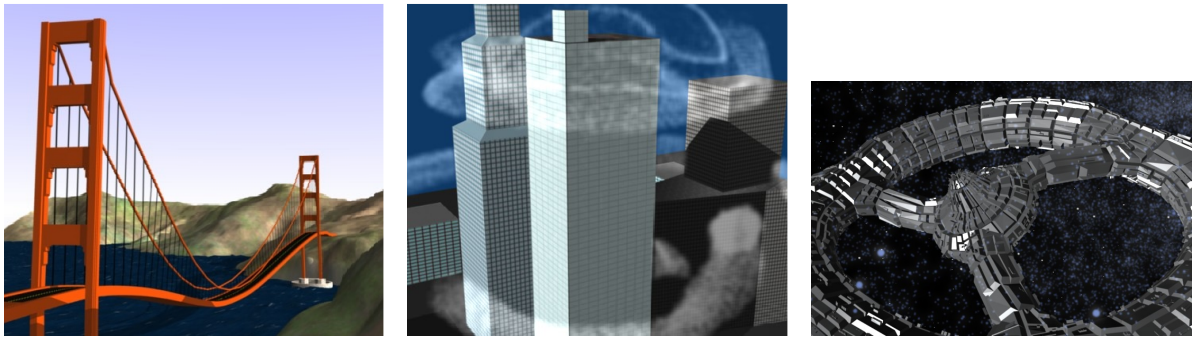
My thesis statement is as follows:

Physically-based animations of compressible flows, including fluids and traffic, may be synthesized efficiently and scalably with numerical methods based on hyperbolic models.

To support this thesis, I present four methods; two dealing with different regimes of compressible gas flow and two approaches for creating animations of large-scale traffic simulations.

1.4 Main results

Fluid/elastic simulation on simplicial meshes



(a) **(b)** **(c)**

Figure 1.4: Compressible fluid/solid body interaction

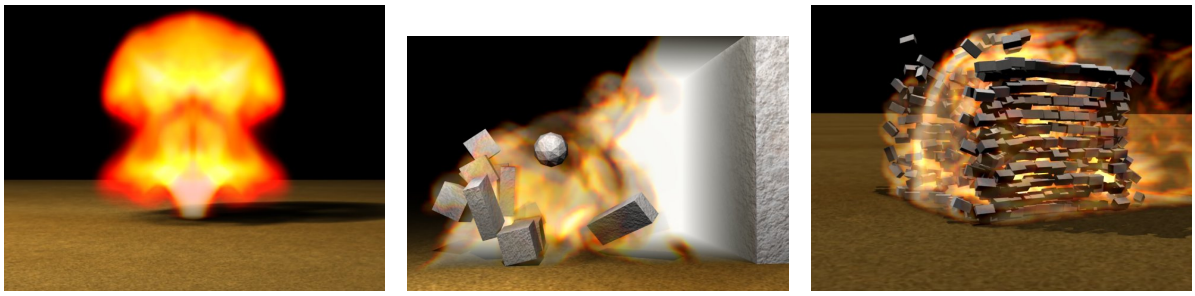
The visually compelling details of fluid motion are often best experienced through their interaction with solid objects; consider the bobbing of a buoy in the swells, the flapping of a flag in the wind, the turbulent wake of a boat, and the curling of smoke out of a chimney. Such phenomena are desirable for visual applications, and yet, in the case of *compressible* fluids, little attention in graphics has been devoted to the efficient handling of their coupled motion with solid bodies. In Chapter 2, I describe a method for the animation of coupled gases and elastic bodies; this technique facilitates the

simulation of compressible fluids and bodies in large domains. I have adapted *residual distribution schemes* [Roe, 1982] to the problems of generating visual compressible fluid flow and present how it may be bidirectionally coupled to other physics modules such as elastic bodies, and I show performance scalability of this technique on multi-core systems. The main results of this are:

- Demonstration of the applicability of Roe's residual distribution schemes to animation of gas flow.
- Animation of coupled compressible gas simulation and deformable-body simulations.
- Demonstration of the scalability of the method on many-core architectures.

These results were published in [Sewall et al., 2007]; see Figure 1.4.

Shockwave-handling gas simulation



(a) **(b)** **(c)**

Figure 1.5: Visual shockwave simulation

High-energy phenomena like explosions, sonic booms, and the effects of blast waves are common special effects in summer blockbusters, and yet little attention has been devoted on the subject of simulating shockwaves for visual applications. These high-speed, dramatic discontinuities are at the heart of the phenomena many visual effects

shots strive to capture, but such effects have usually been realized through artist intervention.

To handle the numerically difficult but visually exciting phenomena associated with trans- and super-sonic fluid flow, I have developed a technique for the animation of fluids featuring physically-based shockwaves, sonic booms, and blast waves. I show how this method addresses the needs of animation problems by demonstrating a wide variety of phenomena, including coupled rigid body motion. Furthermore, I show how the method may be effectively parallelized on modern parallel CPUs and GPUs. This material is presented at length in Chapter 3; the primary results are:

- Demonstration of superior quality over previous methods along with increased efficiency.
- A simple, flexible technique for handling the interaction of shocks and large numbers of rigid bodies.
- Performance scaling through a cache-aware parallelization scheme for many-core architectures.
- Parallelization scheme for modern GPU architectures.

These results were published in [Sewall et al., 2008] and [Sewall et al., 2009]. See Figure 1.5 for example images from this work.

Visual continuum traffic simulation

Automobile traffic is a defining feature of modern life; techniques for simulating traffic flows have the potential to make road networks more efficient and safe by predicting congestion and hazardous conditions. Despite the role traffic plays in our daily lives, little work has been done one efficient methods of traffic simulation for visual applications.

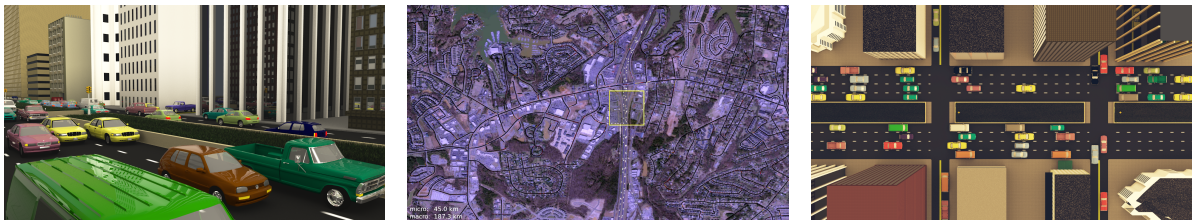
(a) **(b)** **(c)**

Figure 1.6: Continuum traffic simulation adapted for visual applications

To generate animations of traffic flow, I have developed a technique that leverages efficient continuum models of traffic to drive discrete components suitable for interactive visualization. Additionally, these discrete elements (conceptually, 'car particles', which I call *carticles*) are able to affect the underlying simulation in specific ways to produce a wider range of features of vehicle motion. I show how this method scales on parallel architectures and its superior performance to strictly agent-based simulation methods for traffic. This material is found in Chapter 4; the main results therein are:

- A low-overhead method for visual representation of continuum traffic flows.
- Expanded analysis and augmentation of the Aw-Rascle-Zhang system of equations to handle real-world traffic scenarios.
- Demonstration of superiority over agent-based methods for large numbers of vehicles.
- Scalability on multi-core architectures.

These results were published in [Sewall et al., 2010b]; see Figure 1.6 for some visuals from this technique.



(a) (b) (c)

Figure 1.7: A hybrid technique for traffic simulation

Hybrid traffic simulation

While the aforementioned visual continuum traffic simulation is useful for animation large amounts of traffic, there are limitations to the continuum approach of traffic simulation, even when coupled with my 'carticles'; in particular, there is a range of heterogeneous behavior that agent-based traffic simulation techniques allow. In Chapter 5, I present my general technique for coupled continuum and discrete traffic simulations; the road network is partitioned into an arbitrary arrangement of simulation regions, and each region undergoes either continuum or discrete simulation. To account of dynamic flow transitions at interfaces, I introduce *flux capacitors*. To convert regions from one type of simulation to another, I present a novel vehicle instantiation process.

I demonstrate how this coupling framework can be used to drive a hybrid traffic simulation technique that shares the strengths of continuum and discrete techniques, and I show its ability to match real-world traffic flow data. The main results of this section are:

- A hybrid simulation technique capable of dynamically-varying regions of simulation regimes.
- A novel process for instantiating discrete vehicles from continuum data.
- Methods for the conversion of flow between regimes.
- Validation comparison with real-world trajectory data.

See Figure 1.7 for example images from this technique.

1.5 Organization

The rest of the thesis is organized as follows: Chapter 2 presents a method for compressible fluid simulation coupled with elastic bodies. Chapter 3 presents my work on animations of shockwaves, and Chapter 4 describes my work on animating traffic using continuum models of traffic. Chapter 5 discusses my hybrid method for coupling discrete and continuum simulations, and Chapter 6 presents a summary of my contributions and a discussion of future work.

This thesis has several appendices; Appendix A discusses some details of residual distribution schemes, Appendix B presents my expanded analysis of the Aw-Rascle-Zhang model, Appendix C discusses how I extract road networks from publicly-available road data, Appendix D discusses my parallelization scheme for traffic simulation, and Appendix E discusses my smoothed geometric representation for roads.

Chapter 2

Fast Fluid Simulation Using Residual Distribution Schemes

2.1 Introduction

Fluid phenomena play important roles in everyday life — as blowing winds, jet streams, chemical dispersion, granular flows, ocean waves and currents, and so on. Although these phenomena are commonplace, they are fascinating, visually and physically, for the effects they produce. Mathematical models that describe them properly are nonlinear and lead to computational simulation processes that are very complex and challenging to perform efficiently; the intricate interplay of essential processes such as convection, diffusion, turbulence, surface tension, and their interaction with rigid and deformable solids demands careful attention to stability, temporal and spatial scales, and domain representations.

In this chapter, I present an efficient method for physically-based animation of fluids that is also suitable for capturing fluid interaction with elastic solids at large scales of space. The two-way interaction of fluids and elastic bodies is unpredictable and visually interesting. I have developed a simple and efficient method for fluid simulation that also can capture these large-scale fluid phenomena and interaction in complex scenes.

2.1.1 Main results

I have investigated the applicability of *Residual Distribution Schemes* (RDS) for physically-based animation of fluids, that may also interact with solids. These schemes were initially introduced in computational fluid mechanics by [Roe, 1987] as genuinely multidimensional methods for solving PDEs. Residual distribution schemes have not been considered for use in computer graphics, but they exhibit a number of attractive properties:

- They are inherently parallel. The scheme is organized as a loop over computational cells. Each computational cell sends updates to nodal values. The process allows massive parallelization.

- Users can balance accuracy versus cost. In a basic form the schemes are first order in space and time. Higher-order accuracy can be imposed locally or overall by either carrying out multiple iterations over the cells or by higher-order (and more costly) interpolation of physical variables in a single cell.

Furthermore, RDS is capable of describing multi-physics applications. Different physical laws can be defined in each cell; although I have not exploited this property in this chapter, it is an attractive property in when selecting appropriate mathematical formulations.

In this technique, I use an unstructured, adaptive tetrahedral mesh to represent the computational domain and effectively capture boundary conditions. I demonstrate this system on large-scale environments under high-energy forces — a strong wind rocking a bridge, skyscrapers bowing and twisting on a windy day, and a space station deforming in a flow of solar particles.

2.1.2 Organization

The rest of this chapter is organized as follows: Section 2.2 presents a brief review of related work, Section 2.3 presents the theory behind RDS and the model in detail, Section 2.4 describes how our fluid simulation method is coupled with deformable body dynamics, followed by a presentation of results in Section 2.5. I discuss some limitations of our approach and conclude with a discussion of potential future research directions in Section 2.6.

2.2 Previous Work

In this section, I briefly discuss related work in computational fluid dynamics, residual distribution schemes, and modeling of deformable solids.

2.2.1 Computational Fluid Dynamics in Graphics

Realistic animation of fluids has been a topic of considerable interest in computer graphics. Among the first work on visual simulation of fluid dynamics was that of [Foster and Metaxas, 1996], in which finite-difference methods were used to simulate free-surface flows modeled by the full 3D Navier-Stokes system of equations. Stam addressed the standard timestep restrictions due to CFL conditions in his pioneering work "Stable Fluids" [Stam, 1999], introducing semi-Lagrangian advection to the graphics community and applying the *incompressible* Navier-Stokes equations to the problem. This work has made robust animation of realistic fluid phenomena possible and popular. Numerous enhancements, such as particle level set methods for modeling free surfaces, followed in [Foster and Fedkiw, 2001]. I refer the readers to the detailed surveys in [Shi and Yu, 2005, Song et al., 2005, Bridson, 2008].

More recently, there has been an increasing desire to model fluid-structure interaction to achieve still more complex visual effects. [Genevaux et al., 2003] presented impressive

results with a method for fluid interacting with objects represented as particles and [Carlson et al., 2004] described an efficient and elegant method for modeling two-way coupling of fluid and rigid bodies using a finite-difference framework with the Distributed Lagrange Multiplier method. Their approach achieves beautiful results, but is mainly designed to handle the interaction of fluid with rigid bodies. [Guenelman et al., 2005] describe how to handle the interaction of infinitely thin shells with fluids.

Fluid dynamics on irregular grids with finite-volume discretization was introduced around the same time by several authors [Feldman et al., 2005, Elcott et al., 2007, Wendt et al., 2007]. Subsequent work by [Klingner et al., 2006] models fluid interaction with moving boundary conditions by re-meshing the domain at each time step and projecting the field variables from the old mesh to the new. Later work [Chentanez et al., 2006, Chentanez et al., 2007] extends the approach of Klingner et al. to handle the coupled simulation of fluids and elastic bodies with an implicit reformulation of the associated equations, and [Batty et al., 2007] augment the hybrid particle-grid approach of [Zhu and Bridson, 2005] with a variational coupling of rigid body kinetics to the pressure correction step frequently used to simulate incompressible fluids.

Relatively little work in computer graphics has utilized the Euler equations — that is, the compressible, inviscid simplification of the Navier-Stokes system of equations — all of the aforementioned methods from the graphics community are based on an incompressible simplification of the equations.

Work done by [Müller et al., 2003] and [Adams et al., 2007] on particle-based fluid with the Smooth Particle Hydrodynamics (SPH) method strives to represent incompressible flow. The natural tendency of the space between particles to expand and collapse suggests potential future application to compressible phenomena.

2.2.2 Residual Distribution Schemes

Residual distribution schemes (RDS) were first presented in [Roe, 1982] and subsequently developed in [Roe, 1987, Struijs et al., 1991] for the Euler equations. Their applicability to hyperbolic systems of partial differential equations has led to their adoption in the aeronautics community, but RDS have also been adapted to solve other classes of equations, including incompressible Navier-Stokes [van der Weide et al., 1999]. A thorough review of this area was recently presented by [Abgrall, 2006]. Despite their popularity in the aeronautics community, RDS have hitherto not been investigated for computer graphics.

2.2.3 Simulating Elasticity in Graphics

The modeling of deformable bodies have been heavily studied in computer graphics for more than three decades. I refer the readers to the recent survey for more detail [Nealen et al., 2006, Gibson and Mirtich, 1997]. In this chapter, I model deformable objects with linear elasticity using a Galerkin finite-element formulation (c.f. [Hughes, 2000]).

2.3 Residual Distribution Schemes in Flow Simulations

I begin this section with an overview of the method, then describe how the equations of fluid dynamics may be solved with residual distribution schemes.

I solve Euler's equations of compressible, inviscid fluid dynamics with Roe's residual distribution schemes (RDS) [Roe, 1987]. These are simple, narrow stencils applied to an unstructured grid for the solution of hyperbolic systems of partial differential equations. The residual of the governing equation is calculated over each simplicial element and

distributed to each adjacent vertex (which I also refer to as *nodes*) every time step, then the accumulated residual contributed to each vertex is integrated in time and the procedure is repeated. The solution for each simplex in the computational grid is essentially independent for a given time step, affording a highly parallel solution; this proves to be very efficient at solving the expensive Euler equations of gas dynamics.

2.3.1 Overview of fluid solver

The system described by the Euler equations is a simplification of the general Navier-Stokes equations of fluid dynamics where the fluid is assumed to be compressible and inviscid. In terms of conservative variables, the system is as follows:

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) &= 0 \\ \frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \left((\rho \mathbf{u}) \, \mathbf{u} \right) + \nabla p &= 0 \\ \frac{\partial \epsilon}{\partial t} + \nabla \cdot (\mathbf{u} \, (\epsilon + p)) &= 0 \end{aligned} \tag{2.1}$$

Here ρ is density, $\mathbf{u}$ the velocity vector, p pressure, and ϵ the total energy. For simplicity and efficiency, typical visual simulations of gaseous phenomena, such as air and the transport of smoke, further simplify Equation (2.1) to be incompressible, i.e. to assume ρ is constant in space and time and introduce the zero-divergence condition $\nabla \cdot \mathbf{u} = 0$ often used for models of water. While the resulting inviscid, incompressible system does not correctly model many physical phenomena, for low levels of energy, noticeable compression is unlikely to occur and the results are physically plausible.

The introduction of the divergence-free condition has far-reaching ramifications on the character of the system and how it can be efficiently solved. The Euler equations (Equation (2.1)) form a strictly *hyperbolic* system of PDEs, while the imposition of the divergence-free condition adds an *elliptical* character to the system.

Put simply, the unmodified Euler equations model the propagation of perturbations

at finite speeds (the speed of sound, to be specific) — they guarantee that perturbations will be local in a given time interval. Certain perturbations in the system with the zero-divergence condition travel with infinite speed; a change in one part of the spatial domain can instantaneously affect all other parts of the domain.

The Euler equations (Equation (2.1)) are naturally amenable to local solution stencils, while the zero-divergence condition mandates the global pressure projection step used in many contemporary fluid simulation methods. Residual Distribution Schemes (RDS) were developed by [Roe, 1987] to take advantage of the hyperbolic character of the Euler equations; independent stencils are applied at each simplex at each time step. The spatial dependency of the method is minimal and it is well-suited to parallel computation.

Consider a slightly idealized example of driving a car in traffic, where what you do depends entirely on the car ahead of you. If the car ahead slows, you too must slow and if the car ahead of you accelerates, you too must accelerate. There is always a delay involved — you must first observe the change in acceleration of the car in front of you before you adjust your own. The effect of this latency can be great with distance; consider a 'column' of cars stopped at an intersection. When the light turns green, the first car accelerates, then the second car, and so forth. Cars approaching the back of this 'column' must slow down even while the cars at the front are accelerating. This is roughly analogous to a compressible system, where the space between cars is variable. If were to impose a fixed length between all cars (an 'incompressible system'), there can be no delay between changes in velocity (i.e. acceleration) of adjacent cars; their velocity must be uniform and *change uniformly*. Thus, information about changes in velocity must be passed through the system instantaneously.

The implication is that cars in 'compressible' traffic have local behavior — their behavior is governed entirely by the car ahead of them — while the cars in 'incompressible' traffic accelerate based on the state of the entire system. It should be clear that the

former type of system is more amenable to parallelization than the latter.

2.3.2 Residual distribution schemes

RDS apply to *conservation laws* of the form

$$\mathbf{q}_t + \nabla \cdot \mathbf{F}(\mathbf{q}) = 0 \tag{2.2}$$

with unknown vector $\mathbf{q}$ and vector flux function $\mathbf{F}$.

The discretization of Equation (2.2)) used in RDS takes the form

$$\mathbf{q}_i^{n+1} = \mathbf{q}_i^n + \frac{\Delta t}{V_i} \sum_T \beta_T^i \mathbf{\Phi}_T \tag{2.3}$$

for a simplex T, where $\mathbf{q}_i^n$ is the solution vector at node i at time n, Δt the timestep used, V_i the dual volume associated with node i, $\mathbf{\Phi}_T$ the vector of fluctuation (or residual) values of the equations over T, and β_T^i weights specifying how $\mathbf{\Phi}_T$ is distributed to the nodes of T.

The fluctuation $\mathbf{\Phi}_T$ is given by

$$\mathbf{\Phi}_T = -\int_{V_T} \nabla \cdot \mathbf{F}(\mathbf{q})\, dV = -\oint_{S_T} \mathbf{F}(\mathbf{q}) \cdot \mathbf{n}\, dS \tag{2.4}$$

over simplex T's volume V_T and surface S_T. Since $\mathbf{q}$ varies linearly over each simplex T, Equation (2.4) simplifies to

$$\mathbf{\Phi}_T = -V_T \frac{\partial \mathbf{F}}{\partial \mathbf{q}} \cdot \nabla \mathbf{q} = -\sum_{i \in T} \mathbf{K}_i \mathbf{q}_i \tag{2.5}$$

with

$$\mathbf{K}_i = \frac{1}{2D} \frac{\partial \mathbf{F}}{\partial \mathbf{q}} \cdot \mathbf{n}_i = \frac{1}{2D} \mathbf{A} \cdot \mathbf{n}_i \tag{2.6}$$

where D is the spatial dimension of the system, and $\mathbf{n}_i$ is the inward scaled normal of

the face opposite node i in the simplex. The matrix $\mathbf{A}$ represents a problem-specific Jacobian of the *quasilinear* form of the equations; this concept is explained in greater detail in [Abgrall, 2006].

2.3.3 The system N-scheme

The vectors β_i of coefficients describe how the fluctuation $\mathbf{\Phi}_T$ is distributed per simplex; specific variants of RDS have different procedures for computing these coefficients. I use the N-scheme (N for 'narrow'), which is the simplest such scheme that enforces upwinding, preserves linearity, and is monotonic. For a system, it is as follows:

$$\beta_T^i = \frac{\gamma_T^i}{\mathbf{\Phi}_T}, \ \sum_{j \in T} \gamma_T^j = \mathbf{\Phi}_T, \tag{2.7}$$

$$\left(\sum_{j \in T} \mathbf{K}_j^+ \right) \gamma_T^i = \mathbf{K}_i^+ \sum_{j \in T} \left[\mathbf{K}_j^- \left(\mathbf{q}_i^n - \mathbf{q}_j^n \right) \right] \ . \tag{2.8}$$

Here $\mathbf{K}^+$ and $\mathbf{K}^-$ are products of the diagonalization of $\mathbf{K}$

$$\mathbf{K} = \mathbf{R}\Lambda\mathbf{R}^{-1} = \mathbf{R}(\Lambda^+ + \Lambda^-)\mathbf{R}^{-1} = \mathbf{K}^+ + \mathbf{K}^- \tag{2.9}$$

with Λ^+ containing the positive eigenvalues of $\mathbf{K}$ and Λ^- the negative.

2.3.4 Discretization of the Euler equations in 3D

The Euler equations (Equation (2.1)) can be written as a conservation law (Equation (2.2)):

$$\mathbf{q} = \begin{bmatrix} \rho & l & m & n & \varepsilon \end{bmatrix}^T \tag{2.10}$$

$$l = \rho u, \ m = \rho v, \ n = \rho w \tag{2.11}$$

$(u,\, v,\, w)$ are the components of velocity. In 3D, Equation (2.2) has the form $\mathbf{q}_t + \mathbf{f}_x(\mathbf{q}) + \mathbf{g}_y(\mathbf{q}_z) + \mathbf{h}_z(\mathbf{q}) = 0$. The flux functions for the Euler equations $\mathbf{f}(\mathbf{q})$, $\mathbf{g}(\mathbf{q})$, $\mathbf{h}(\mathbf{q})$ are defined as:

$$f = \begin{bmatrix} l \\ p + l^2/\rho \\ lm/\rho \\ l\,n/\rho \\ \dfrac{l(p+\varepsilon)}{\rho} \end{bmatrix} \quad g = \begin{bmatrix} m \\ ml/\rho \\ p + m^2/\rho \\ mn/\rho \\ \dfrac{m(p+\varepsilon)}{\rho} \end{bmatrix} \quad h = \begin{bmatrix} n \\ nl/\rho \\ nm/\rho \\ p + n^2/\rho \\ \dfrac{n(p+\varepsilon)}{\rho} \end{bmatrix} \tag{2.12}$$

2.3.4.1 Similarity transformations

Analytical expressions of the fluid eigenmodes are desirable, but these difficult to obtain from the Euler equations in conservation form (Equation (2.1)). These can be transformed to the primitive variables $\mathbf{Q} = (\rho, u, v, w, p)^T$ of the Euler equations through the differential relationship:

$$\frac{\partial \mathbf{q}}{\partial \mathbf{Q}} = \mathbf{M}, \quad \frac{\partial \mathbf{Q}}{\partial \mathbf{q}} = \mathbf{M}^{-1} \tag{2.13}$$

($\mathbf{M}$ is given in Equation (A.1) in Appendix A) The primitive variable equations are of the form

$$\mathbf{Q}_t + \mathbf{F}_\mathbf{Q} \nabla \mathbf{Q} = 0 \tag{2.14}$$

so I obtain the relationship between Jacobians

$$\mathbf{F}_\mathbf{Q} = \mathbf{M}^{-1} \mathbf{F}_\mathbf{q} \mathbf{M}, \quad \mathbf{F}_\mathbf{q} = \mathbf{M} \mathbf{F}_\mathbf{Q} \mathbf{M}^{-1} \tag{2.15}$$

I assume wave solutions of the form

$$\mathbf{Q} = \mathbf{R} \exp i\,(\mathbf{k} \cdot \mathbf{x} - \lambda t) \tag{2.16}$$

with eigenvectors $\mathbf{k}$, eigenvalues λ, and $\mathbf{R}$ the right eigenvector matrix from Equation (2.9). This leads to the eigenproblem

$$(\mathbf{F_Q k})\,\mathbf{R} = \left[\mathbf{M}^{-1}(\mathbf{F_q k})\mathbf{M}\right]\mathbf{R} = \lambda\mathbf{R} \tag{2.17}$$

2.3.4.2 Primitive variable eigensystem

The structure of solutions is given by the eigensystem for the matrix $\mathbf{K} = \mathbf{F_Q k}$,

$$\mathbf{K} = \begin{bmatrix} \mathbf{V}\cdot\mathbf{k} & k_x\rho & k_y\rho & k_z\rho & 0 \\ 0 & \mathbf{V}\cdot\mathbf{k} & 0 & 0 & k_x/\rho \\ 0 & 0 & \mathbf{V}\cdot\mathbf{k} & 0 & k_y/\rho \\ 0 & 0 & 0 & \mathbf{V}\cdot\mathbf{k} & k_z/\rho \\ 0 & k_x a & k_y a & k_z a & \mathbf{V}\cdot\mathbf{k} \end{bmatrix} \tag{2.18}$$

with $\mathbf{V} = \langle u, v, w\rangle$.

The matrix varies over a computational simplex as a result of the linear variation of the flow variables; to establish a single set of eigenmodes upon which to base an upwinding procedure (as in Section 2.3.3), I must choose an appropriate reference state for the flow variables. It can be shown [Roe, 1987] that the Roe average is the proper choice to ensure discrete conservation, a property crucial to shock capturing. In the following, assume that all flow variables are evaluated at the Roe average (given in Equation (A.7) of Appendix A). The eigenvalues are

$$\lambda_+ = (\mathbf{v}\cdot\mathbf{k} + c) > \lambda = \mathbf{v}\cdot\mathbf{k} > (\mathbf{v}\cdot\mathbf{k} - c) = \lambda_- \tag{2.19}$$

with λ having an algebraic multiplicity of 3, and the right eigenvector matrix is

$$\mathbf{R} = \begin{bmatrix} \rho & 1 & 0 & 0 & \rho \\ k_x c & 0 & -k_y & 0 & -k_x c \\ k_y c & 0 & k_x & -k_z & -k_y c \\ k_z c & 0 & 0 & k_y & -k_z c \\ c^2\rho & 0 & 0 & 0 & c^2\rho \end{bmatrix} \tag{2.20}$$

2.3.4.3 K-matrix decomposition

Let $\mathbf{n}_i$ be the unit normal vector to face i pointing into the simplex. The eigenmodes computed above must be classified as inflowing and outflowing; this is needed for proper upwinding of the scheme. The four potential cases are enumerated in Appendix A.

2.3.5 Solution procedure

Each simplex T has four nodes, and a vector of unknowns for the conservative Euler equations $\mathbf{q}$ is stored at each of these nodes. The fluctuation over a simplex T at a given time step is calculated and split according to the pseudocode given in Algorithm 2.1.

2.3.6 Parallel application of RDS

Residual distribution schemes are straightforward to parallelize; the pseudocode in Algorithm 2.2 describes the procedure for a full solve of the Euler equations in parallel (each **for** statement can be applied in parallel over its iterates). The subroutines in Algorithm 2.2 are defined as follows.

ClearAccumulator(n) Clears the $\mathbf{q}$ accumulator at node n

EulerRDS(T) Computes the fluctuation over simplex T according to Algorithm 2.1; returns $\mathbf{q}$ updates for T adjacent nodes.

Algorithm 2.1 EULER-RDS

EULER-RDS(T)

for each node $i \in$ INCIDENTNODES(T)
  // As in Section 2.3.4.1
  $\mathbf{Q}_i =$ PRIMITIVEVARIABLES($\mathbf{q}_i$)
  // Roe parametrization
  $\mathbf{Z}_i = \sqrt{\rho_i}\,\langle 1, u_i, v_i, w_i, H_i\rangle$
// Compute Roe average (See Appendix A)
$\bar{\mathbf{Z}} = \frac{\sum_i \bar{\mathbf{z}}_i}{4}$
for each node $i \in$ INCIDENTNODES(T)
  // As in Equation (2.19)
  $\Lambda_i =$ EIGENVALUES($\bar{z}$, $\mathbf{n}_i$)
  // As in Section 2.3.4.3
  $(\mathbf{K}_i^+,\ \mathbf{K}_i^-) =$ KMATRIXDECOMP($\Lambda_i\,\bar{z}$, $\mathbf{n}_i$)
  // Compute reference inflow state $\bar{\mathbf{Q}}$
  $\bar{\mathbf{Q}} = (\sum_i \mathbf{K}_i^-)^{-1}(\sum_i \mathbf{K}_i^- \mathbf{Q}_i)$
  $\Phi_i = \mathbf{K}_i^+(\mathbf{Q}_i - \bar{\mathbf{Q}})$
  // $\mathbf{M}$ from Appendix Equation (A.1)
  return $\mathbf{M}\Phi_i$

RDS for the Euler equations

Algorithm 2.2 PARALLEL-EULER-RDS

PARALLEL-EULER-RDS()

for each node $n \in$ *LeafFluidNodes*
  CLEARACCUMULATOR(n)
// implicit barrier
for each simplex $T \in$ *LeafFluidCells*
  NodeUpdates = EULER-RDS(T)
  for each node $n \in$ INCIDENTNODES(T)
    ATOMICINC(n, *NodeUpdates*[n])
// implicit barrier
for each node $n \in$ *LeafFluidNodes*
  TIMEINTEGRATE(n)

Parallel procedure for solving the Euler equations

IncidentNodes(T) as in Section 2.3.5

AtomicInc(n, *NodeUpdate*) Atomically adds *NodeUpdate* to the accumulator at node n

TimeIntegrate(n) Multiplies the quantity in the accumulator of node n by $\Delta t/V_n$ and adds it to n's solution vector

2.4 Fluid-Solid Interaction

One of the complex and interesting fluid phenomena is interaction with deformable solids. To test the suitability of our simulation method in modeling such visual effects, I will next describe a preliminary system that uses my fluid simulation method based on residual distribution schemes (RDS) and couples it together with the commonly used finite element methods (FEM) for modeling deformable bodies.

2.4.1 Overview

This system is composed of several modules working in unison: a fluid solver based on RDS, an elastic solver using a standard FEM formulation, mesh management utilities, and other numerical methods and geometric operations binding these components together. Figure 2.1 provides an overview of the system and the interaction between these components. My method solves Euler's equations of compressible, inviscid fluid dynamics with Roe's residual distribution schemes [Roe, 1987] as described above. The elastic solver is a Galerkin formulation of the equations of linear elasticity commonly used in computer graphics; the solution is implicitly integrated in time for stability and the method yields a sparse, symmetric, positive-definite system of equations efficiently handled by iterative solvers.

This technique uses adaptive mesh refinement to focus computational resources on

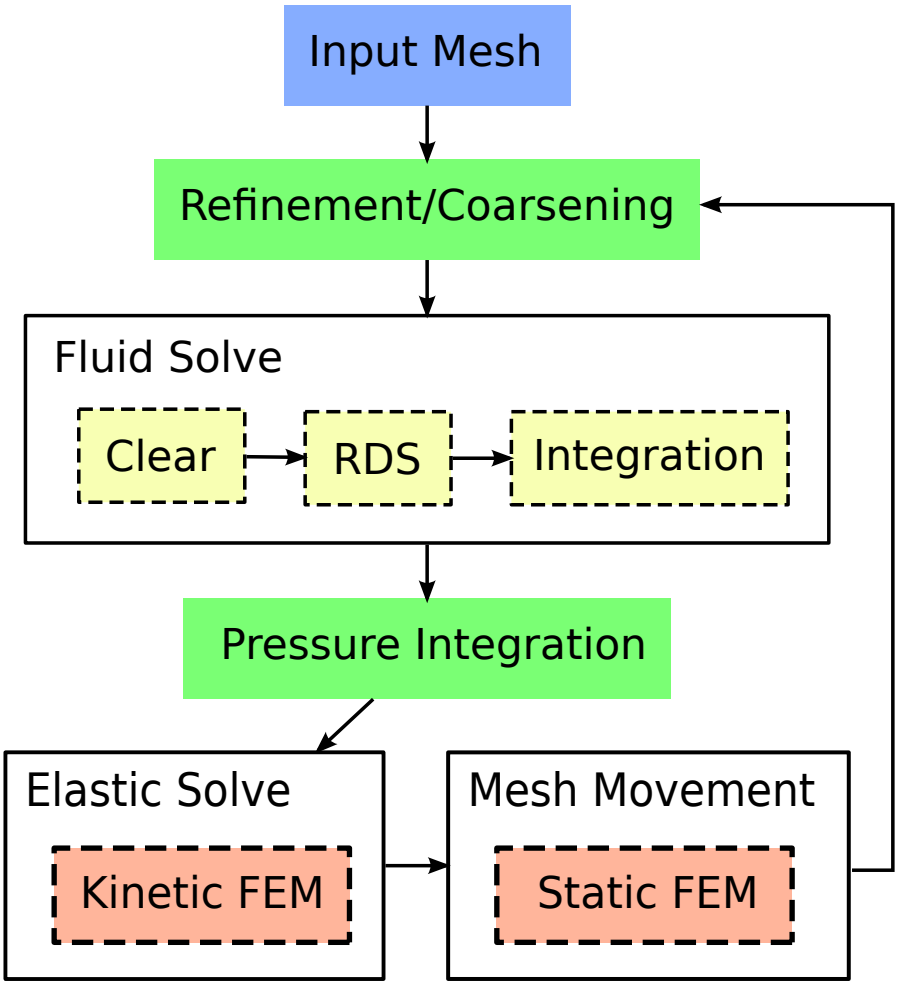


Figure 2.1: The structure of the simulation system

the areas of the simulation most interesting from a standpoint of both visual and dynamic effects. These modules are combined together with a facility for the coupling of the two dynamical systems and an effective method for the updating movement of the computational domain.

2.4.2 Elasticity Simulation

The elastic bodies in this system are modeled with the equations of linear elasticity; I use a Galerkin finite element method (FEM) formulation as described in [Hughes, 2000, Nealen et al., 2006] to build the stiffness matrix $\mathbf{K}$ for elastic bodies at rest state; this is used to construct the following system for the displacement $\Delta\mathbf{x}$ of each node in the

body:

$$\mathbf{M}\Delta\ddot{\mathbf{x}} + \mathbf{K}\Delta\mathbf{x} = \mathbf{F} \tag{2.21}$$

where $\mathbf{M}$ is a diagonal matrix of the mass associated with the dual volume around each node, $\mathbf{K}$ the linear elastic stiffness matrix as above, and $\mathbf{F}$ the forces acting on each node. This method employs a backward Euler temporal discretization as per [Baraff and Witkin, 1998] to obtain:

$$\left(\mathbf{M} + \Delta t^2\mathbf{K}\right)\Delta\mathbf{v} = \Delta t\mathbf{F}_{\text{ext}} - \Delta t\mathbf{K}\Delta\mathbf{x} - \Delta t^2\mathbf{K}\Delta\mathbf{v}^n \tag{2.22}$$

$$\Delta\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta\mathbf{v} \tag{2.23}$$

$$\Delta\mathbf{x}^{n+1} = \Delta\mathbf{x}^n + \Delta t\mathbf{v}^{n+1} \tag{2.24}$$

The left-hand side of Equation (2.22) is a symmetric, positive-definite matrix; its sparsity and block structure is such that each compressed row with can be stored with the associated elastic domain node. I use the conjugate gradient method [Shewchuk, 1994] to solve for the velocity $\mathbf{v}^{n+1}$ and apply Equations (2.23) and (2.24) for the resulting displacement.

2.4.3 Fluid-Structure Coupling

As mentioned previously, I split the solution of the system in time, advancing the fluid in time, then the elasticity. Between these separate solution stages, I propagate the necessary information across domains in the form of boundary conditions. This method is considerably simpler to formulate and solve than an implicitly coupled system as in [Chentanez et al., 2006] and allows the individual solvers to be changed independently.

The force due to pressure on a given surface S is simply:

$$\mathbf{F}_S = \int_S p\mathbf{n}\, dS \tag{2.25}$$

where p is the pressure along the surface, and $\mathbf{n}$ the surface normal oriented toward the interior of the body under pressure. In this simulation, Equation (2.25) is integrated over the dual area on the surface of the fluid-solid interface surrounding each node. For simplicity's sake, I assume that the pressure p is constant over this area; the formula for the force $\mathbf{f}_i$ due to the pressure on node i with dual surface area A_i and incident faces enumerated by j is then:

$$\mathbf{F}_i = A_i \mathbf{n}_i \frac{1}{|j|} \sum_j p_j \tag{2.26}$$

The effect of a solid body's motion on the surrounding fluid is obtained by simply setting the fluid velocity of each node on the fluid-structure boundary to be the velocity of the body at the point.

2.4.4 Adaptive, Semi-Regular Simplicial Meshes

I use unstructured simplicial meshes to represent the fluid and elastic domains and have developed a robust and efficient method for managing the geometric and simulation data used in the numerical methods.

I provide a coarse initial mesh with minimal boundary information and subdivide the mesh as needed to efficiently and accurately represent the solution; the refinement criteria for this process can range from geometric predicates to more sophisticated, domain-specific approaches based on the solution state during simulation. Currently, I only refine cells to enforce volume constraints, but I am investigating schemes based on solution configurations.

2.4.4.1 Splitting Scheme

For a given simplex, this subdivision scheme uses the midpoints of each edge along with the original vertices as the vertices of the child simplices. For triangles, I produce 4 similar triangles of $\frac{1}{4}^{\text{th}}$ the area of the original, and for tetrahedra, I produce 4

similar tetrahedra incident on each of the original vertices, leaving an octahedron with the new edge-midpoint vertices as its vertices. I further divide this octahedron into 4 tetrahedrons; each of these children are $\frac{1}{8}^{\text{th}}$ the volume of their parent tetrahedron.

2.4.4.2 Representation

The adaptive mesh cannot guarantee on the order in which the simplices and vertices (which I call *cells* and *nodes* respectively, to emphasize their role in the solution of a physical system) are created. Thus, for efficiency's sake, I explicitly track each face and edge in addition to cells and nodes to facilitate the quick location of child nodes and incident faces, edges, and cells. An additional advantage of this practice is that it is able to efficiently and directly operate on each of these computational elements without having to locate them by searching incidence information and deal with the inevitable multiplicity of reference.

2.4.5 Mesh movement

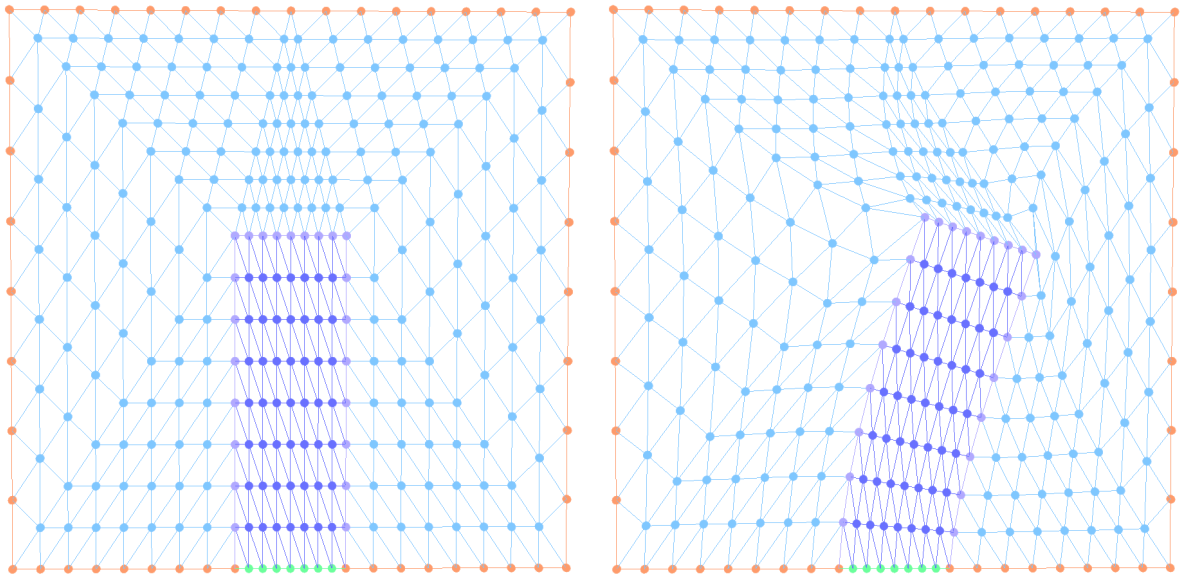
One difficulty in coupling fluid and solid dynamics lies with the way they are typically formulated; the most popular equations describing the behavior of fluids are Eulerian formulations, while elasticity is most naturally described with a Lagrangian formulation. If not carefully handled, difficulty will arise in solving the fluid equations as cells are inverted and become co-located by the changing boundary conditions.

To reconcile these opposing models, the simulation adapts the fluid mesh to capture the moving boundary conditions due to the motion of solid bodies. The motion of the actual fluid-solid interface is completely described by the displacements Δx of the solid body; it remains to determine how to best move the internal fluid nodes to maintain correct meshes. This step is achieved by treating the mesh itself as an elastic body and solving for the node displacements of the internal fluid nodes with a steady formulation

of elasticity. This can be obtained by solving the simple system:

$$\mathbf{K}\Delta\mathbf{x} = 0 \tag{2.27}$$

The displacements of the solid body (see Section 2.4.2) and the domain boundaries provide Dirichlet-type boundary conditions for Equation (2.27)), yielding a system that can be efficiently solved via the method of conjugate gradients. Figure 2.2 demonstrates this mesh movement scheme for a simple 2D configuration. There are situations where this approach does not work well. In particular, topological changes in the fluid domain would require more sophisticated mesh management; one very viable fix for this problem would be to re-mesh the domain whenever deformations become severe enough to invaldidate tetrahedra, as in [Klingner et al., 2006].



(a) An elastic body and enveloping mesh at rest **(b)** An elastic body and enveloping mesh under deformation

Figure 2.2: An elastic body and enveloping mesh's rest and deformed states

2.5 Implementation and Results

My mesh management and simulation code was developed in C++ and the multi-threaded components parallelized with OpenMP (see [OpenMP ARB, 2005]). The tetrahedral meshes were generated using the Tetgen package (see [Si, 2004]). I performed modeling and animation with Blender [Blender Foundation, 2007], and rendering was performed with a combination of Blender and the open-source RenderMan-like renderer, Pixie [Arikan, 2010].

2.5.1 Benchmarks and performance

I have tested and applied this method to a number of challenging problems with applications in computer animation: (a) air current speeding past an iconic bridge, rocking it back and forth, (b) wind buffeting a skyscraper, causing it to bend and twist, and (c) a flow of solar particles passing over a space station suspended high above the Earth. The numbers of tetrahedra listed are for the input meshes given to the solver.

Bridge The first benchmark scene is shown in Figure 2.3. The bridge and the fluid domain are composed of $31,478$ tetrhedral elements.

Skyscrapers In the skyscrapers benchmark as shown in Figure 2.4, the buildings and the fluid domain are composed of $9,088$ tetrahedral elements.

Space Station For the space station benchmark in Figure 2.5, the space station and the fluid domain are composed of 25,129 tetrahedral elements.

Table 2.1 shows the runtime performance achieved by my prototype implementation on the three benchmarks. The timings were collected on a Pentium D 3.4GHz processor with 2 GB of RAM. The fluid simulation using RDS runs in real time. The dominating computational cost in our simulator is due to FEM simulation of deformable solids.

Figure 2.3: Simulated flows rocking a suspension bridge.

2.5.2 Scaling

To demonstrate the scalability of RDS, I have implemented this algorithm (as described in Algorithm 2.2) with the parallization facilities provided by OpenMP. This model of parallel computing is well-suited to the multi-core, shared-memory architectures commonly available on desktop workstations and laptops. It will also be directly applicable to many-core architectures. This method achieves respectable scaling for up to 8 processors on the skyscraper model (see Table 2.2).

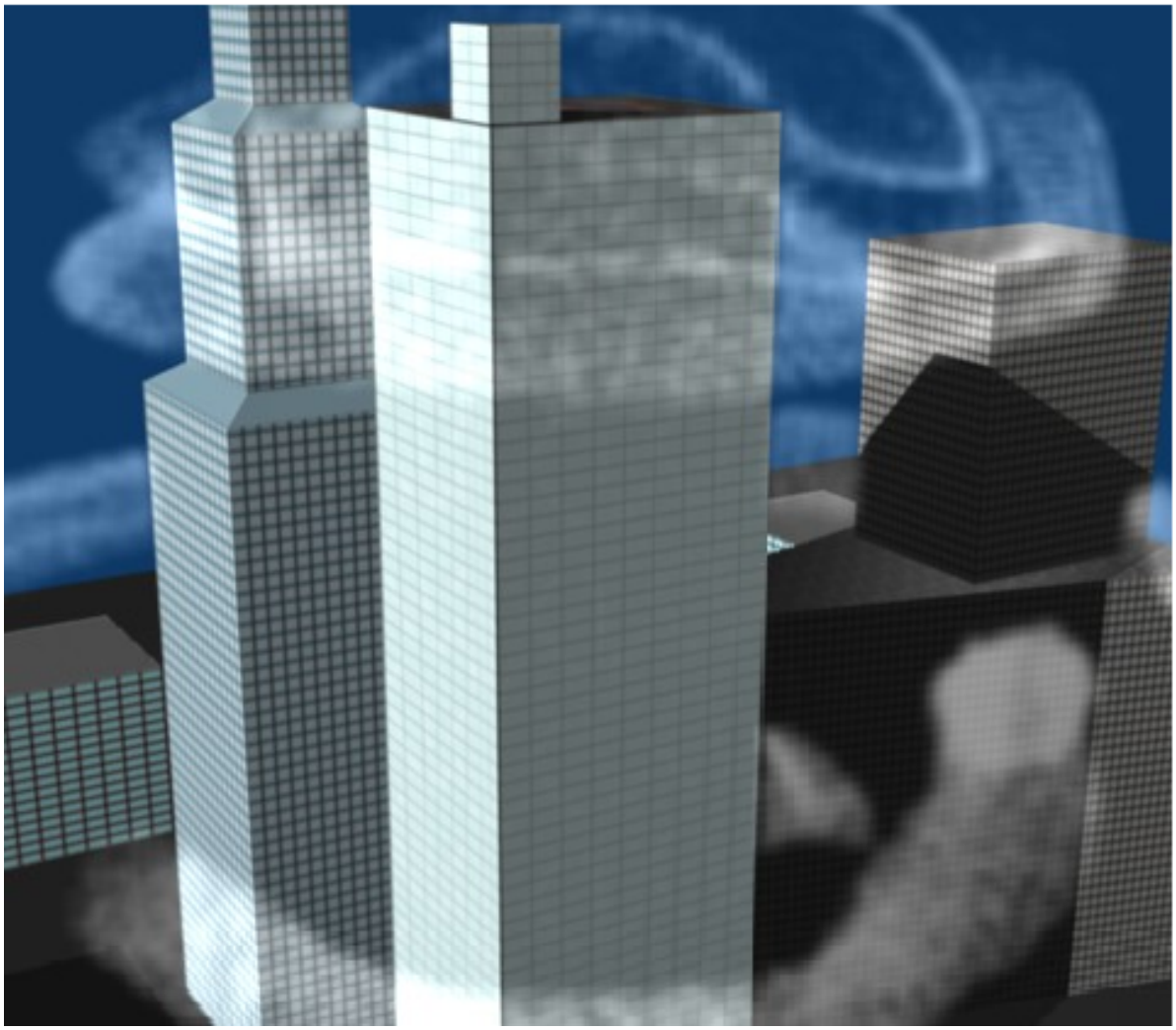


Figure 2.4: Skyscrapers in a whirlwind (9,088 tetrahedra)

2.6 Summary and Conclusion

I have introduced residual distribution schemes (RDS) to computer graphics as method for efficiently simulating high-energy fluids on modern architectures. I demonstrate that RDS are computationally attractive in several regards; they can effectively model multi-physics phenomena, such as two-way coupling between fluids and solids. It offers a natural balance between efficiency and accuracy. The method also takes advantage of adaptive mesh refinement to focus computational efforts on areas of visual and physical significance. Therefore, it is able to deform the computational domain and avoid

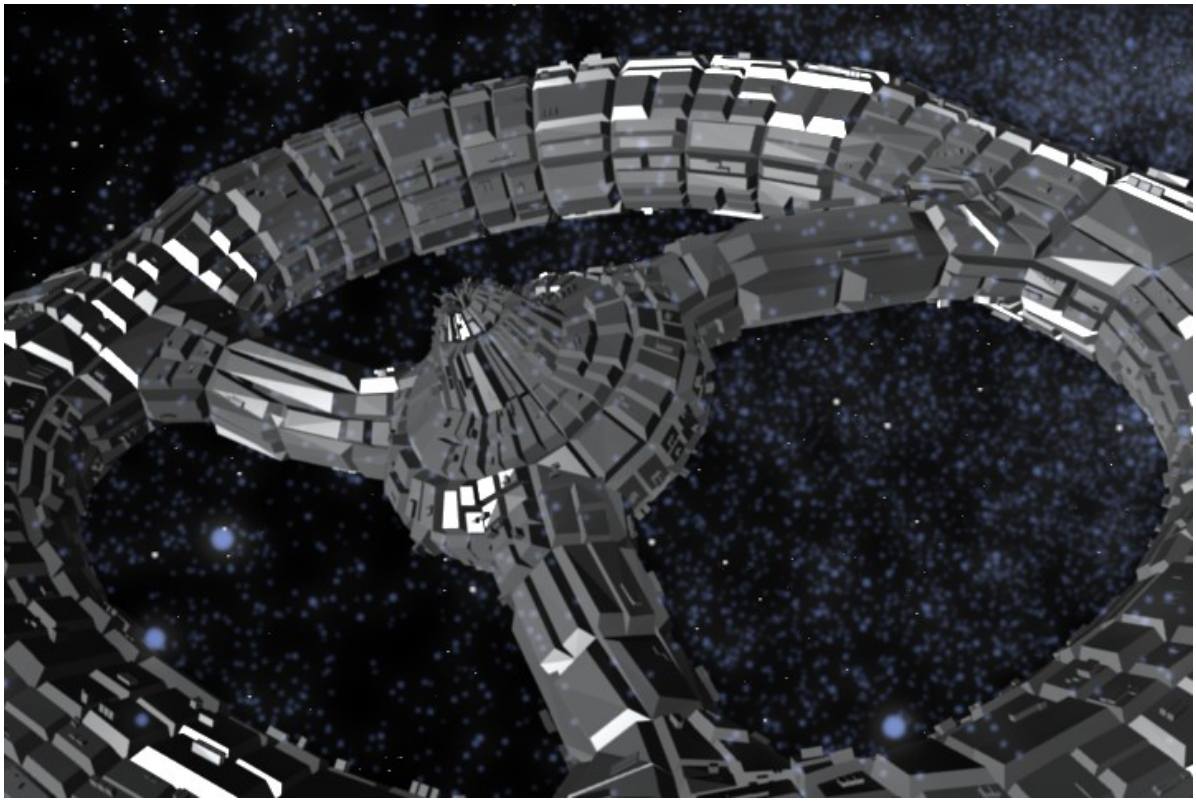


Figure 2.5: Twisting Space Station (25, 129 tetrahedra)

inaccuracies due to inverted computational cells.

2.6.1 Limitations

This method has some limitations. The mesh adaptation scheme assumes limited solid movement and would require re-meshing to handle arbitrary object motion (in particular, topological changes to the computational volume). The N-type residual distribution scheme I use is linear and thus suffers from diffusion; this could be addressed with a non-linear scheme such as Low-Diffusion Advection (LDA) and Positive-Streamwise Invariant (PSI) schemes (see [Abgrall, 2006]).

Scene	# cells	secs/frame		
		Fluid	Solid	Total
Bridge	31k	0.6	5.73	6.36
Skyscrapers	9k	0.15	4.77	4.92
Space station	25k	0.46	14.53	14.99

Table 2.1: Runtime performance for each benchmark

Processors	Frames/second
1	11.22
2	21.25
4	32.37
8	47.39

Table 2.2: Performance scaling of residual distribution schemes for the Euler equations over the Skyscraper scene on an SGI Altix cluster.

2.6.2 Future work

There are a number of directions for future work. Mesh deformation and adaptation methods could benefit from some refinement, and I would like to investigate multiple splitting schemes and more advanced criteria for performing the splits. I would also like to investigate alternatives to remeshing gross deformations in the computational mesh.

There are many options for future work with residual distribution schemes; I would like to combine them with a Poisson solver to allow for the simulation of incompressible fluid, and I would like to investigate higher-order and less diffusive distribution schemes, such as the LDA and PSI methods described in the aeronautics community. I am currently investigating the potential application of RDS to the equations of elasticity for a more integrated approach to fluid-solid interaction, and would be interesting to investigate different time integration schemes for RDS as well.

Due to the inherent parallelism of RDS, I also plan to implement parts of the algorithm on GPUs and future many-core architectures, as well as other new commodity parallel architectures, to exploit RDS' computational properties and further improve its

overall performance. I hope to achieve at one to two orders of performance gain, making this method interactive on desktop workstations or mobile platforms.

In the next chapter, I present my technique for the simulation of compressible gases with shockwaves.

Chapter 3

Visual Simulation of Shockwaves

3.1 Introduction

Recent developments in simulating natural phenomena have made it possible to incorporate stunning, realistic animations of complex natural scenes filled with flowing, bubbling, and burning fluids. Computer-animated and live-action films alike have made great use of these advances in modeling to recreate familiar and interesting effects. Notably, little investigation has been made on how to properly capture shocks and propagate discontinuities in visual simulation. These remarkable phenomena give rise to dramatic events such as explosions, turbulent flows, and sonic booms. Such effects are common in films and are notoriously difficult to handle with numerical methods. Additionally, many state-of-the-art simulation techniques do not fully take advantage of the kind of new, powerful hardware that is emerging; these algorithms are often not designed to handle large domains efficiently and many that are based on specially simplified formulations often are not applicable to phenomena occurring at large spatial scales.

This chapter discusses a method for efficient simulations of nonlinear, compressible gas dynamics and describes how it may be best utilized to generate visually interesting, plausible animations. Many natural phenomena are nonlinear but can often be reasonably approximated through linearization; one example is the linear formulation

of elasticity that is commonly used in graphics-targeted simulation. The equations of fluid motion are not generally suitable for linearization — waves crashing on the beach, curling smoke, and surging shockwaves all arise from the nonlinear characteristics of the system. To solve these highly nonlinear equations in a reasonable amount of time, numerical methods typically discretize simplifications of the true equations that still capture the nonlinearity of the system.

Furthermore, shocks that arise in problems of gas dynamics themselves present a numerical challenge; a shock is a region of rapid spatial variation in a small interval that propagates with tremendous speed — the blast wave that emanates from an explosion or the bow shock that forms around a supersonic projectile are some examples of these phenomena. These have a striking effect on the fluid motion but are very difficult to simulate properly with traditional numerical methods; the scale of motion one would like to capture (namely the space the shock traverses) is at odds with the need to represent the shock itself. Many numerical techniques behave poorly or fail completely in the presence of discontinuous solutions — to simulate shocks with such methods, the resolution of the discretization must be high enough for the shock to appear as a smooth transition, and thus can be prohibitively expensive to compute.

Physically correct methods for shockwave modeling focus less on conventional metrics of accuracy (such as order of convergence) and emphasize the ability to propagate discontinuities stably and with minimal diffusion. Specifically, techniques based on the finite volume method (FVM) have been developed that handle discontinuities well and allow for relatively coarse grids to capture shock behavior. The method presented here, through judicious simplification and application, adapts and improves the efficiency of a class of FVM techniques designed to capture shocks on coarse grids efficiently.

I have demonstrated the application of this method for generating animations of complex fluid motion, including chambered explosions, nuclear detonations, and the turbulence and bow shock around a supersonic projectile (see Figures 3.7, 3.8, 3.9).

The method is also able to describe the interaction of coupled fluids and objects; I demonstrate shockwaves knocking over stacked objects and blowing a brick house to pieces, as well as the effects of an explosion within a tower of heavy blocks. My method is able to considerably reduce the computational complexity of these highly complex effects to the level comparable to existing fluid animation techniques in graphical simulation.

Furthermore, while a naïve parallelization of our method achieves only mediocre scalability, it is possible to achieve much better scaling with a more carefully constructed parallelization scheme. I have explored the components of such a near-optimal scheme and its application to the shared-memory, multi-core architectures that are becoming commonplace. The essential locality of the numerical schemes used in this method allows parallel performance far greater than that typical of methods for fluid simulation in graphics.

3.2 Previous Work

I use the finite volume method to describe compressible fluid dynamics; this follows the based Reconstruct-Evolve-Average paradigm established by [Godunov, 1959]. This has received much attention from the aeronautics community, and I use numerical Riemann solvers based on the work of [Roe, 1981], [van Leer, 1977], and others. For a superb introduction to the topic of the finite volume method and Riemann solvers, see [Leveque, 2002].

The problem of describing the evolution of shocks — known as "shock capturing" — has been addressed from a variety of directions. My work follows the vein of *Riemann-solver based* approaches that strive to treat areas with and without shocks with the same numerical technique. Another family of approaches, generally known as *front-tracking* methods, uses standard solvers in areas away from shocks and explicitly models shocks as evolving surfaces in the domain. Front-tracking approaches have been successful, but

are extremely complicated for two- and three-dimensional simulations and have difficulty handling situations where multiple shocks interact. The survey of [Fedkiw et al., 2003] gives a good overview of the topic.

I have summarized much of the related work in graphics in Section 2.2; as mentioned, relatively little work in computer graphics has utilized the Euler equations, choosing instead to model their effects either through more general equations, greatly simplified models, or models of similar complexity that have been modified to achieve certain effects. [Yngve et al., 2000] use the full Navier-Stokes equations in their method that animates explosions and their secondary effects. [Feldman et al., 2003] simulate combustive phenomena based on an incompressible model of flow with additional density tracers, and [Selle et al., 2005] present an approach that generates what they describe as "rolling explosions". Like Feldman et al., they use an incompressible model of fluid, which precludes the presence of shocks. My approach aims to model phenomena similar to those addressed by [Yngve et al., 2000]. The greater fidelity and higher efficiency afforded by this method opens up a wide range of new applications of these phenomena to visual effects.

There has been some work on simulating the effects of blast waves through analytical models of blast propagation. [Mazarak et al., 1999] used an expanding ball to determine forces on bodies to fracture or propel them. [Neff and Fiume, 1999] use similar analytic models of blast waves to fracture objects and are unable to generate the aforementioned effects of shock dynamics. These approaches are typically quite fast, but their extremely simple model of blast dynamics does not allow for the effects of shock-object interaction — notably reflection and vortex shedding — nor do they have the ability to visualize the blast itself.

3.3 Method

The key challenge is to simulate shockwave and compressible gas dynamics by designing a practical numerical method that can stably handle moving boundary conditions in three-dimensional space and is efficient enough to be used in a visual simulation production pipeline. I present a basic introduction to the finite volume method and refer the readers to [Leveque, 2002] for more detail.

3.3.1 Conservation laws

The method seeks solutions to the Euler equations of gas dynamics. These equations form a *hyperbolic conservation law*, the general, three dimensional form of which is:

$$\mathbf{q}_t + \mathbf{F}(\mathbf{q})_x + \mathbf{G}(\mathbf{q})_y + \mathbf{H}(\mathbf{q})_z = 0 \tag{3.1}$$

where subscripts indicate partial differentiation.

Here $\mathbf{q}$ is the vector of unknowns and $\mathbf{F}$, $\mathbf{G}$, and $\mathbf{H}$ are vector-valued *flux functions* specific to each conservation law. A conservation law states that a quantity of unknowns $\mathbf{q}$ over an arbitrary domain S changes in time due only to the flux across the boundary ∂S of the domain.

3.3.1.1 Integral form

The derivatives found in partial differential equations such as Equation (3.1) are not defined around discontinuities; to capture them properly, I use an integral form of the equations.

In one dimension, consider an interval $[a, b]$; the change in $\mathbf{q}$ over that interval is due to the flux at a and the flux at b. More formally,

$$\frac{d}{dt}\int_a^b \mathbf{q}(x,t)\,dx = \mathbf{F}(\mathbf{q}(a,t)) - \mathbf{F}(\mathbf{q}(b,t)) \tag{3.2}$$

where $\mathbf{F}$ is a flux function such as $\mathbf{F}$, $\mathbf{G}$, or $\mathbf{H}$ from Equation (3.1) and I follow the convention that 'positive flux' is left-going and 'negative flux' right-going.

3.3.2 The finite volume method

The finite volume method (FVM) on regular grids follows directly from Equation (3.2); the presentation here is for scalar equations in one dimension with scalar unknowns q and scalar fluxes f, but the formulae for systems of equations in multiple dimensions are straightforward extensions of these.

The spatial interval $[a, b]$ is discretized into m intervals ("cells") of equal size $\Delta x = \frac{b-a}{m}$. For each time t_n, the m quantities Q_i^n are defined as the average value of q over the cell:

$$Q_i^n = \frac{1}{\Delta x} \int_{\chi_i^l}^{\chi_i^r} q\left(x, t_n\right)\, dx \tag{3.3}$$

where $\chi_i^l = a + i\Delta x$, and $\chi_i^r = \chi_i^l + \Delta x$ are the positions of the cell boundaries. Observe that $\chi_{i-1}^r = \chi_i^l$. I will also occasionally refer to these boundary positions with half-index increments; for example, the flux at χ_i^l is $F_{i-\frac{1}{2}}$.

I apply Equation (3.2) to each of the intervals i

$$\frac{d}{dt} \int_{\chi_i^l}^{\chi_i^r} q(x,t)\, dx = f\Big(q\big(\chi_i^l, t\big)\Big) - f\Big(q\big(\chi_i^r, t\big)\Big) \tag{3.4}$$

and integrate Equation (3.4) from t_n to t_{n+1}

$$\int_{\chi_i^l}^{\chi_i^r} q\left(x, t_{n+1}\right)\, dx - \int_{\chi_i^l}^{\chi_i^r} q\left(x, t_n\right)\, dx = \int_{t_n}^{t_{n+1}} f\Big(q\big(\chi_i^l, t\big)\Big) - f\Big(q\big(\chi_i^r, t\big)\Big)\, dt \tag{3.5}$$

Observe that we can substitute Equation (3.3) if we divide Equation (3.5) by Δx.

$$Q_i^{n+1} - Q_i^n = \frac{1}{\Delta x} \int_{t_n}^{t_{n+1}} f\Big(q\big(\chi_i^l, t\big)\Big) - f\Big(q\big(\chi_i^r, t\big)\Big)\, dt \tag{3.6}$$

The right-hand side of this equation is a *flux difference* that cannot generally be evaluated exactly; I approximate the integrals with averages over the cell interfaces from $[t_n, t_{n+1}]$:

$$F^n_{\chi^l_i} \approx \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} f\Big(q(\chi^l_i, t)\Big)\, dt \tag{3.7}$$

Substituting Equation (3.7) into Equation (3.6), one obtains the most basic FVM update scheme (Equation (3.8)):

$$Q^{n+1}_i = Q^n_i - \frac{\Delta t}{\Delta x}\left(F^n_{\chi^l_i} - F^n_{\chi^r_i}\right) \tag{3.8}$$

This scheme is first-order and is subject to the numerical viscosity typical of first-order methods. Second-order schemes such as Law-Wendroff [Lax and Wendroff, 1960] can be employed with comparable computation effort; I complement this with a *flux limiter*, which minimizes diffusive and dispersive artifacts. I have used the MC limiter of [van Leer, 1977] in the method presented here.

3.3.3 The Riemann problem

According to Equation (3.7), the flux at $\mathbf{F}_{i-\frac{1}{2}}$ is dependent on the state $\mathbf{Q}_{i-\frac{1}{2}}$ at the interface between $\mathbf{Q}_{i-1}$ and $\mathbf{Q}_i$ over the interval $(t_n,\, t_{n+1})$; thus one must determine the value at this interface as it evolves in time. $\mathbf{Q}_i$ here is the vector version of the discrete unknowns first introduced in 3.3.2. Given the initial data:

$$\mathbf{Q}(x, 0) = \begin{cases} \mathbf{Q}_l, & x < 0 \\ \mathbf{Q}_r, & x \geq 0 \end{cases} \tag{3.9}$$

I wish to solve for $\mathbf{Q}(x, t)$ for $t > 0$ subject to the governing equations. This formulation is known as the *Riemann problem*; the resulting $\mathbf{Q}(0, t)$ obtained can then be used to compute the flux at the cell interface.

Numerical methods based on Riemann solvers can often succeed where other meth-

ods fail because their solution is achieved through analysis of the governing equations. Whereas many techniques are developed through the mechanical application of numerical stencils to the terms of an equation, Riemann solvers inherently incorporate more information about the equation in their formulation.

3.3.3.1 Riemann problem for linear systems

Consider linear, constant-coefficient (but not necessary scalar) hyperbolic conservation laws, i.e. Equation (3.1) where the flux function $\mathbf{F}$ takes the form $\mathbf{F}(\mathbf{q}) = \mathbf{A}\mathbf{q}$, where $\mathbf{A}$ is a *flux matrix*. (Assume that the other flux functions $\mathbf{G}$, and $\mathbf{H}$ are of the same form).

Such a system of order n can be diagonalized into n decoupled equations $\mathbf{Q}^+_t + \Lambda \mathbf{Q}^+_x = 0$, where $\mathbf{Q}^+ = \mathbf{R^{-1}Q}$. Here $\mathbf{R}$ is the matrix of right eigenvectors of $\mathbf{A}$, and Λ is a diagonal matrix of the eigenvalues of $\mathbf{A}$ satisfying $\mathbf{A} = \mathbf{R}\Lambda\mathbf{R}^{-1}$.

The solution to the Riemann problem for these equations is given by n weighted eigenvectors $W_i = \alpha_i \mathbf{r}_i$ (also known as *waves*) propagating with speeds λ_i, the corresponding eigenvalues.

The waves W_i are determined by projecting the jump in the initial states $\Delta\mathbf{Q} = \mathbf{Q_r} - \mathbf{Q_l}$ onto the space formed by the eigenvectors of the system:

$$\sum_i W_i = \sum_i \alpha_i \mathbf{r_i} = \Delta\mathbf{Q}$$

$$\mathbf{Ra} = \Delta\mathbf{Q} \tag{3.10}$$

$$\mathbf{a} = \mathbf{R}^{-1}\Delta\mathbf{Q} \tag{3.11}$$

where $\mathbf{a} = [\alpha_0, \alpha_1, \ldots, \alpha_{n-1}]^{\mathrm{T}}$

The waves define k intermediate states $\mathbf{Q}^{*_i} = \mathbf{Q}_l + \sum_{j=0}^{i} W_j$, and the solution to the

Riemann problem is therefore the piecewise-constant function

$$\mathbf{Q}(x,t) = \begin{cases} \mathbf{Q}_l, & x < \lambda_1 t \\ \vdots & \vdots \\ \mathbf{Q}^{*i}, & \lambda_i t < x \leq \lambda_{i+1} t \\ \vdots & \vdots \\ \mathbf{Q}_r, & x \geq \lambda_n t \end{cases} \tag{3.12}$$

3.3.3.2 The Riemann problem for nonlinear systems

For nonlinear systems such as the Euler equations, the wave structure of the solution is much more complicated and costly to compute — typically, iterative root-finding methods must be employed at each cell interface to determine the intermediate states $\mathbf{Q}^{*i}$.

However, it is often possible to obtain good results by approximately solving the Riemann problem; through linearizations of the flux evaluated at carefully chosen states, one can obtain solutions that fit Equation (3.12). Such approximate Riemann solvers must be used with care, as they can often produce non-physical solutions. I discuss the applicability of these solvers and how these undesirable conditions can be addressed in Section 3.3.4.1.

3.3.3.3 Upwinding flux splitting

The basic FVM Equation (3.8) update scheme developed in Section 3.3.2 is not able to stably handle hyperbolic systems; it needs to be modified to obey the principle of *upwinding*. One must take care to ensure that waves traveling in the positive direction use information from the *negative* direction.

Rather than use Equation (3.8) to compute cell updates, I employ a scheme

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}\left(F^{n+}_{\chi_i^r} + F^{n-}_{\chi_i^l}\right) \tag{3.13}$$

where $F^{n-}_{\chi^r_i}$ is the part of the flux $F^n_{\chi^r_i}$ traveling in the negative direction and $F^{n+}_{\chi^r_i}$ the part traveling in the positive direction.

The waves W_i and speeds λ_i from the solution to a Riemann problem at χ^r_i is then

$$F^{n-}_{\chi^r_i} = \sum_{c=0}^{j} \lambda_c W_c \quad F^{n+}_{\chi^r_i} = \sum_{c=j}^{k} \lambda_c W_c \tag{3.14}$$

Where $\ldots < \lambda_j < 0 < \lambda_{j+1} < \ldots$; waves traveling with negative speeds are added to F^{n-} while those traveling with positive speed are added to F^{n+}.

3.3.3.4 Solution procedure

Given cell values $\mathbf{Q}^n$ for time t_n, a timestep is performed as follows to compute $\mathbf{Q}^{n+1}$:

1. For each interface between cells, compute the wavespeeds λ_i and fluxes F^n_i by solving the Riemann problem at that interface (described in Section 3.3.4.2)

2. Find the wavespeed with largest magnitude from $|\lambda_i|$ to compute timestep length Δt as described in Section 3.3.4.4.

3. For each cell i, advance to next time $\mathbf{Q}^{n+1}$ using the fluxes F^n at its neighboring interfaces χ^l_i, χ^r_i using Equation (3.13).

For three-dimensional problems (see Section 3.3.4.4), one must compute three fluxes for each cell in the domain; solving the Riemann problems in step 1 becomes the computational bottleneck for non-trivial systems of equations. While expensive to obtain, carefully calculated fluxes are the key to handling discontinuous solutions on a coarse grid. Next, I describe what the Riemann problem is and how it can be used to compute flux between cells.

3.3.4 The Euler equations

I are interested in studying the motion of a compressible gas; the natural choice is the Euler system of equations. The simplification of Navier-Stokes that omits viscous terms results in this nonlinear hyperbolic system of conservation laws. The omission of viscosity is a reasonable one to make for many physical problems in gas dynamics, just as the incompressible simplification of Navier-Stokes frequently used in graphics is reasonable for liquid simulation.

The Euler equations in conservation form (see Equation (3.1)) are

$$\mathbf{q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix}, \quad \mathbf{F}(\mathbf{q}) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (E+p)u \end{bmatrix}$$

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ (E+p)v \end{bmatrix}, \quad \mathbf{H}(\mathbf{q}) = \begin{bmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ (E+p)w \end{bmatrix} \tag{3.15}$$

Here ρ is density, u, v, and w the components of velocity, p the pressure, and E the total energy. An additional *equation of state* completes the system

$$E = \frac{p}{\gamma - 1} + \frac{\rho}{2}\left(u^2 + v^2 + w^2\right) \tag{3.16}$$

where γ is the adiabatic exponent of the fluid — typically 1.4 for air. It should be noted that for solutions to be physically valid, ρ, p, and E must all be strictly greater than zero.

3.3.4.1 Approximate Riemann solutions

As discussed in Section 3.3.3.2, computing the exact solution to the Riemann problem for nonlinear systems such as the Euler equations is prohibitively expensive for practical problems. Suitably approximated solutions to the Riemann problem are often able to achieve acceptable results for a fraction of the cost of solving them exactly.

To apply the method for solving Riemann problems for linear systems presented in Section 3.3.3.1 to nonlinear problems; one desires a matrix $\mathbf{A}$ such that $\mathbf{A}$ approximates $\mathbf{F}'(\mathbf{Q})$; here $\mathbf{F}'(\mathbf{Q})$ is the Jacobian of $\mathbf{F}$ as seen in the *quasilinear* form of the conservation law. This is simply the chain rule applied to (3.1): $\mathbf{Q}_t - \mathbf{F}(\mathbf{Q})_x = \mathbf{Q}_t - \mathbf{F}'(\mathbf{Q})\mathbf{Q}_x = 0$.

In a seminal paper, [Roe, 1981] presented a simple method for approximating $\mathbf{F}'(\mathbf{Q})$ that preserves important conditions of the system, and it is this method that I have adapted for this solver. Roe's method uses a flux matrix $\mathbf{A}$ that is $\mathbf{F}'(\bar{\mathbf{Q}})$ evaluated at a specially chosen state $\bar{\mathbf{Q}}$ given $\mathbf{Q}_l$ and $\mathbf{Q}_r$ — this state has come to be known as the *Roe average state*.

Eigenvectors and eigenvalues of the flux Jacobian The eigenvectors of the Jacobian $\mathbf{F}'(\mathbf{Q})$ give the waves necessary to compute the intermediate states as in Section 3.3.3.1, and its eigenvalues give the characteristic speeds λ_i with which these waves propagate. The eigenvalues of the flux Jacobian $\mathbf{F}'$ as computed from (3.15) are:

$$\lambda_{0\ldots4} = (u - c, u, u, u, u + c) \tag{3.17}$$

and the corresponding eigenvectors are:

$$\mathbf{r}_1 = \begin{bmatrix} 1 \\ u-c \\ v \\ w \\ H-uc \end{bmatrix} \quad \mathbf{r}_2 = \begin{bmatrix} 1 \\ u \\ v \\ w \\ \frac{1}{2}(u^2+v^2+w^2) \end{bmatrix}$$

$$\mathbf{r}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ v \end{bmatrix} \quad \mathbf{r}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ w \end{bmatrix} \quad \mathbf{r}_4 = \begin{bmatrix} 1 \\ u+c \\ v \\ w \\ H+uc \end{bmatrix} \tag{3.18}$$

Here $c = \sqrt{\frac{\gamma p}{\rho}}$ is the speed of sound and $H = \frac{E+p}{\rho}$ the total specific enthalpy. I have given only the eigenvalues and eigenvectors for $\mathbf{F}'$, but those for the Jacobians of the other flux functions $\mathbf{G}'$ and $\mathbf{H}'$ have similar structure.

Roe average state Given $\mathbf{Q}_l = [\rho_l, u_l, v_l, w_l, E_l]$ and $\mathbf{Q}_r = [\rho_r, u_r, v_r, w_r, E_r]$, the Roe average is

$$\bar{\mathbf{Q}} = \left[\bar{\rho}, \bar{u}, \bar{v}, \bar{w}, \bar{H}\right]^{\mathrm{T}} \qquad \bar{\rho} = \tfrac{\rho_l+\rho_r}{2} \tag{3.19}$$

$$\bar{u} = \frac{\sqrt{\rho}_l u_l + \sqrt{\rho}_r u_r}{\sqrt{\rho}_l + \sqrt{\rho}_r} \quad \bar{v} = \frac{\sqrt{\rho}_l v_l + \sqrt{\rho}_r v_r}{\sqrt{\rho}_l + \sqrt{\rho}_r} \tag{3.20}$$

$$\bar{w} = \frac{\sqrt{\rho}_l w_l + \sqrt{\rho}_r w_r}{\sqrt{\rho}_l + \sqrt{\rho}_r} \quad \bar{H} = \frac{\frac{E_l+p_l}{\sqrt{\rho}_l} + \frac{E_r+p_r}{\sqrt{\rho}_r}}{\sqrt{\rho}_l + \sqrt{\rho}_r} \tag{3.21}$$

The specific variables shown here (in contrast to the conservative variables given in (3.15)) appear because they are precisely what is necessary to evaluate the eigenvalues Equation (3.17) and eigenvectors Equation (3.18) and obtain the waves and speeds for a given Riemann problem.

This average state has attractive properties when considering the structure of the Riemann problem; were I to choose a simple arithmetic average of the quantities at $\mathbf{Q}_l$ and $\mathbf{Q}_r$, the resulting eigenvectors may fail to be distinct and the solution would fail entirely. The criteria behind this particular choice of average are explained in detail in [Roe, 1981] and [Leveque, 2002]; other linearizations are possible — Roe's enforces conservation and exactly captures single discontinuities.

Enforcing physicality Using the Roe average state (Equation (3.19)) to approximately solve the Riemann problem is significantly faster then computing the exact solution to the Riemann problem, but the solver is known to generate nonphysical states for certain inputs $\mathbf{Q}_l$ and $\mathbf{Q}_r$ — in particular, rarefactions can be improperly represented as discontinuities. While the exact solution to the Riemann problem could be computed to obtain the physically valid intermediate state, this is unnecessary and overly expensive for visual simulation. When the approximate Riemann solver produces invalid states, I apply slight corrections to enforce physicality. I clamp ρ and p to be no less than 0.05 — in the case of p, this entails adjusting E according to Equation (3.16).

For example, the fluid-rigid body simulations illustrated in Figures 3.8, 3.11, 3.1, and 3.2 demonstrate plausible motion, and would not be possible using a simple approximate Riemann solver without these corrections.

3.3.4.2 Riemann solver for Euler equations

I have developed the theory of Riemann solvers for the Euler equations sufficiently to present the procedure for computing the Riemann solution at an interface given left and right states $\mathbf{Q}_l$ and $\mathbf{Q}_r$:

1. Compute Roe average $\bar{\mathbf{Q}}$ using Equation (3.19)
2. Make $\bar{\mathbf{Q}}$ physically valid if needed, as per Section 3.3.4.1
3. Compute wavespeeds λ_i using Equation (3.17)

4. Compute eigenvectors $\mathbf{r}_i$ using Equation (3.18)

5. Project $\Delta\mathbf{Q}$ onto the eigenspace by computing the wave coefficients α_i and waves W_i using Equation (3.11)

6. Compute left and right fluctuations $F^{n\pm}$ using Equation (3.14)

3.3.4.3 Boundary conditions

Boundary conditions are applied where needed through modified Riemann solvers; these do not solve for the flux at an interface due to two adjacent cells; I compute a 'ghost' intermediate state at the interface to determine these fluxes. In practice, I have found three types of boundary conditions useful:

Free-slip: This common boundary condition simply states that the component of flow normal to the interface is zero. I obtain this by modifying the Roe average Equation (3.19) used in the cell to have zero velocity in the component normal to the boundary; thus $\bar{u}$ on a free-slip boundary normal to the x-direction is set to zero. Other components of the intermediate state $\bar{\mathbf{Q}}$ are simply treated as though $\mathbf{Q_l}$ were equal to $\mathbf{Q_r}$.

Velocity: This is a generalization of free-slip boundary conditions; rather than enforce zero velocity along an interface, some user-specified velocity is imposed as the intermediate component of velocity in the appropriate direction. Other components are treated as though the two adjacent cells were identical except for the energy E; given an imposed velocity $\bar{u}$ and the same component of velocity in the adjacent cell u_r, the velocity in the ghost cell is $u_l = 2\bar{u}(\bar{u} - u_r)$. Due to this difference in velocity, the energy in the ghost cell is not equal to its neighbor and is adjusted with Equation (3.16).

Absorbing: It is often desirable to perform simulations where outgoing waves are simply absorbed rather than reflected; the computational domain behaves as if it

were suspended in an infinite passive medium. At these interfaces, the fluxes in the Riemann problem are simply set to zero.

3.3.4.4 Dimensional splitting

The discussion so far has been limited to one dimension — the Equations (3.15) are three-dimensional, but the solution procedure in Section 3.3.3.4 performs updates in only a single dimension.

To solve three-dimensional problems, this method performs *dimensional splitting*. To advance from time t_n to t_{n+1}, the technique makes sub-step "passes" of a one-dimensional solver in each direction — first using the flux function $\mathbf{F}$ along x for all rows of constant y and z, then using $\mathbf{G}$ along y for all rows of constant x and z, and finally using $\mathbf{H}$ along z for all rows of constant x and y.

This approach allows the application of the one-dimensional techniques previously described here in a straightforward manner; however, I must address how best to choose the timestep to take over the three passes.

Choosing a timestep The timestep size Δt that the method is able to take while advancing the solution with Equation (3.8) is limited by the maximum characteristic speed $\lambda_{\max}$ from Equation (3.17) in the solution being updated, as per the Courant-Friedrichs-Lewy (CFL) condition [Courant et al., 1928]. For simulation in a single dimension, the procedure in Section 3.3.3.4 works perfectly — the solution to all Riemann problems in the domain is computed, which gives the maximum characteristic speed, which is then used to compute the timestep $\Delta t = \frac{\Delta x}{\lambda_{\max}}$. With dimensional splitting, it is not possible to compute the maximum speed in dimension y prior to advancing the solution in x with some previously chosen Δt; the maximum speed in y depends on the results of the x-pass and is not generally equal to the $\lambda_{\max}$ from the x pass.

There are several ways to address this problem — for example, I could adopt a guess-

and-check approach of estimating a timestep, advancing the solution with it, checking to see if it satisfies the CFL condition based on the maximum speed of the next level, and rewinding the whole computation if not, but this would be prohibitively expensive.

I take the very simple approach of always advancing the solution in a dimension with the largest timestep that satisfies the CFL condition in that dimension. This method clearly has effects on the solution; effectively, the grid is 'warped' over a timestep based on the ratios of maximum speeds in each dimension. However, I have found these effects to be negligible in the simulations I have run, even in cases where the flow (and therefore $\lambda_{\max}$) is highly biased along a single dimension (see for example Figures 3.8, 3.9, and 3.11).

This approach has an advantage over other methods and is particularly desirable for visual simulation; each dimension is advanced according to the chosen CFL number of the simulation. No dimension is forced to take a timestep at a low CFL number because of other, higher speed dimensions. This technique helps reduce the numerical artifacts that frequently plague visual simulations of natural phenomena.

3.3.5 Fluid-object interaction

I achieve bidirectional coupling through voxelization (as in [Carlson et al., 2004] and [Batty et al., 2007]). The technique presented here uses simple modifications to the Riemann solvers on boundary interfaces to affect the interaction; this is an explicit scheme that is simple and efficient.

To capture the objects' effect on the fluid, I use the aforementioned free-slip modification to the Riemann solver along the boundary (in Section 3.3.4.3). This solver ensures that incoming waves are reflected off of solid bodies and enables effects like those seen in Figures 3.1, 3.2, 3.8, 3.10, and 3.11; these demonstrate the effects of the solids in the scene on the flow.

The force exerted by the fluid on the objects is obtained by multiplying the pressures

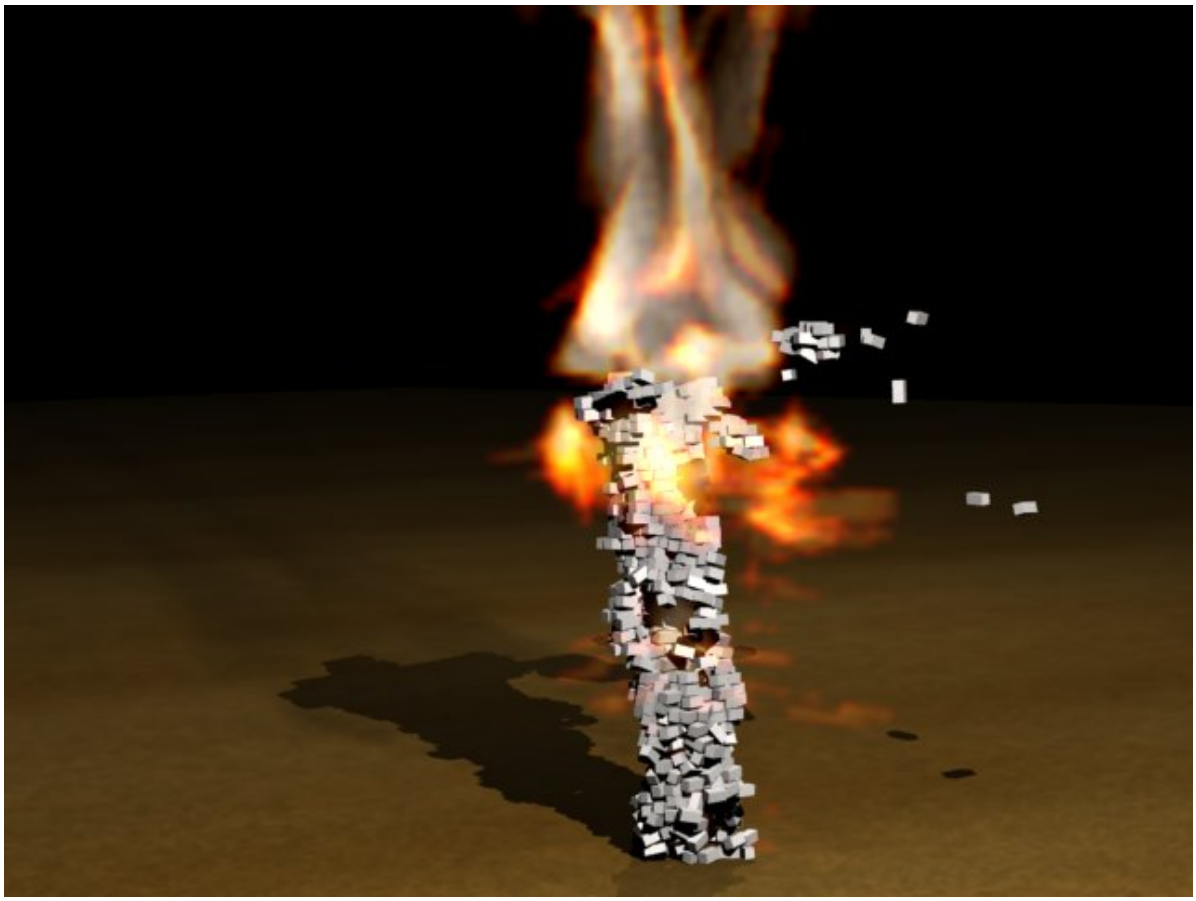


Figure 3.1: Tower (without cap) blown apart by internal blast

at each incident cell by the interface's normal direction and applying the resulting force to the object. This simple technique is responsible for the forces buffeting the objects in Figures 3.1, 3.2, 3.8, and 3.11.

Any rigid body simulator is suitable for use with this method; I have used the *Bullet* collision and dynamics engine [Game Physics Simulation, 2010] because of its completeness and availability. Voxelization is achieved with a simple custom tool based on triangle-grid intersections.

Considerable work [Carlson et al., 2004, Chentanez et al., 2006, Batty et al., 2007] has been done to achieve stable fluid-solid interactions in the past, but these methods have focused on the interaction of rigid and deformable bodies with *incompressible* fluids. Stability problems frequently arise in such situations because of the differing needs of the

Figure 3.2: Tower (with cap) blown apart by internal blast

rigid body dynamics and the fluid simulator; the implicit solver for incompressible fluid simulation generally takes large timesteps, which can result in a loosely-coupled, unstable simulation when rigid bodies are handled naïvely. This method naturally takes many small, inexpensive timesteps to advance the solution; this allows tighter communication between the rigid body and fluid simulators and results in a more stable interaction.

3.4 Parallelization

The vast majority of the computation time in the algorithm described in Section 3.3.3.4, is spent in two kernels; the computation of solutions to the many Riemann problems across the grid (as described in Section 3.3.4.2) and the application of these Riemann so-

lutions to the cells of the grid to advance to the next timestep (see Equation (3.13)). Each of these kernels is executed once per dimensional pass (as described in Section 3.3.4.4.)

The computation of Riemann solutions is essentially independent across all cell interfaces along the current dimension. The two cells adjacent to a given interface determine the fluxes and speeds that comprise the Riemann solution at that interface.

The update procedure is similarly data-parallel across each grid cell; to update a cell, I need only the global timestep being used for this dimensional pass (computed as per Section 3.3.4.4) and the Riemann solutions corresponding to the two interfaces shared with the cell's neighbors along the current dimension.

One therefore expects to achieve significant performance scaling from a parallelization of these kernels across the grid. As I shall show, there are a great number of factors to take into consideration when developing an effective parallel computation scheme.

3.4.1 Initial parallelization

The two computation kernels described above are parallel across each interface and grid cell, respectively, but the number of these generally exceeds the number of processors available by several orders of magnitude, so it is reasonable to assign groups of these computations to each processor.

An obvious way of doing this is to group 'rows' of computation — for both kernels, one can consider the computation performed on the interfaces or cells along each row of the current pass as a single task. These tasks may then themselves be partitioned among the available processors; Figure 3.3 depicts this scheme. This scheme is exceedingly simple to implement atop an existing serial implementation but scales poorly, achieving about 8x scaling on 16 processors. Figure 3.15 shows the scaling results for this scheme.

At first, the poor behavior of this approach may seem surprising, since it has many of the hallmarks of a good parallel algorithm. There is no explicit communication between threads, only synchronization barriers at the end of each kernel. The threads execute

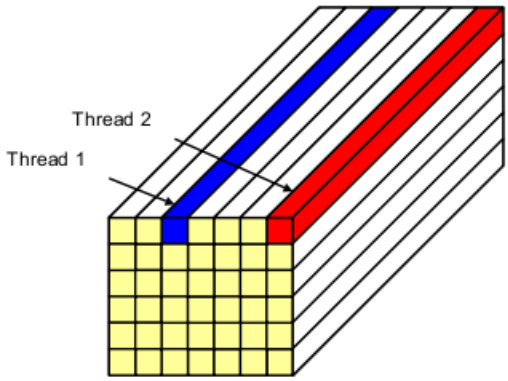


Figure 3.3: Computation of Riemann solutions and solution updates done in a pass are divided among threads

nearly identical code paths on equally-sized portions of the grid; one can therefore be confident that each thread performs a similar amount of work in each kernel. Additionally, the algorithm is compute-intensive — the Riemann solution kernel performs over 400 floating-point operations per interface, so concerns over bandwidth are mitigated.

The reason why this approach scales poorly becomes apparent when one considers the memory hierarchy of modern computer architecture and multi-core processor layout; after taking the memory layout into account, a superior parallelization scheme can be achieved.

3.4.2 GPU parallelism

The power of highly parallel commodity Graphics Processing Units (GPUs) has led many researchers to develop specialized implementations on these stream processor architectures in what is called General Purpose GPU programming, or GPGPU. The parallel algorithm outlined in Section 3.4.1 performs moderately well on small-scale modern shared-memory architectures; it certainly is a reasonable candidate for investigation on GPUs. In the following section, I present my strategy for implementing my shock simulation on moderns GPUs, discuss details and ramifications of implementation, and

present results on performance.

3.4.2.1 Implementation with CUDA

The very first programmable GPUs lacked high-level programming environments and had stringent limitations in code size (a significant barrier in for many computation problems). Before long, GPU-specific programming languages were developed — chief among them Nvidia's Cg [Mark et al., 2003], ATI and Microsoft's HLSL [Peeper and Mitchell, 2003], and OpenGL's GLSL [Kessenich et al., 2009] — which greatly simplified the process by which these increasingly complex specialty processors were programmed. These languages were primarily aimed at facilitiating graphics programming, and the computing environment still posed significant challenges to truly general purpose computing on GPUs.

More recently, more general platforms have emerged that aim to furnish the user with enough abstraction from the hardware to be a truly general solution. I chose Nvidia's CUDA [CUDA, 2009] platform because of its well-established position and stability.

CUDA's general computing model is based upon multiple *blocks* of *threads*. The threads in a block have access to some limited shared memory and are executed on a single processor unit; threads in a block can also be synchronized together.

A kernel is executed by specifying Bn, the number of blocks, and Tn, the number of threads in each block. The shared memory for each block is allocated at execution time and is valid over the lifetime of the kernel. Furthermore, there is no inter-block synchronization mechanism; only that all blocks have finished computation when a kernel has finished executing.

The current hardware supported by CUDA introduced some hard limits and strong guidelines to the way an algorithm is parallelized. The 8800 series has 16 processors with 16k of local storage space — since each block runs on a single processor, it is wise to have at least 16 blocks per kernel execution. Furthermore, each processor is able to

execute 32 threads concurrently, so one would do well to have at least 32 threads per block per kernel as well.

A block is assigned to each row in the grid along which a Riemann pass is to be performed, and divide that row into at least 32 threads. A reduction is performed to determine the global maximum in parallel across the threads in a block and then across all blocks. A dimensional pass is finished by performing the update pass in the same manner as the Riemann pass, with a block assigned to each row and 32 threads assigned to each block.

In the Riemann and update passes, the interfaces and cells along a given row in a block are divided evenly among the threads. In general, I avoid contention in reads; each thread in a block works sequentially on its portion, and thus the overlaps (where thread $n-1$'s last cell is the same as thread n's first cell) are taken care of.

I have implemented both absorbing and zero-velocity boundary conditions for the solution; this is simply achieved by using a modified Riemann solver at the boundary cells. I did not implement internal boundary conditions (due to walls or objects inside the fluid domain); while this would be a relatively simple thing to do, it is likely to cause thread divergence within a block and significantly degrade performance. The "best-case" speedup is of greater importance, and thus I focused on that.

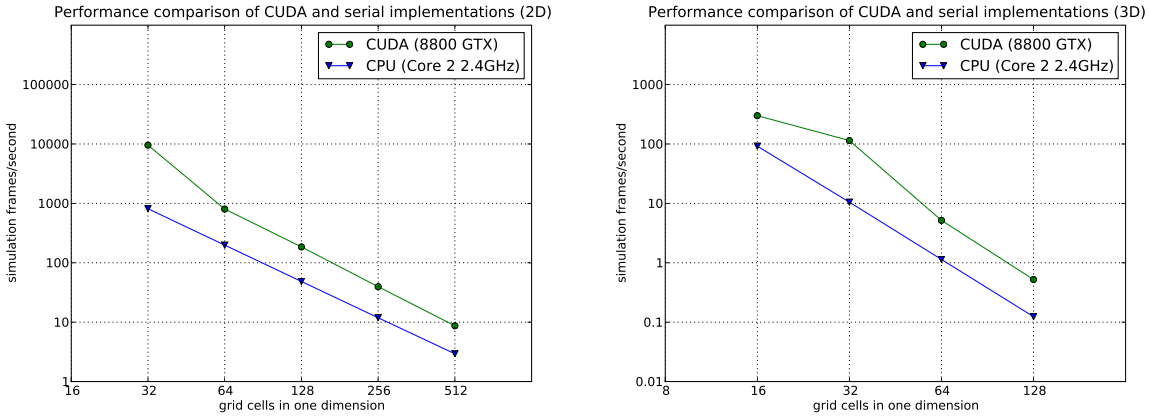
Unfortunately, a fundamental limitation of CUDA prevents the mapping of this algorithm to the hardware as I would like to. Modern graphics cards do not have a true bidirectional cache like a modern CPU; each processor instead has a small amount of local memory that acts as a manual cache for individual blocks to use.

In my algorithm, I compute a row of Riemann solutions per block and perform a reduction on these solutions to obtain the Δt necessary to combine with the Riemann solutions to advance the solution along each such row. Aside from the reduction, the Riemann solution computed in a block is completely local; it is only used to advance the solution along the same row after Δt has been computed.

Ideally, one would compute the per-row Riemann solutions in each blocks' shared memory, perform the reduction, and use the stored Riemann solution to update the solution, all without the need to write to global memory. However, CUDA's architecture prevents inter-block communication without clearing out each blocks' shared memory, since this is deallocated after kernels exit. Instead, each block must write its Riemann solution to the global memory space and read it back during the update pass.

Given that this "local compute, global reduce, local update" paradigm is a fairly common one in parallel programming, it is surprising that is not better supported by CUDA.

3.4.2.2 GPU results



(a) 2d **(b)** 3d

Figure 3.4: Comparative timings for CUDA and serial implementations.

The results of this implementation are shown in figure 3.4; the CUDA implementation's relative speed over the serial implementation is shown in figure 3.5. The low relative performance of CUDA for the 16^3 grid is understandable, since I am not able to fill each block with the full 32 threads needed to utilize all of the hardware. The spike at 32^3 is likely due to the the 16k shared memory that is allocated each block. The average speedup is about 4x, largely owing to a limitation in CUDA's memory

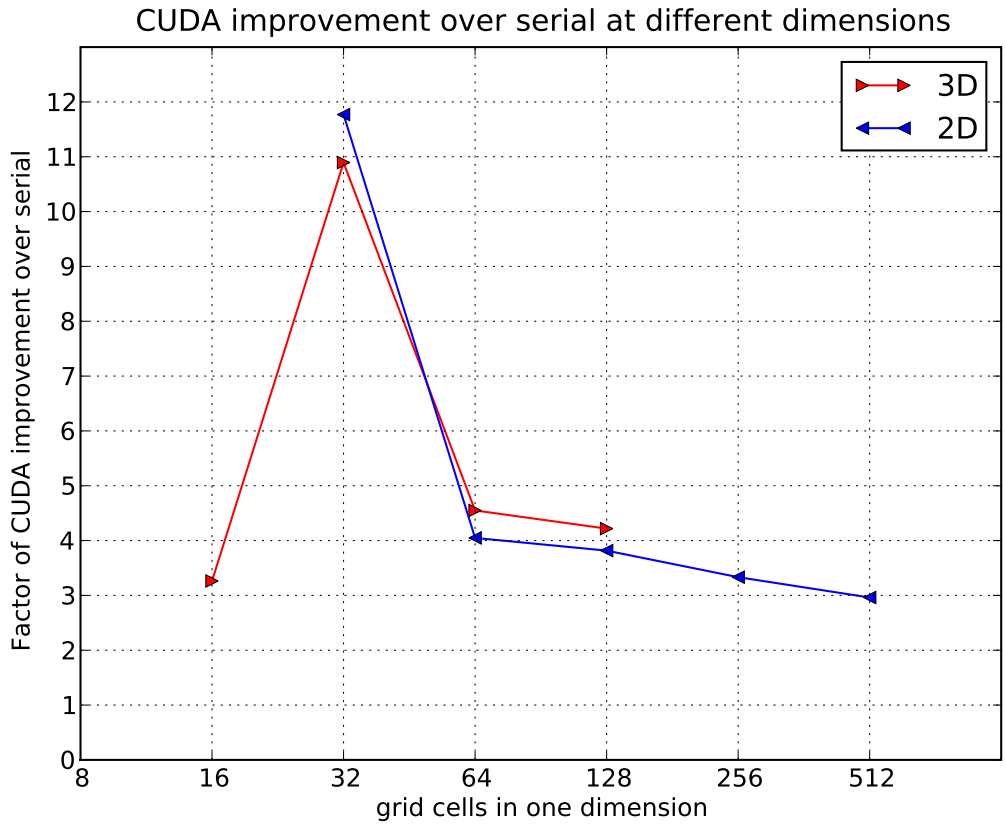


Figure 3.5: Speedup of CUDA over serial implementation.

model that prevents utilization of shared memory properly. My initial parallel scheme, implemented in OpenMP, achieves as much as 9–10x on a 16-core machines.. The trade-offs between these approaches is an interesting one; while the extension of the serial version to the OpenMP version was nearly zero-effort, one could argue that a 16-core computer is hardly commodity hardware. The CUDA implementation was considerably more work than the OpenMP, but runs well on mid-range graphics hardware — while machines with more than 4 processors are still beyond mainstream computing, a modestly powerful GPU can be found on many commodity machines.

It is interesting to consider how performance might change on newer graphics cards; the Nvidia GTX 480 and GTX 580 products feature Nvidia's GF100 architecture, which introduces several performance-enhancing features — most notably, a greater number

of stream processors and a true cache. While many applications have benefited from these changes, I predict that the same issue that dogs the GPU performance shown here would affect these newer chips as well. The CUDA compute model precludes lightweight synchronization and prevents us from effectively storing intermediate results. As a result, the computation is heavily bandwidth bound and would see little benefit from the architectural improvements in the latest Nvidia GPU products.

3.4.3 Many-core hardware considerations

Modern processors — specifically, those found in commodity desktop and laptop computers — utilize a hierarchical memory layout, with several levels of cache between the processor and main memory. The latency of cache memory is typically an order of magnitude faster than that of main memory; multiple transactions with a given memory location can be greatly sped up if the contents of said memory can be kept in the cache.

Performance-minded implementations of important algorithms — such as the linear algebra operations found in the Automatically Tuned Linear Algebra Software (ATLAS) [Whaley et al., 2001] package and the fast Fourier transforms found in the Fastest Fourier Transform in the West (FFTW) [Frigo and Johnson, 2005] library — are designed with CPU caches in mind, carefully blocking access patterns to maximize the effects of the cache's fast memory.

A serial implementation of this algorithm can be designed to traverse memory in a "cache-friendly" manner, but cannot be blocked in the same manner as some matrix operations. Ideally, the memory accesses during a timestep could be partitioned such that a portion of the grid is loaded into the cache and all operations necessary for that partition during the timestep would be performed before moving on to the next partition. The multi-pass nature of our algorithm requires that the method traverse the grid multiple times — for each dimensional pass, once to compute the solutions to the Riemann problems and determine the maximum speed, and again to apply the updates

to the grid.

However, an implementation on a parallel system will typically have much more cache available, albeit divided up among the various processors in the system. To effectively take advantage of the capabilities of a multi-core system it is therefore essential that one takes the various caches available into account.

The cache structure found in multi-core computers is quite intricate and can vary greatly from system to system. For example, Intel's Pentium D processor featured two cores, each with a separate 1MB L2 cache, while Intel's Core 2 Duo processor's two cores share a single 4MB L2 cache. The system upon which I performed the parallelization benchmarks has four sockets, each with an Intel Xeon processor — these processors are in turn composed of four cores and two 2MB L2 caches, with each cache shared by two of the cores.

Consider how the initial parallelization scheme presented in Section 3.4.1 behaves with a mind to memory access and cache behavior. For any single dimensional pass, each thread is assigned a disjoint portion of the grid to work with; assuming (for the moment) suitably aligned data, it is safe to say that no two threads will try to read or write the same location in memory.

However, each dimensional pass divides the grid up differently — for x, groups of rows of constant y and z, for y, groups rows of constant x and z, and for z groups of rows of constant y and z. This means that the portion of the grid assigned to each thread changes for each dimensional pass; in between each pass, all of the changes written by the pass's update kernel must be flushed out of all caches and exchanged. What initially looked like a moderately memory-intensive algorithm turns out to require tremendous amounts of bandwidth to satisfy the constantly-changing mapping of data to threads.

3.4.3.1 Cache lines and alignment

The above discussion of the effects of the system's caches on the initial parallelization is itself simplified; to fully appreciate the subtleties of caches and how they affect a parallel program's performance, one must consider how caches are filled.

Caches always fetch and store contiguous groups of memory of fixed length in quanta known as *cache lines*. The size of a cache line varies with architecture; the Xeon processors in the 16-core machine used for the parallelization benchmarks use 64-byte cache lines.

Every address in memory maps to exactly one cache line and a transaction with an address will result in the cache line to which it belongs being read into the cache. This has the ramification that certain memory access patterns can be very inefficient; access with strides greater than a cache line, for example, can result in wasted bandwidth and cache use.

Parallel programs are subject to a more subtle issue due to cache lines: *false sharing*. Consider a situation in which a region of memory is partitioned among multiple threads, such as the partitioning of groups of rows in my initial parallelization scheme. Depending on how the memory is partitioned and its alignment with respect to cache line boundaries, it is possible for multiple threads to be assigned the same cache line. If these threads are writing to these shared cache lines, a considerable amount of bandwidth and time is consumed as the cache line is read, written to and flushed by one thread after another.

False sharing can be prevented by ensuring that no two threads are reading and writing the same cache line; this is typically accomplished by ensuring that data structures are allocated along cache line boundaries and that shared portions of memory are padded appropriately.

3.4.3.2 Processor affinities

I have heretofore used the terms threads and processors interchangeably, assuming that each thread in the parallelization runs on a single processor for the entirety of its lifetime. In fact, operating systems are free to assign threads to any processor and *migrate* them to other processors during runtime.

Thread migration can result in a significant performance penalty for an algorithm designed to maximize the benefits of cache locality. Fortunately, most modern operating systems support the assignment of *affinity masks* to threads, which enumerate the set of processors the thread may execute on. Through this mechanism, I am able to request that each thread to run on a specific processor and therefore ameliorate the effects of thread migration.

The initial approach to parallelization described in Section 3.4.1 does not scale well. Considering the effects of cache contention that occurs between dimensional passes and the likelihood of false sharing, the poor performance is understandable. With the limitations of this inferior scheme in mind, I will now describe a new parallelization scheme that performs much better.

3.4.4 Domain decomposition

Given a grid of dimension $l \times m \times n$ and a number of processors p, split the grid into p rectilinear *tiles* using planes in x, y, and z. Note that I am presenting this algorithm in three dimensions, but is easily applied to two-dimensional problems.

The exact arrangement of the tiles depends on the factors of p, but it is desirable for each tile to represent an equal amount of work along each dimension. For example, given a $64 \times 64 \times 64$ grid and 12 processors, the simulation might decompose the problem into $2 \times 3 \times 2$ tiles, with 8 tiles of dimension $32 \times 21 \times 32$ and 4 tiles of dimension $32 \times 22 \times 32$ (see Figure 3.6). Within each tile, the solution is updated according to the serial algorithm described in Section 3.3.3.4 except:

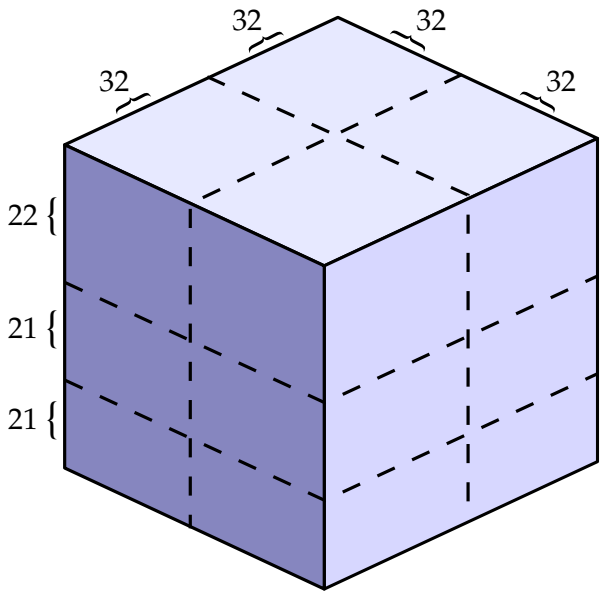


Figure 3.6: Decomposition of a 64^3 grid into 12 tiles: $2 \times 3 \times 2$

- Step 1 and step 3 are each computed within synchronization barriers across all threads.
- In each dimensional pass, step 2 computes Δt based on the largest speed found in *all* tiles and the resulting Δt is used in step 3 in each tile.
- The Riemann problems computed at interior boundaries — that is, those boundaries shared by adjacent tiles — must take into account the cells of the adjacent tiles. Boundaries that tiles share with the original grid are computed as described in Section 3.3.4.3.

To properly handle the computation of Riemann problems at interior boundaries, one needs to make the values of the cells along the boundary shared with each adjacent tile available. For each tile, for each adjacent tile (with which it necessarily shares an internal boundary), a buffer is maintained into which a copy of the necessary cells is written directly before the data is needed for the associated Riemann solution pass. Then, rather than computing a special simplified variant of the Riemann problem based on the boundary condition as in the standard algorithm, values from the appropriate

neighbor buffer are used.

During the neighbor-update step for a dimensional pass, each tile copies the necessary values from its down grid data to the appropriate buffers belonging to its neighbors. Note that it may be tempting to eliminate the need for these buffers by simply have a tile read the data directly from a appropriate neighbor tile's grid, but doing so risks false sharing behavior and wasted bandwidth.

Indeed, rather than share one global grid data structure as in the initial scheme, i eliminate the possibility of false sharing by having each tile allocate its own grid, suitably aligned and padded so as to share no cache lines with other tiles' data. This slightly complicates the output of grid data, as each tile must carefully copy its own portion of the aggregate grid to the output location.

The results of this scheme are shown in Figure 3.16; the scaling has improved slightly for small grid sizes, but is slightly *worse* than the initial scheme for larger grid sizes.

One more optimization is necessary to achieve the desired scaling. A decrease in scaling for large grid sizes suggests that the method is bandwidth limited; for these large grid sizes, each tile's portion of the grid no longer fits in cache, and the increased accesses to main memory begins to saturate the bus for large numbers of processors.

3.4.4.1 Reducing memory usage

To reduce the bandwidth requirements of the algorithm and improve scalability, the working set associate with each tile must be reduced. The original solution procedure as described in Section 3.3.3.4 computes the full Riemann solution at each interfaces in step 1, uses the computed speed to determine Δt in step 2, and saves the computed fluxes for the updates in step 3.

This memoization saves computation but consumes a considerable amount of bandwidth; the full Riemann solution for an interface of the 3-dimensional Euler equations contains 5 waves (which are vectors of 5 values) and 5 speeds, whereas a single cell of the

grid has only a single vector of 5 values (see Equations (3.17) and (2.19)). Since there are roughly the same number of interfaces where these Riemann solutions are stored as there are actual cells, saving the solution to all of the Riemann solutions in a pass requires 6 times the storage of the grid alone.

This algorithm can be modified to instead compute just the maximum speeds at each interface in step 1 and compute the full Riemann solutions as they are needed to advance the solution in step 3. There is a net increase in actual computation, since I am computing a significant portion of the Riemann solution at each interface in step 1 just to determine the maximum speeds, but the expected reduction in bandwidth can greatly improve the scaling of the algorithm.

As demonstrated in Figure 3.17, the scaling is now very close to linear for the larger problem sizes. The 32^3 grid does not scale as well because the relative overhead of copying data during the neighbor-update as compared to the work done computing the solution is larger for a smaller grid.

The dip in performance in the 32^3 grid for 14 threads is due to the factors being 2 and 7; the grid cannot be divided evenly among 7 tiles in one dimension, thus there are 6 tiles with a dimension along one axis of 4 while the remaining has a dimension of 9. This disparity in workload size is particularly exaggerated at the small grid size.

3.5 Results

I have implemented and tested this algorithm on several challenging scenarios. In this section, I first show some demonstrations of the algorithm, then describe our rendering methods, and finally discuss timing.

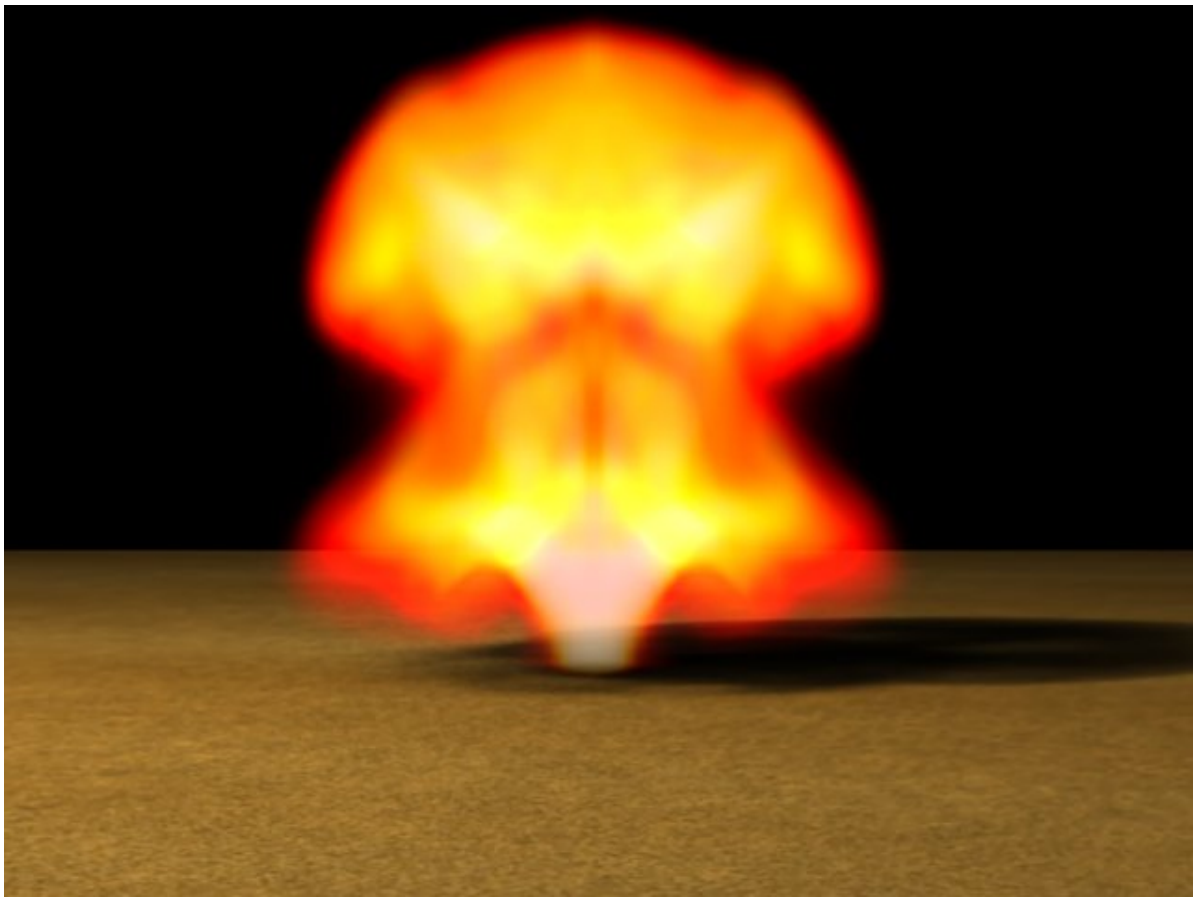


Figure 3.7: A mushroom cloud generated by my method

3.5.1 Applications

I have constructed a number of scenarios that demonstrate the ability of this method to simulate visually interesting phenomena. The first segment of supplementary video is a two-dimensional simulation demonstrating vortex shedding — a traveling shock crosses a sharp obstacle and a powerful vortex forms in its wake. Further reflections of the shocks create new vortices which combine and travel around the domain; this scenario is also shown in Figure 3.12. Figure 3.7 shows a mushroom cloud formed in the aftermath of a nuclear explosion; a low-density, high-temperature region left by the expanding shock is forced upwards by the pressure gradient caused by gravity; as it rises, the region expands and curls downward, forming a distinctive mushroom shape.

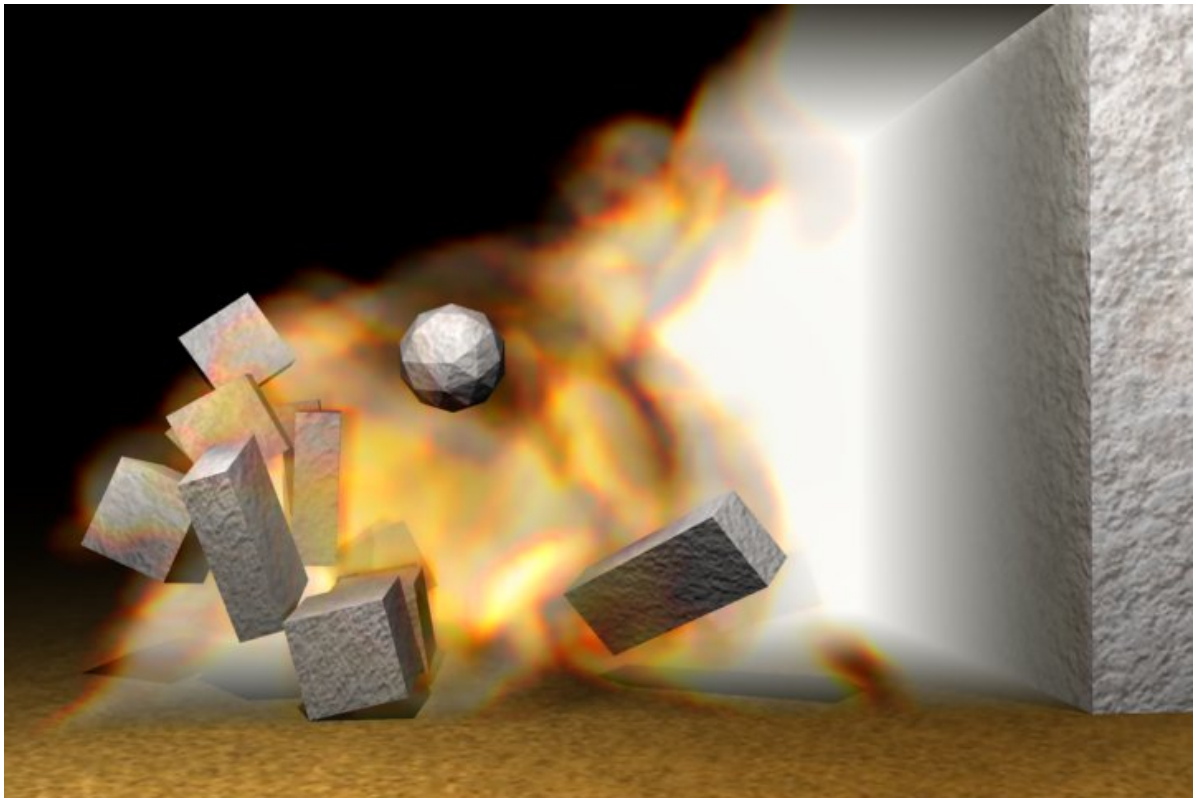


Figure 3.8: A stack of rigid bodies knocked over by a shock

Figure 3.8 demonstrates the method's ability to interact with moving boundary conditions; the stack of rigid bodies in this scene are bidirectionally coupled to the fluid. A traveling shock topples them, reflects off a nearby wall, and rebounds on the objects, throwing them away. The bodies' force upon the fluid creates vorticial patterns in the gas.

Figure 3.9 shows a 2D slice of a 3D simulation of a projectile traveling faster than the speed of sound. The bow shock ahead of the body is typical of this type of rounded object and the rarefaction region behind the projectile creates a twisting trail of turbulence.

Figure 3.1 and Figure 3.2 are similar; in each, a cylindrical tower of 600 bricks is toppled by an explosion from within. Figure 3.1 has no cap; the explosion forces nearly all of the air out of the cylinder as it bursts out of the top. The low-pressure area formed inside the cylinder causes it to collapse in upon itself while the force of the explosion

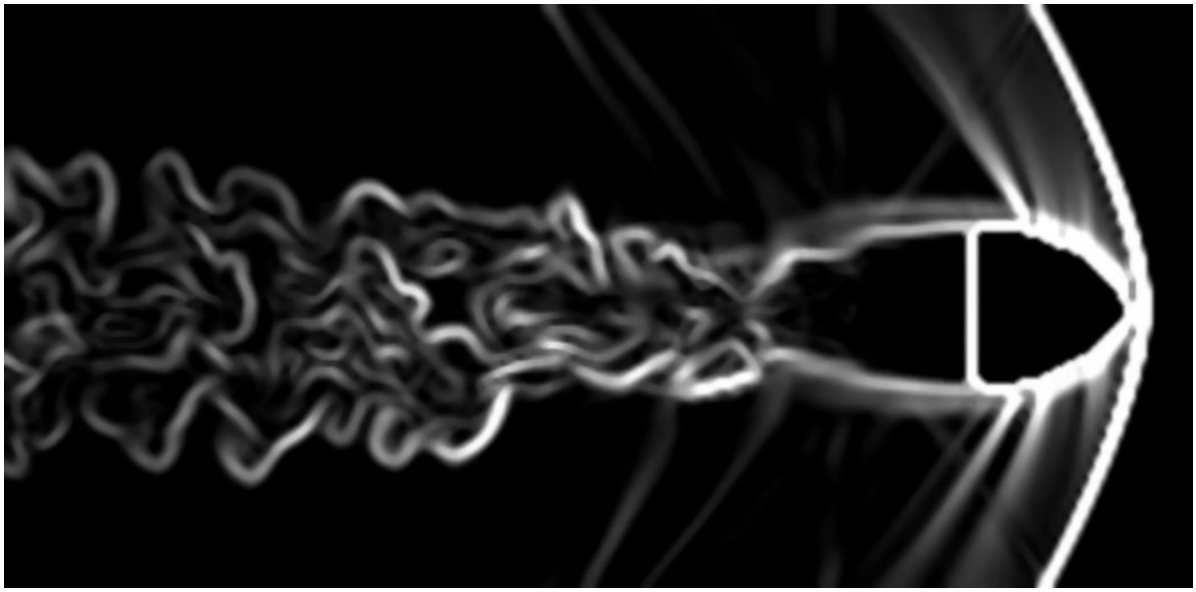


Figure 3.9: A bow shock and turbulence formed by the passage of a supersonic bullet

venting from the top of the structure send bricks flying. Figure 3.2 has a very heavy cap atop it; the explosive force cannot escape so easily and is partially reflected back into the structure, forcing a hole in the base and blowing out bricks near the top.

Figure 3.10 shows an explosion occurring in an enclosed area; the force of the explosion forces air through the small openings in the chamber and creates high-density, turbulent tendrils.

Figure 3.11 shows a series of frames from a simulation where a "house" made of 480 concrete bricks is struck by a powerful shock, causing the bricks to fly in all directions. The bricks shape and reflect the shock as it propagates through the scene.

Figure 3.13 is a visual recreation of the first moments of the detonation of the first nuclear bomb 'Trinity'. The glossy "bubble" around the explosion is the expanding shockfront; the heat at the interface is such that light traveling through the region is dramatically refracted. Inside the shock, dust and flame are rising with a bright glow.

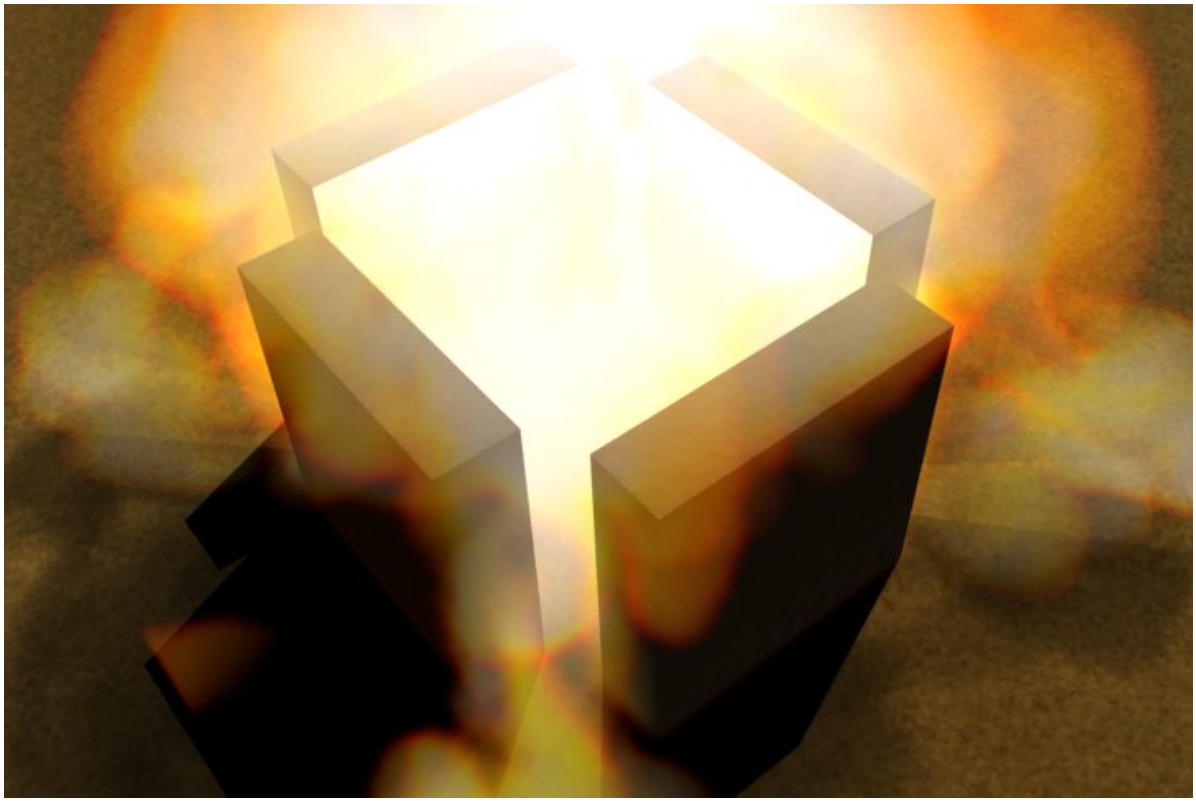
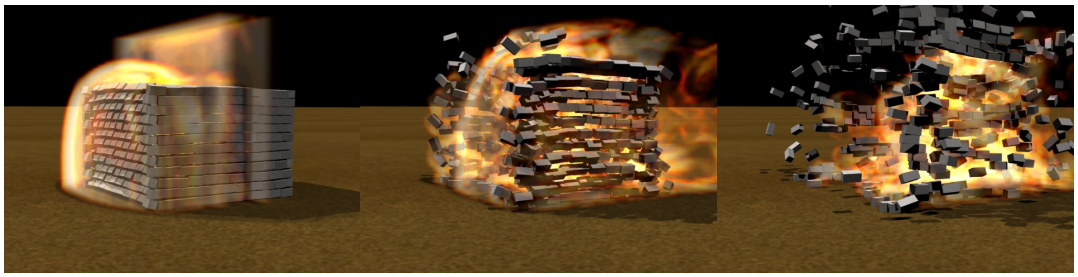


Figure 3.10: An explosion in a confined space

3.5.2 Rendering

The 3D demonstrations were modeled in Blender [Blender Foundation, 2007] and rendered with the V-Ray raytracer [Chaos Group, 2010]; the visualization of fluid effects in 3D were handled by a Monte Carlo volume raytracer plug-in for V-Ray. Atmospheric scattering was not used; these renders use ρ as advected by the fluid for the emissive and absorbing factors for the volume tracer, with color determined by a linear mapping of ρ into a blackbody colormap. The 2D demonstrations were rendered with a simple custom 2D plotting tool; those using a monochrome colormap demonstrate our method's preservation of sharp shock features through a *schlieren* plot — namely, $\sqrt{|\nabla\rho|}$. The term schlieren refers to a particular type of image formed by the passage of light through inhomogeneous media that causes shadows to appear in areas of high inhomogeneity; see [Settles, 2001] for more detail.



(a) (b) (c)

Figure 3.11: Rigid body-fluid interaction

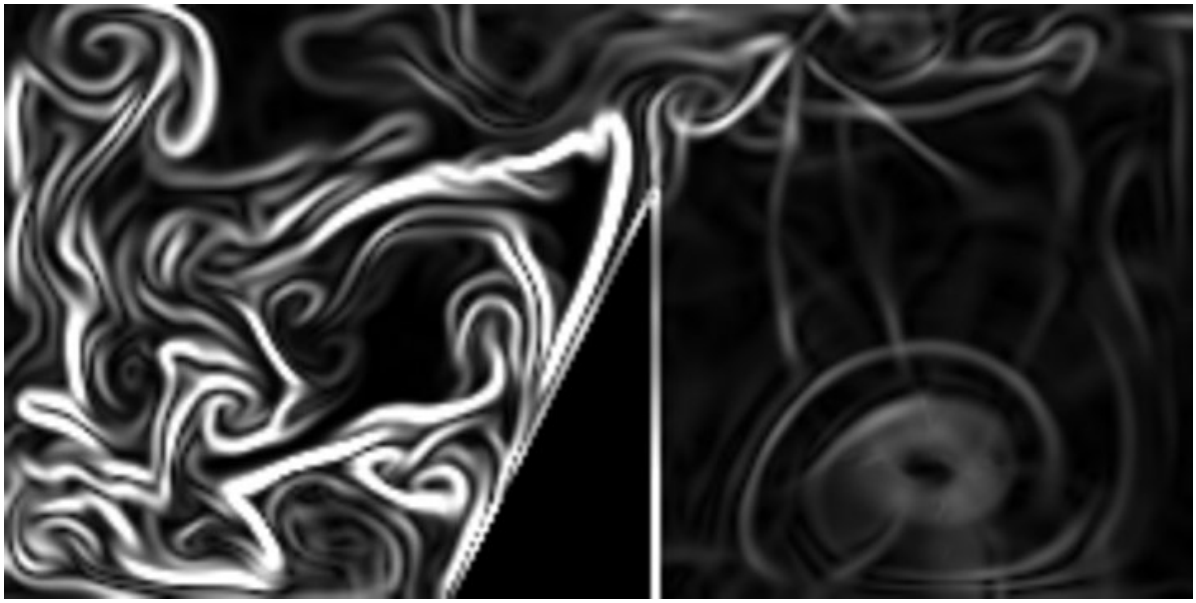


Figure 3.12: Vortex shedding from a shock interacting with wedge

3.5.3 Timings

Performance data is listed in Table. 3.1; these timings were collected on a 2GHz Core 2 laptop. Memory usage is linear in the number of grid cells — each demonstration fits within 500MB of memory. These timings are all for a single thread of computation; parallelization results were discussed in Section 3.4.

Direct comparisons with previous works are difficult to produce because little or no timing information is available for these papers. Figure 3.14(a) (included, with permission, from [Yngve et al., 2000]) shows a 2D slice of a 101^3 simulation of a shockwave

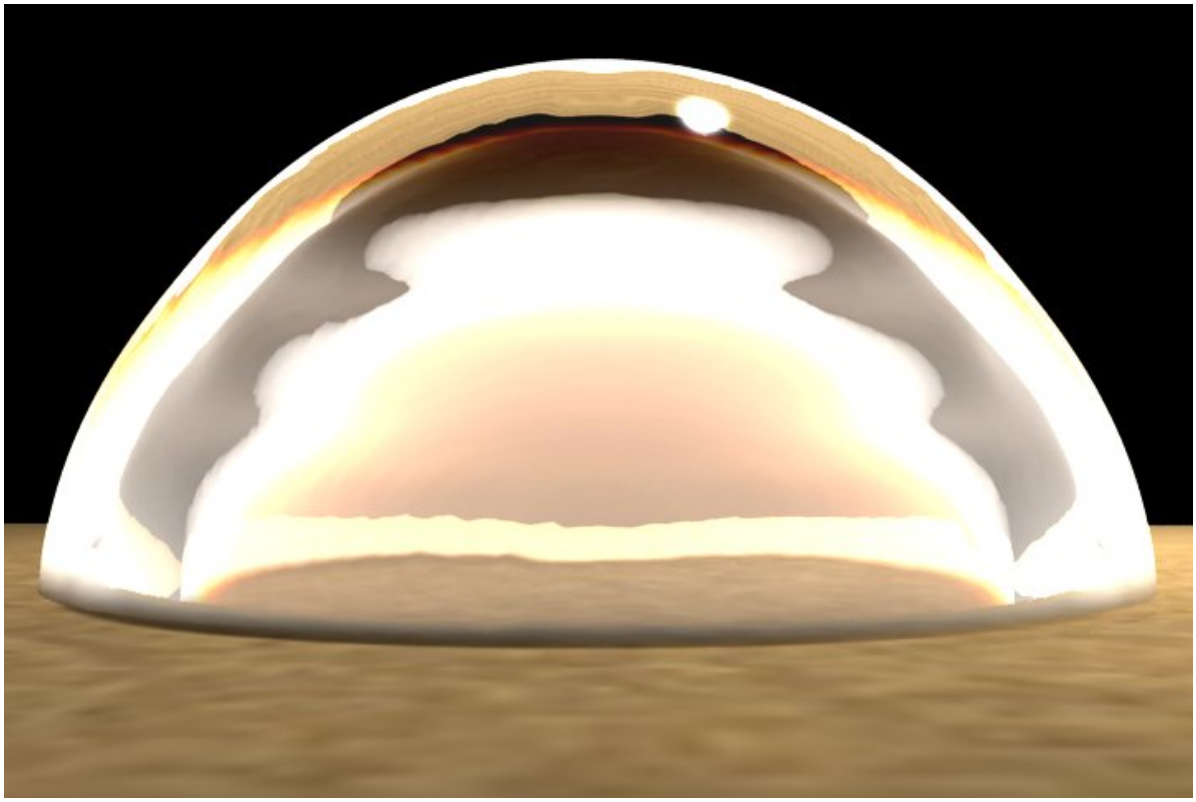


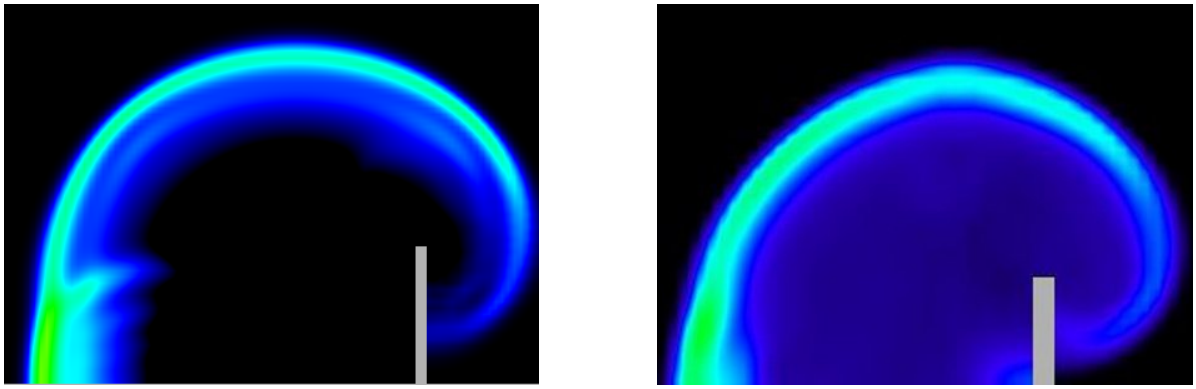
Figure 3.13: The initial moments of the "Trinity test" — the first atomic bomb

interacting with a stationary wall; they reported a simulation time of 'overnight'. I reproduced this simulation with the method presented here; for a 101^3 grid, I recorded a total simulation time of *15 minutes*. Conservatively estimating that a single core of our hardware is nearly 7 times faster and that 'overnight' is about 10 hours, the serial version of our method is at least 6 times faster than theirs at equivalent resolutions, and our simulation contains more visual detail. To demonstrate the ability of this method to produce detailed results at coarse resolutions, I performed the same simulation on a 60^3 grid (see Figure 3.14(b); this took less than *2 minutes* (roughly 45x faster) and the generated results exhibit more detail than the results computed on a 101^3 grid using [Yngve et al., 2000] (shown in Figure 3.14(a)).

Table 3.1: Demonstrative timings of the simulation method

Scene	resolution	sim. fps	avg. Δt	sim. time
Blast chamber	120×80×120	1.56	1.4e-4 s	16.25 min
Rigids w/ refl.	60×60×100	0.779	2.5e-4 s	29.93 min
Tower (top)	60×80×60	1.14	7.2e-4 s	30.21 min
Trinity	200×75×200	0.102	2.9e-5 s	32.88 min
Tower (no top)	60×100×60	0.939	6.8e-4 s	51.74 min
Mush. cloud	120×100×120	0.243	2.4e-2 s	57.74 min
House	100×100×100	0.310	4.6e-5 s	58.35 min
Projectile	250×100×100	0.191	1.0e-5 s	191.5 min

Timings showing grid resolution, simulation frames per second, average simulation timestep, and the total computation time needed for the entire simulation run.



(a) Results from Yngve et al.; simulation on a 101^3 grid

(b) My method on a 60^3 grid

Figure 3.14: Comparison of pressure in blast reflection/diffraction scenario

3.5.4 Parallelization

Figures 3.15, 3.16, 3.17 demonstrate the scaling for the various parallel schemes described in Section 3.4. These timings were collected on system with four Intel Xeon X7350 quad-core processors running at 2.93GHz; the system runs Microsoft Windows Server 2003 (64-bit) and has 16GB of physical memory.

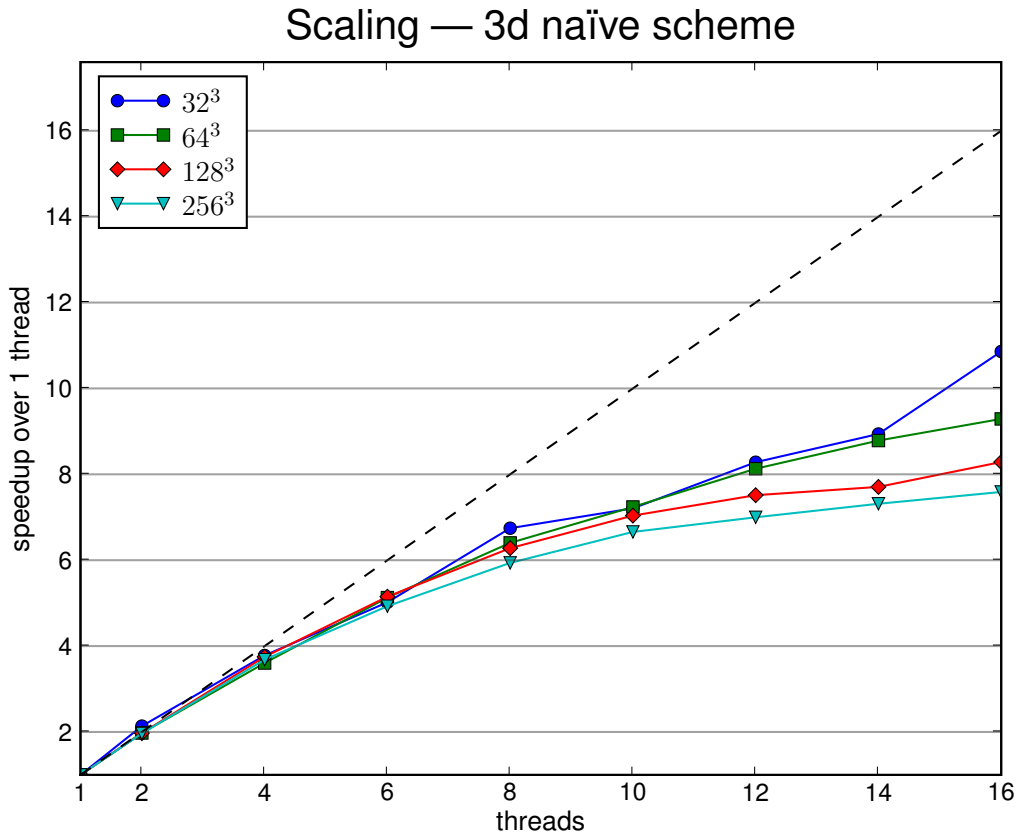


Figure 3.15: A naïve parallelization scheme scales poorly with the number of threads

3.6 Conclusion

I have presented a method for efficient simulations of supersonic flows in compressible, inviscid fluids that is based on the finite volume method. I have demonstrated the ability of this method to capture the behavior of shocks and to handle complex, bidirectional object-shock interactions stably. I have also demonstrated an effective parallelization scheme based on architectural considerations that achieves near-linear scaling on modern multi-core architectures.

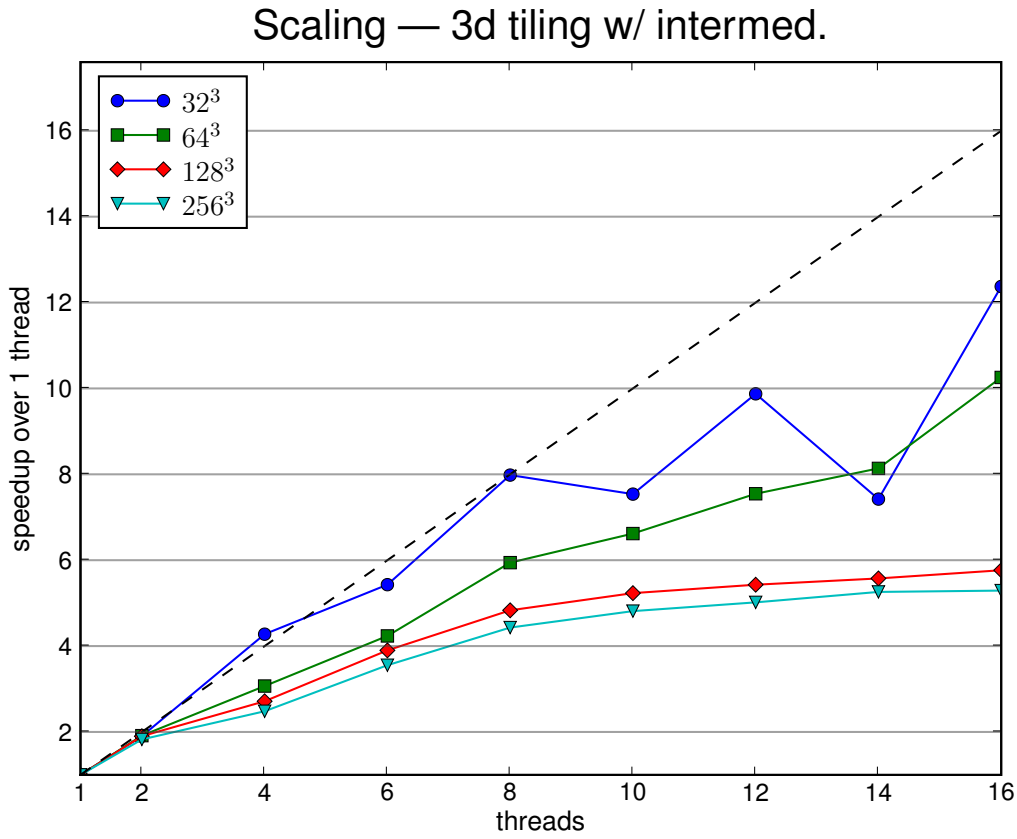


Figure 3.16: Scaling of the initial version of the tiled parallelization. Memoization of Riemann solutions leads to bandwidth saturation for large numbers of processors.

3.6.1 Limitations

Hyperbolic systems of equations (i.e. the compressible, inviscid Euler equations simulated here) are subject to the CFL condition as a requirement for convergence *and* stability. The unconditionally stable solvers popular for incompressible fluid dynamics are subject to the CFL condition for convergence, but not stability — indeed, the convention seems to take the CFL condition as a "guideline" and use CFL numbers upwards of 5.

My technique performs well at simulating truly hyperbolic phenomena such as compressible, inviscid fluid dynamics, but cannot handle nearly incompressible phenomena (e.g. liquids) as efficiently as those simulations currently used in computer graphics.

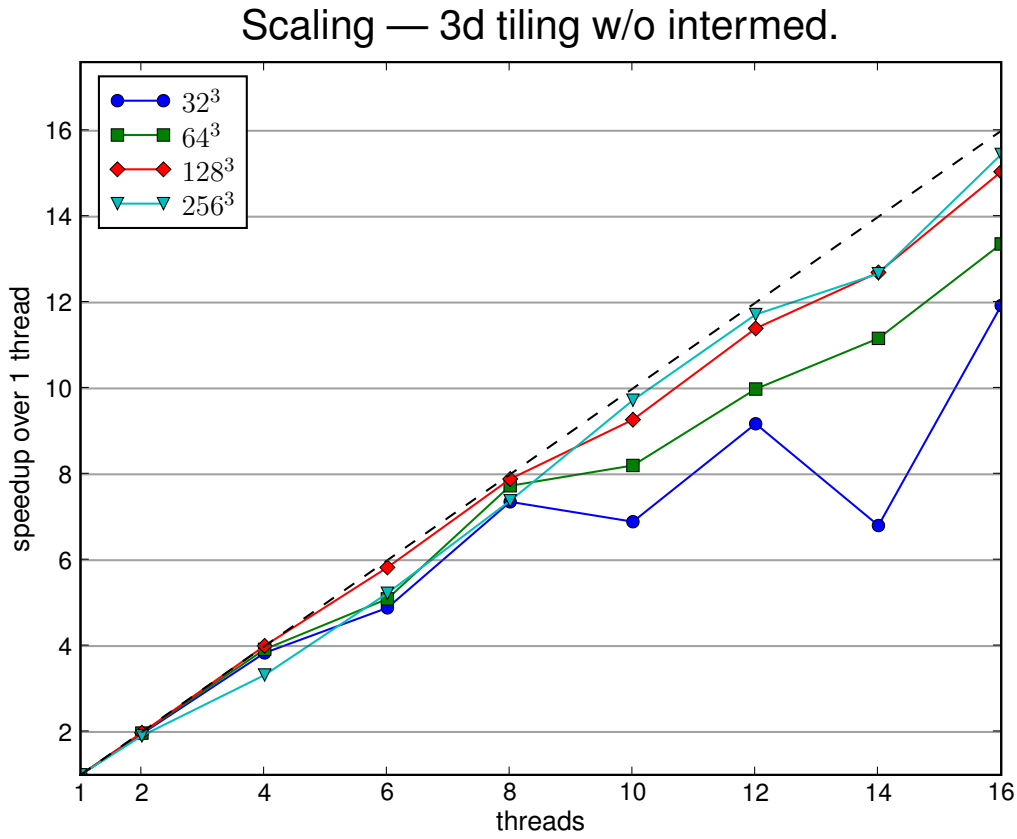


Figure 3.17: Scaling of the revised version of the tiled parallelization. The reduction in bandwidth requirements greatly improves scaling.

This fundamental limitation is due to the choice of equations — the actual propagation of acoustic waves so important to compressible fluids has a negligible effect on incompressible fluids.

3.6.2 Future work

There are a number of promising areas for future work. Many natural phenomena give rise to shocks — of particular interest to graphics are hydraulic jumps in the Saint-Venant (or shallow water) equations.

The tiled parallelization scheme I employ scales very well compared to a naïve parallel decomposition, but there is potential for further improvement with a nuanced investi-

gation of further cache effects, operating system scaling, and processor layout. I also would like to investigate the extension of this method to new parallel architectures, such as Intel's Larrabee, IBM's Cell, next-generation graphics cards leveraging OpenCL and CUDA.

In the next section, I present my technique for generating visual traffic flows using a compressible fluid-like model of vehicle motion.

Chapter 4

Visual Continuum Traffic Simulation

4.1 Introduction

As the world's population continues to grow, traffic management is becoming one of the growing challenges for many cities and towns across the globe. Increasing attention has been devoted to modeling, simulation, and visualization of traffic flows to investigate causes of traffic congestion and accidents, to study the effectiveness of roadside hardware, signs and other barriers, to formulate improved policies and guidelines related to traffic regulation, and to assist urban development and design of highway and road systems. In addition, with increasing volumes of traffic data and related software tools capable of visualizing urban scenes (such as Google Maps and Virtual Earth), there is a growing need to add realistic street traffic in virtual worlds for virtual tourism, feature animation, special effects, traffic monitoring, and many other applications.

There exists a vast amount of literature on modeling and simulation of traffic flows, with existing traffic simulation techniques generally focusing on either agent-based *microscopic* or continuum-based *macroscopic* models. However, little attention has been paid to the possibility of extending macroscopic models to produce detailed 3D animations and visualization of traffic flows. In this chapter, I present a fast method for efficient simulations of large-scale, real-world networks of traffic using continuum dynamics that maintains discrete vehicle information to display each vehicle. The con-

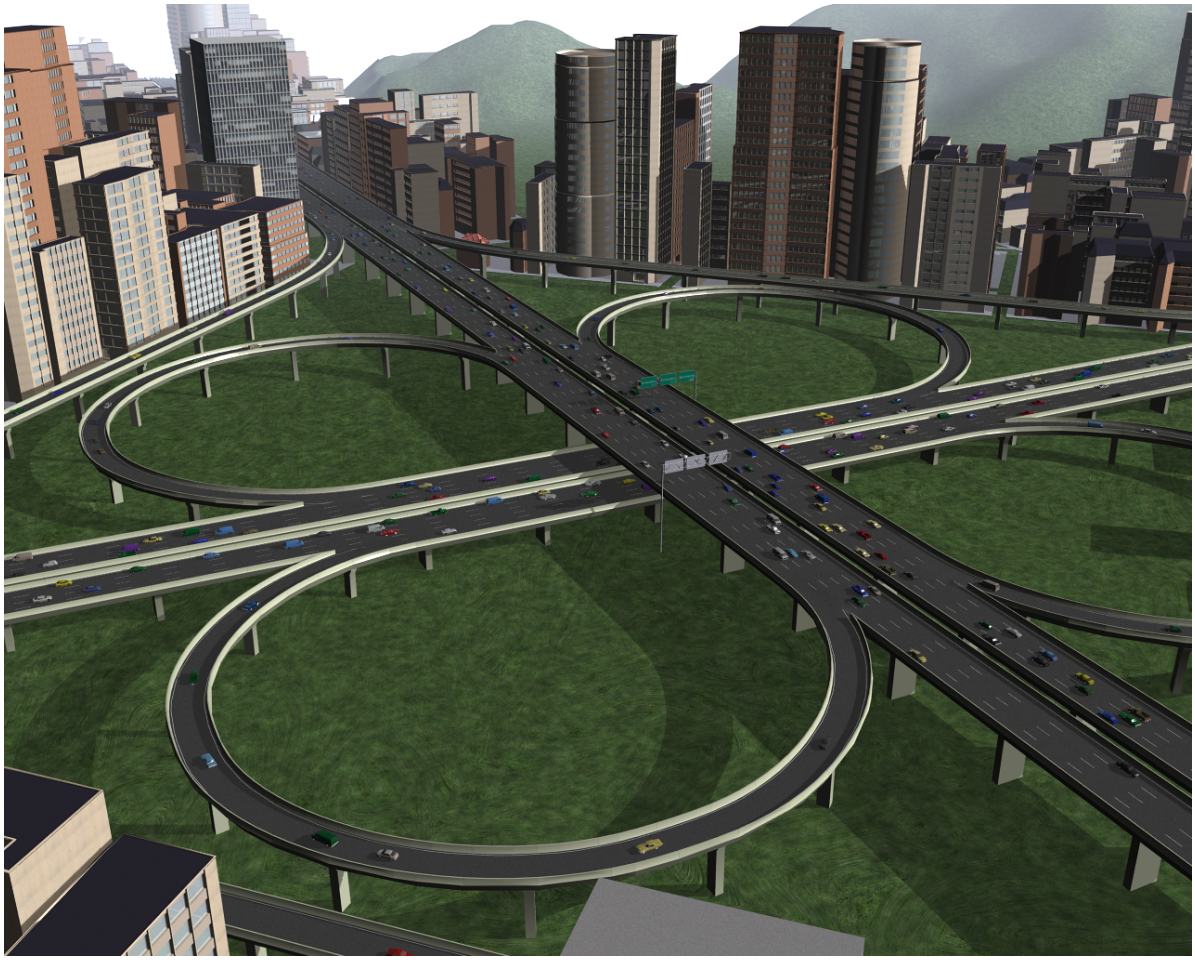


Figure 4.1: A bird's-eye view of animated traffic

tinuum approach describes realistic behavior and is efficient — it is able to describe the movement of many vehicles with a single computational cell — while individual vehicle information facilitates visual representation and allows per-vehicle information to influence the large-scale simulation. This technique produces detailed, interactive animations of enormous traffic flows on a wide variety of road types, including urban streets, multi-lane highways, and winding rural roads at *more than* $100x$ *faster than real time*.

My approach extends a per-lane flow model to a continuum, multi-lane traffic flow model by introducing a novel model of lane changes and using a discrete visual representation for each vehicle. I compare my technique's efficiency with that of agent-based

methods, and demonstrate how this simulation technique is able to effectively utilize the processing power of many-core shared memory architectures for scalable simulation. Figure 4.1 shows a snapshot image of the highway traffic in an urban scene simulated by this continuum model.

4.2 Related work

While much recent work on physical simulation in graphics has focused on natural phenomena such as fluids and deformable objects, there is growing interest in the efficient simulation of man-made phenomena; since the influential 'boids' model of [Reynolds, 1987] proposed a framework for the simulation of multiple intelligent agents, a large body of work has since appeared in the area of crowd dynamics, covering important sub-problems ranging from motion planning and collision avoidance techniques to behavioral models (see a recent survey [Pettré et al., 2008] for more detail). There has been comparatively little investigation in computer graphics on vehicle and traffic simulation; the method this chapter was published as [Sewall et al., 2010b], and interest on synthesizing vehicle motion using algorithmic robotics techniques [Go et al., 2005, Sewall et al., 2010a] has been growing.

The growing ubiquity of vehicle traffic in everyday life has generated considerable interest in models of traffic behavior, and in the last 60 years, a large body of research in the area has appeared. The problem of traffic simulation — given a road network, a behavior model, and initial car states, how the traffic in the system evolves — has been extensively studied outside of the graphics community. Most of the existing methods are typically designed to explore specific phenomena, such as jams and unsteady, "stop-and-go patterns" of traffic, or to evaluate network configurations to aid in real-world traffic engineering.

Most work on traffic simulation and modeling is found in civil engineering and urban

planning, and numerous techniques for effective simulation of traffic flows have been developed to address issues associated with road design, road construction and evaluation, as well as intersection signal planning, training, and forensics. There are three broad classes of traffic simulation techniques: the agent-based *microscopic* and continuum-based *macroscopic* techniques as mentioned in Sec. 5.1 as well as kinetic *mesoscopic* techniques based on Boltzmann-like statistical mechanics.

[Gerlough, 1955] introduced agent-based simulation; later work by [Newell, 1961], [Algers et al., 1997], and [Helbing, 2001] incorporated further features of traffic flows into the 'car-following' model. [Nagel and Schreckenberg, 1992] describe how to handle agent-based traffic simulation through a cellular automata.

The equations for macroscopic traffic flow were independently discovered by [Lighthill and Whitham, 1955] and [Richards, 1956] based on observed similarities between one-dimensional compressible gas dynamics and the way traffic flows along a lane. The resulting so-called 'LWR' equation is a scalar, nonlinear partial differential equation describing the motion of traffic in terms of density, i.e. 'cars per lane'. Because the LWR equation is a scalar equation for density, the velocity of traffic at any point is given by an equation of state; all traffic at a given density necessarily has the same velocity.

To achieve a more complete model of traffic where velocity does not depend wholly on density, [Payne, 1971] and [Whitham, 1974] proposed a 2-variable model, also known as the 'Payne-Whitham' or 'PW' model, based more directly on the Euler equations of gas dynamics. This was later shown to have incorrect behavior by [Daganzo, 1995], who pointed out that the isotropy of gas dynamics was not compatible with traffic dynamics.

More recently, [Aw and Rascle, 2000] and [Zhang, 2002] independently proposed a 2-variable model of traffic flow with correct anisotropic behavior. [Lebacque et al., 2007] noted that the two models could easily be unified through a change of variables, and dubbed the system the 'Aw-Rascle-Zhang' ('AWR') system of equations.

Mesoscopic methods were was pioneered by [Prigogine and Andrews, 1960] and im-

proved upon by [Nelson et al., 1997], [Shvetsov and Helbing, 1999] and others.

4.3 Method

In this section, I describe my data structures and method for the simulation of traffic flow in detail.

4.3.1 Overview

I simulate traffic on a network of roads; each road has one or more lanes and is connected to other roads through intersections and interchanges. See Section 4.3.2 for details on how I represent these networks, and how they may be synthesized or adapted from real-world data.

The simulation describes the flow of traffic through a system of nonlinear *hyperbolic conservation laws* that represent traffic as a continuum along lanes. Many systems of equations have been developed to describe the flow of traffic with varying degrees of completeness. I use the model recently proposed by [Aw and Rascle, 2000] and [Zhang, 2002], which I refer to as the Aw-Rascle-Zhang (ARZ) model, following [Lebacque et al., 2007]. This equation is described in Section 4.3.3.3.

To obtain a numerical solution to the ARZ equations, I use a Finite Volume Method (FVM) spatial discretization combined with a Riemann solver to determine the fluxes between adjacent computational cells. See Section 4.3.3.1 for details on the FVM and how it applies to conservations laws like the ARZ equations, and see Section 4.3.3.4 for a description of the Riemann solver.

The ARZ equations describe the motion of traffic along individual lanes; to handle merges and lane changes, my solution combines continuum-level dynamics with the discrete car information. I describe this representation in detail in Section 4.3.5.

The continuum approach to traffic simulation has numerous advantages in efficiency

and robustness; however, to display the motion of traffic for visualization and animation purposes, I introduce "car particles", or *carticles* that represent the individual cars in the flow of traffic. These carticles are moved along by the underlying continuum flow, and can play decision-making roles in parts of the simulation — particularly in regards to lane changes and merges. Section 4.3.4 describes these in detail.

Finally, while this method is efficient enough to simulate heavy volumes of traffic on large networks on a single processor, it is simple to parallelize and scales well on multi-core machines so as to make massive-scale simulations possible. I describe how this method can be implemented to further benefit from parallel systems in Appendix D.

4.3.1.1 Basic solution procedure

This technique simulates traffic by taking repeated time steps of a solver that computes the dynamics of traffic motion at instants in time. For a given timestep, the solver operates as follows:

1. Solve the ARZ equations along each lane, taking into account boundary information.
2. Initiate and advance lane changes
3. Advance carticles according to the solution in each lane
4. Apply relaxation of relative velocity
5. Update network state

These steps will be explained in greater detail below.

4.3.2 Representation of road networks

The simulation operates in the domain of road networks, and the fidelity and realism of the results it is capable of generating clearly depends on the quality and detail of

the network. In this section, I present a robust data structure for representing a road network suitable for simulation.

4.3.2.1 Features of road networks

Road networks can be arbitrarily complicated; it takes only a moment to consider that in addition to information about lanes and intersections, one could also have descriptions of road quality and conditions, information about driveways and parking spaces along the road, and a host of other types of data.

This method currently handles multi-lane road segments with variable speed limits, and as such, my data structure is designed to efficiently store and answer queries about such information. However, other types of information can be easily included in the structure.

Roads A road network could be described as a collection of roads and intersections — roads have some spatial description of the path they take, a number of lanes, and they stop and end at other roads. However, such a naïve description fails to properly capture many features desired in a road network; how should one describe a highway interchange, where several lanes split off from roads, take a curving path, and join another road? For this reason, the data structure treats roads simply as descriptions of spatial information; here, each road provides a sequence of connected lines that describes its path in space.

Lanes I have chosen the lane to be the atomic data type in the data structure. This is motivated by their relationship to roads and other lanes — in that a single lane may belong to many roads and be adjacent to many different lanes along its length — but also by simulation methodology. Since I solve the ARZ equations for the flow of traffic along each lane, it is desirable from an efficiency standpoint to have lanes as long and unbroken as possible, to minimize the need to handle special-case boundary conditions.

Each lane is parametrized by its length in space to the unit interval, and properties

of the lane — such as speedlimit, the road to which it belongs and obtains its spatial description from, and the other lanes it may be adjacent to — are mapped to this parametric interval.

This parametrization of lane properties is motivated by the need for various queries during simulation. While advancing the solution along each lane, for example, it use essential to know what the speed limit is in the current cell, and if it differs from that in the next. Similarly, when merging traffic, one needs to know which lanes (if any) are to the left and to the right of the current position along a lane.

4.3.3 Numerical traffic simulation with gas-like laws

This technique simulates the flow of traffic along lanes with a numerical discretization of a *hyperbolic conservation law*. This is a class of partial differential equations commonly associated with physical laws and with gas dynamics in particular. The basis for the traffic flow simulation used here, the so-called Aw-Rascle-Zhang (ARZ) model ([Aw and Rascle, 2000, Zhang, 2002]), is one such law that is closely related to the hyperbolic systems of equations that describe gas dynamics.

4.3.3.1 Conservation laws & the finite volume method

The ARZ model is a conservation law; it fits the general model

$$\mathbf{q}_t + \mathbf{f}(\mathbf{q})_x = 0 \tag{4.1}$$

where subscripts denote differentiation, $\mathbf{q}$ is a vector-valued quantity of unknowns, and $\mathbf{f}(\mathbf{q})$ is a vector-valued function of the unknowns. The choice of $\mathbf{f}$ (known as the *flux function*) uniquely characterizes the dynamics of the system. Solutions to Equation (4.1) may be readily discretized with the Finite Volume Method (FVM) of numerical discretization. Taking the integral of Equation (4.1) in space over some arbitrary interval

$x \in [a, b]$:

$$\begin{aligned} \int_a^b \mathbf{q}_t \, \mathrm{d}x + \int_a^b \mathbf{f}(\mathbf{q})_x \, \mathrm{d}x &= 0 \\ = \frac{\mathrm{d}}{\mathrm{d}t} \int_a^b \mathbf{q} \, \mathrm{d}x + \mathbf{f}(\mathbf{q}(b)) - \mathbf{f}(\mathbf{q}(a)) & \end{aligned} \tag{4.2}$$

Now divide Equation (4.2) by $(b - a) = \Delta x$ and discretize $\mathbf{q}$ into quantities $\mathbf{Q}_i$ representing the *average* of $\mathbf{q}$ over $[a, b]$:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} \frac{1}{b-a} \int_a^b \mathbf{q} \, \mathrm{d}x + \frac{1}{b-a} \left[\mathbf{f}(\mathbf{q}(b)) - \mathbf{f}(\mathbf{q}(a)) \right] &= 0 \\ = \frac{\mathrm{d}}{\mathrm{d}t} \mathbf{Q}_i + \frac{1}{\Delta x} \left[\mathbf{f}(\mathbf{q}(b)) - \mathbf{f}(\mathbf{q}(a)) \right] & \end{aligned} \tag{4.3}$$

Equation (4.3) can be discretized in time with a simple Euler scheme and manipulated to obtain the FVM update scheme:

$$\begin{aligned} \frac{\mathbf{Q}_i^{n+1} - \mathbf{Q}_i^n}{\Delta t} + \frac{1}{\Delta x} \left[\mathbf{f}(\mathbf{q}(b)) - \mathbf{f}(\mathbf{q}(a)) \right] &= 0 \\ \mathbf{Q}_i^{n+1} = \mathbf{Q}_i^n - \frac{\Delta t}{\Delta x} \left[\mathbf{f}(\mathbf{q}(b)) - \mathbf{f}(\mathbf{q}(a)) \right] & \end{aligned} \tag{4.4}$$

Here $\mathbf{Q}_i^n = \mathbf{Q}_i(t^n)$ and $\Delta t = t^{n+1} - t^n$. More information on hyperbolic conservation laws and the FVM can be found in [Leveque, 2002].

Flux calculations Equation (4.4) is a straightforward update scheme; what remains to be computed are the quantities $\mathbf{f}(\mathbf{q}(b))$ and $\mathbf{f}(\mathbf{q}(a))$ — that is, the flux that occurs at the boundaries between cells. This can be difficult for nonlinear $\mathbf{f}$ (such as that found in the ARZ system of equations) and accounts for the bulk of the computation in the numerical scheme. The problem of determining these fluxes is termed the *Riemann problem*, and I discuss how its solution for the ARZ model in Section 4.3.3.4.

Computation of Δx for each lane Each lane j in simulation is divided into a number of discrete cells of equal length Δx_j. Δx_j varies only slightly from lane to lane; when preparing the simulation, the user suggests a "target" Δx that all lanes should have, and for each lane j of length L_j, the number of cells in the lane N_j and the related cell length Δx_j are determined as follows:

$$N_j = \left\lfloor \frac{L_j}{\Delta x} \right\rfloor, \quad \Delta x_j = \frac{L_j}{N_j} \tag{4.5}$$

This ensures that all cells in a given lane have the same length. So long as all lanes are at least Δx in length, it is ensured that $\Delta x_j > \frac{\Delta x}{2} \, \forall j$. In general, the guide Δx should be greater than the length of the longest vehicle in the simulation (see Section 4.3.4).

Timestep restrictions Δt in Equation (4.4) must be chosen to satisfy the Courant-Friedrichs-Lewy (CFL) condition for the integration to be stable. According to this condition, (described in more detail in [Courant et al., 1928]), the follow must hold:

$$\Delta t < \min_j \left(\frac{\Delta x_j}{\lambda_{\max j}} \right) \tag{4.6}$$

With the minimum taken over all lanes j. $\lambda_{\max j}$ is the *maximum speed* in lane j at the current timelevel; these speeds are determined while solving the Riemann problem at each interface — for the system of equations used in this technique, Equations (4.13) and (4.14). This has ramifications on how to choose to apply Equation (4.4); since the same Δt must be used to integrate each cell, we must compute *all* speeds λ before integrating any cells.

It should be noted that obeying the CFL condition does not mandate that simulation timesteps be smaller than what would be desired for display. For example: in a network with a maximum speed of 100 km/s, $\lambda_{\max}$ will not exceed $27.\bar{7}$ m/s; this gives $\Delta t < 0.0036 s/m \Delta x$. In a simulation, the user are free to choose any Δx — generally, this is

a multiple of car lengths. Even for a the smallest Δx I have used in my experiments — 9m ($2 \times 4.5m$), this gives $\Delta t < 0.324s$, which is on par with the frame rate desired for most conventional visualization techniques.

4.3.3.2 Numerical update procedure

Given cell values $\mathbf{Q}^n$ at time t_n for all lanes j, I compute $\mathbf{Q}^{n+1}$ as follows:

1. At each interface between cells, compute the speeds λ_i and fluxes F_i^n by solving the Riemann problem at that interface (described in Section 4.3.3.4)

2. Find the speed with largest magnitude and compute timestep length Δt as described in Section 4.3.3.1.

3. For each cell i, advance to next time $\mathbf{Q}^{n+1}$ using Equation (4.4) and the fluxes from step 1.

4.3.3.3 Aw-Rascle-Zhang model

The ARZ model ([Aw and Rascle, 2000, Zhang, 2002]) can be written as a conservation law of the form

$$\mathbf{q}_t + \mathbf{f}(\mathbf{q})_x = 0, \quad \mathbf{q} = \begin{bmatrix} \rho \\ y \end{bmatrix}, \ \mathbf{f}(\mathbf{q}) = \begin{bmatrix} \rho u \\ y u \end{bmatrix} \tag{4.7}$$

Here ρ is the density of traffic, i.e. "cars per car length", u is the velocity of traffic, and y the "relative flow" of traffic.

y, ρ, and u are related by the equation:

$$y(\rho, u) = \rho(u - u_{\text{eq}}(\rho)) \tag{4.8}$$

where $u_{\text{eq}}(\rho)$ is the "equilibrium velocity" for ρ. There is a single criterion on u_{eq} that

must be satisfied (see Equation B.31) on this function; the following is suitable for most types of traffic:

$$u_{\text{eq}}(\rho) = u_{\max}(1 - \rho^{\gamma}) \tag{4.9}$$

where $\gamma > 0$. The first derivative of Equation (4.9) is significant as well:

$$u'_{\text{eq}}(\rho) = -u_{\max}\gamma\rho^{\gamma-1} \tag{4.10}$$

In the above equations, $u_{\max}$ is the speed limit of the road. Using Equations (4.8) and (4.9), we can write u in terms of y and ρ:

$$u(\rho, y) = \frac{y}{\rho} + u_{\text{eq}}(\rho) \tag{4.11}$$

In what follows, I shall interchangeably use $u_{\text{eq}}(\rho)$ and u_{eq} as well as $u(\rho, y)$ and u.

4.3.3.4 Basic Riemann problem for the ARZ model

To compute fluxes such as $\mathbf{f}(\mathbf{q}(b))$ and $\mathbf{f}(\mathbf{q}(a))$ in Equation (4.4), one must be able to determine the value of $\mathbf{q}$ between the piecewise-constant states in adjacent cells.

Given initial constant states $\mathbf{q}_l$ for $x < 0$ and $\mathbf{q}_r$ for $x > 0$ (with components which I term ρ_l, y_l, and u_l and ρ_r, y_r, and u_r, respectively), what happens for $t > 0$? In the case of the ARZ model, there are several distinct possibilities. Depending on the relative values of $\mathbf{q}_l$ and $\mathbf{q}_r$, we expect the solution to consist of two or more distinct "regions" of self-similar solutions traveling with varying speeds. The following discussion expands upon the analysis in [Aw and Rascle, 2000].

Waves and speeds The eigenstructure of the Jacobian of the flux function defined in Equation (4.7) is the key to determining the speeds in the system (for Equation (4.6))

and the structure for the Riemann problem; said Jacobian is

$$\mathbf{J_f} = \begin{bmatrix} u_{\text{eq}} + \rho u'_{\text{eq}} & 1 \\ y u'_{\text{eq}} - \frac{y^2}{\rho^2} & \frac{2y}{\rho} + u_{\text{eq}} \end{bmatrix} \tag{4.12}$$

The eigenvalues of Equation (4.12) are:

$$\lambda_0 = u + \rho u'_{\text{eq}} \tag{4.13}$$

$$\lambda_1 = u \tag{4.14}$$

with corresponding eigenvectors

$$\mathbf{r}_0 = \begin{bmatrix} 1 \\ \frac{y}{\rho} \end{bmatrix} \tag{4.15}$$

and

$$\mathbf{r}_1 = \begin{bmatrix} 1 \\ \frac{y}{\rho} - \rho u'_{\text{eq}} \end{bmatrix} \tag{4.16}$$

Field classification One can regard the pairs of eigenvalues and eigenvectors as distinct 'families' of solution fields — solutions associated with λ_0 and $\mathbf{r}_0$ are referred to as the '0-family' of solutions, and those associated with λ_1 and $\mathbf{r}_1$ as the '1-family' of solutions.

While the ARZ equations (Equation (4.7)) are clearly nonlinear, different familiar may exhibit different characteristics — some linear, some nonlinear. By classifying these fields as either *genuinely nonlinear* or *linearly degenerate*, it is possible to obtain information about the types of solutions to expect. For more detail on how this classification is performed, see Appendix B. For this system, the first family of solutions (those associated with λ_0 and $\mathbf{r}_0$) is genuinely nonlinear — the related waves deform

with propagation. The second family of solutions (λ_0 and $\mathbf{r}_0$) is linearly degenerate and behaves as a linear system might.

Riemann invariants In the same spirit of the field classifications discussed above, one can determine what quantities are preserved across each solution family; $\mathbf{q}_l$ and $\mathbf{q}_r$ are connected through an unknown intermediate state $\mathbf{q}_m$ and these invariants help determine this intermediate state. A quantity ω_i is a *Riemann invariant* for the i-family of solutions if it satisfies the following equation:

$$\nabla\omega_i \cdot \mathbf{r}_i = 0 \tag{4.17}$$

The invariant for the first wave (corresponding to λ_0 and $\mathbf{r}_0$) can be computed by substituting Equation (4.15) into Equation (4.17):

$$\omega_0 = \frac{y}{\rho} = u - u_{\text{eq}} \tag{4.18}$$

The 1-family is linearly degenerate, so Equation (4.18) is clearly satisfied by Equation (B.33):

$$\omega_1 = \lambda_1 = u \tag{4.19}$$

The 0-family invariant looks similar to Equation (4.8); intuitively, one can say that this 'equilibrium velocity' is conserved across waves from the first family. The 1-family invariant simply states that the velocity u does not change across waves of the second family.

Intermediate state To compute the left- and right-going fluctuations so that states can be integrated in time, the value of q_0 must be computed — that is to say, the value of $\mathbf{q}$ at $x = 0$ for $t > 0$, given $\mathbf{q}_l$ and $\mathbf{q}_r$. In general, $\mathbf{q}_0$ can be $\mathbf{q}_l$, $\mathbf{q}_r$, or the intermediate value $\mathbf{q}_m$. The left state $\mathbf{q}_l$ and the intermediate state $\mathbf{q}_m$ are separated by the line

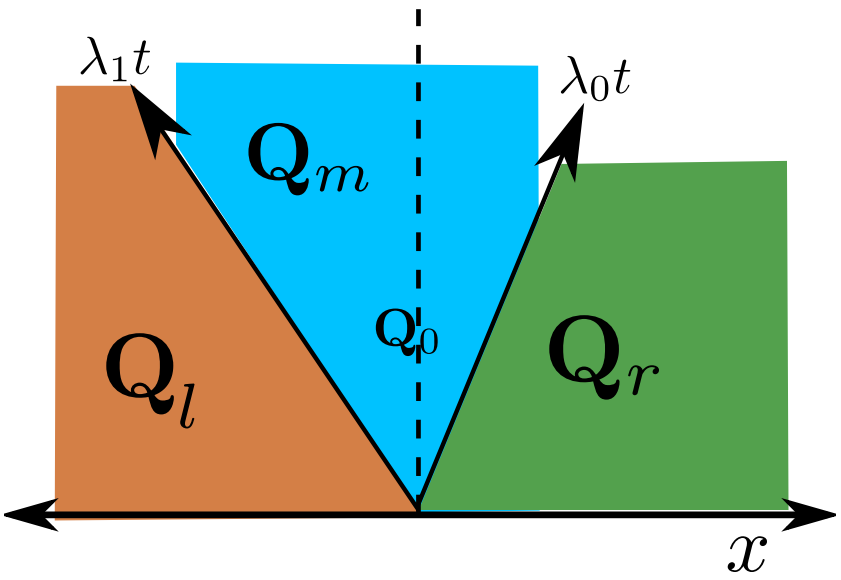


Figure 4.2: A schematic of a Riemann problem; the up-axis represents both time and $\mathbf{Q}$. Here, an intermediate state $\mathbf{Q}_m$ arising between $\mathbf{Q}_l$ and $\mathbf{Q}_r$. To compute the flux between these cells, $\mathbf{Q}_0$ must be determined.

$x = \lambda_0 t$, and $\mathbf{q}_m$ and $\mathbf{q}_r$ are separated by the line $x = \lambda_1 t$. For the ARZ model under consideration, $\lambda_1 = u_r > 0$ and therefore $\mathbf{q}_0$ cannot be $\mathbf{q}_r$. It remains to determine if $\mathbf{q}_0$ is $\mathbf{q}_l$ or $\mathbf{q}_m$, and if $\mathbf{q}_m$, what the value of $\mathbf{q}_m$ along $x = 0$ is. The fact that ω_0 from Equation (4.18) and ω_1 from Equation (4.19) are conserved across the 0- and 1-families can be used to solve for $\mathbf{q}_m$:

$$\rho_m = \left(\rho_l^\gamma + \frac{u_l - u_r}{u_{\max}}\right)^{\frac{1}{\gamma}} \tag{4.20}$$

$$u_m = u_r \tag{4.21}$$

4.3.3.5 Structure of the Riemann problem

For $\mathbf{q}_m$ to exist, the speeds of the system (Equations (4.13) and (4.14)) must be distinct. This occurs when $\rho_l > 0$; in this case, there are three distinct regions of the solution: q_l for $x \leq \lambda_0 t$, q_m for $\lambda_0 < \frac{x}{t} < \lambda_1$, and q_r for $x \geq \lambda_1 t$. In the case where $\rho_l = 0$, $\lambda_0 = \lambda_1$ and q_m vanishes. I shall deal with cases where $\rho_l = 0$ or $\rho_r = 0$ separately below.

In Section 4.3.3.4, it is established that the 0-family solutions are always shock or rarefaction waves. To determine which, one must consider the 0-family speeds λ_{0l} and λ_{0m} on either side of the nonlinear wave:

$$\lambda_{0l} = u_l - u_{\max}\gamma\rho_l^{\gamma} \tag{4.22}$$

$$\lambda_{0m} = u_r - u_{\max}\gamma\rho_l^{\gamma} + \gamma\left(u_r - u_l\right) \tag{4.23}$$

When $\lambda_{0l} < \lambda_{0m}$, the solution is a rarefaction, and when $\lambda_{0l} > \lambda_{0m}$, the solution is a shock. From Equations (4.22) and (4.23):

$$\lambda_{0l} > \lambda_{0m} \Leftrightarrow u_l > u_m \tag{4.24}$$

Classification of solutions One can identify 6 distinct conditions on the states $\mathbf{q}_l$ and $\mathbf{q}_r$ that determine the structure of the solution of any Riemann problem in the system. The derivation of these cases is discussed at length in Appendix B.

4.3.3.6 Inhomogeneous Riemann problem

The above discussion on the solution to the Riemann problem for the ARZ system of equations has assumed that $u_{\max}$ remains constant in space — i.e. that the speedlimit on either side of the interface is the same.

Clearly speedlimits vary from road to road and change even along a single lane, and the effects of these variations in speedlimits have discernible effects on traffic flow. At a decrease in speedlimit, one expects traffic to slow and increase in density, while an increase in speedlimit might cause traffic to accelerate and rarefy.

Whereas the solution to the Riemann problem developed above is a *homogenous* Riemann problem, when speedlimits on either side of a cell interface differ, the *inhomogeneous* Riemann problem must be solved.

Formally, given initial constant states $\mathbf{q}_l$ (subject to the speedlimit $u_{\max l}$) for $x < 0$ and $\mathbf{q}_r$ for $x > 0$ (subject to speedlimit $u_{\max r}$), what happens for $t > 0$? Solutions should follow the same basic structure of the homogeneous Riemann problem described previously, but rather than have a single intermediate state $\mathbf{q}_m$ emerge between $\mathbf{q}_l$ and $\mathbf{q}_r$, there may be as many as two intermediate states divided at $x = 0$ by the jump in $u_{\max}$. I term these states $\mathbf{q}_{ml}$ and $\mathbf{q}_{mr}$.

Supply and demand [Lebacque, 1996] presented the concepts of *supply* and *demand* as tools for solving inhomogeneous Riemann problems for the older LWR model of traffic flow, and later [Lebacque et al., 2005] extended their application to the ARZ model.

In the context of an (inhomogeneous) Riemmann problem, supply represents the *available* flow of traffic from the left (i.e. the flow associated with $\mathbf{q}_l$) while demand represents the *capacity* for flow on the right (associated with $\mathbf{q}_r$). The key to using these concepts to handle inhomogeneous Riemann problems is to select the lesser of these two quantities to determine the intermediate states.

I follow the method presented by Lebacque, which uses these concepts to solve for $\mathbf{q}_{ml}$ and $\mathbf{q}_{mr}$; see [Lebacque et al., 2005] for the details of their approach.

4.3.3.7 Relaxation of 'relative flow'

I have hitherto discussed the ARZ equations as homogeneous conservation laws (not to be confused with the homogeneous Riemann problem) — they fit the form of the conservation law shown in Equation (4.1) with the right-hand side of the equation as 0. This ensures that each primitive variable ρ and y is conserved in the system. Such a property is useful when describing natural phenomena, since most such equations are derived from conservation principles themselves. In my case, while ρ should certainly be conserved in regular traffic (cars chouldn't appear or disappear spontaneously), it may be desirable to relax the conservation of relative flow y. As discussed in [Aw and Rascle, 2000],

conserving this quantity leads to an unnatural dependence on initial conditions; vehicles with no traffic ahead of them (such as situations found in Case 5 above) will not accelerate beyond a quantity related to their initial value of y.

To correct this, Aw and Rascle suggest adding a small relaxation term to the right-hand side of Equation (4.7). Rather than have the two quantities equal to the zero vector $[0\,0]^{\mathrm{T}}$, scaled quantity $-\frac{y}{\rho\tau} = \frac{u_{\mathrm{eq}}-u}{\tau}$ can be added to the right-hand side of the second equation. Here τ is a time constant (typically greater than 1) representing the propensity for acceleration in the system. The modified ARZ system then becomes:

$$\begin{aligned} \rho + \rho u &= 0 \\ y + yu &= u_{\mathrm{eq}} - u \end{aligned} \tag{4.25}$$

The first equation is unchanged, while the second encourages the velocity of traffic to slowly increase towards the speed limit.

Although the underlying equations have been modified, the previously presented method for the solution of the Riemann problem remains unchanged. To account for the new system shown in Equation (4.25), a relaxation step is taken after performing the integration shown in Equation (4.4). The quantity obtained from that update ($\mathbf{Q}^{n+1}$) I instead label $\mathbf{Q}^{n+1^*}$ and the following is applied:

$$y^{n+1} = y^{n+1^*} - \Delta t \frac{u_{eq}^{n+1^*} - u^{n+1^*}}{\tau} \tag{4.26}$$

Note that the equation for ρ remains unchanged.

4.3.3.8 Lane-end boundary conditions

Each lane necessarily has a start and beginning, and to properly integrate the solution at the first and last cells in each lane as per Equation (4.4), the flux at these boundaries must be integrated. Here, I enumerate the various configurations of lane start and end

types, and how flux is computed for each of these. At the end of lane that does not flow into another (and conversely, at the start of a lane that is not downstream of another), the user may wish to impose 'no-flow' boundary conditions; lacking either a $\mathbf{q}_l$ or a $\mathbf{q}_r$, the simulation must solve a 'one-sided' Riemann problem. The two simplest (and most useful) are:

Stopped outflow When a lane's end boundary dictates that no traffic flows out of the network, a 1-wave of increasing density and decreasing velocity travels backwards through the lane. The one-sided Riemann solver for this case is identical to case 1 above with $u_r = 0$. Substituting this into Equations (4.20) and (4.21):

$$\rho_m = \left(\rho_l^\gamma + \frac{u_l}{u_{\max}}\right)^{\frac{1}{\gamma}} \tag{4.27}$$

$$u_m = 0 \tag{4.28}$$

$\lambda_s \leq 0$ and therefore $q_0 = q_m$ (see from Appendix B). (In the degenerate case where $\lambda_s = 0$, $q_l = q_m = [0\,0]^{\mathrm{T}}$)

'Starvation' inflow In the case where no traffic is flowing into a network, there is no 1-wave, and the 2-wave simply propagates to the left — the numerical flux is obtained through a Riemann solver similar to that used for case 4 above. Certainly $\lambda_1 = u_r \geq 0$, and as there is no wave from the 1-family, we know that $q_0 = q_l = [0,\, 0]^{\mathrm{T}}$

Network boundaries It is technically very simple to construct a network that has no external boundaries — where no lane flows out of the network, nor does any lane receive traffic from outside the network — but such a network is not very realistic. Generally, it is desirable to simulate a network that is ultimately a subset of a larger traffic network. There are any number of conditions the user may wish to impose on these boundaries, and they differ for inflow and outflow conditions.

For lanes that start at a boundary (inflow conditions), one will generally wish to prescribe some type of upstream traffic, possibly in a time-dependent manner. This traffic is imposed by doing a full Riemann solve using the approach described in Section 4.3.3.4 with the right state $\mathbf{q}_r$ the first computation cell of the relevant lane, and a given $\mathbf{q}_l$ that describes the type of flow entering the network. The case where no external traffic enters the lane is simply a special case of this approach. This corresponds to the so-called 'Dirichlet' type boundary condition from the partial differential equation literature.

Lanes with an end point at a boundary require some outflow condition. Dirichlet boundary condition may be imposed here, just as with the inflow case; one could, for example, impose a high-density, low-velocity condition that captures the behavior of an out-of-network traffic jam, or we could have a zero-density (and implicitly, high-velocity) condition to represent an empty road. For outflow, it will also sometimes make sense to impose a 'Neumann' type boundary condition where we specify a certain flux at the boundary directly. In his case, we will almost always wish to specify that the flux is zero; this represents an 'absorbing' boundary that causes no waves to travel backwards through the simulation. To effect this condition, we can proceed with a full Riemann solution with $q_l = q_r$, or equivalently (and more cheaply) simply omit the $\mathbf{f}(\mathbf{q}(a))$ term in Equation (4.4).

4.3.4 Visual representation of vehicles

I use a discrete, particle like representation of vehicles for graphical rendering called "carticles". These primarily serve to provide a visual representation of traffic, but also play a role in deciding when to begin lane changes (see Section 4.3.5). Each lane i has an associated set of carticles $C_i = \{c_0, c_1, c_2 \ldots\}$; each carticle in turn has a minimal amount of state associated with it:

Position — the parametric position $s \in [0, 1]$ of the rear axle of the carticle along the lane.

Lane change state — an enumerant that signifies that the carticle is changing lanes and if so, which direction (left or right) the change is in.

Lane change progress — a scalar $s_m \in [0, 1]$ representing the progress of the carticle's current lane change — 0 signifies that the carticle has not begun to turn while 1 represents a carticle that has completed its lane change.

Vehicle type — an enumerant representing the type of vehicle of the carticle. Currently, this simulation technique only uses information about the length of each vehicle type and position of the rear axle relative to the overall length.

4.3.4.1 Carticle motion

The parametric position of each carticle j (s_j) in the system is advanced at each simulation step via the simple ODE:

$$s_j'(t) = \frac{u_{\text{lane}}(s_j(t), t)}{L_{\text{lane}}} \tag{4.29}$$

Here $u_{\text{lane}}(s_j(t), t)$ represents the velocity field of the lane to which carticle j belongs (defined in Equation (4.11)) and L_{lane} the length of said lane. Since the evaluation of the right-hand side of Equation (4.29) consists of inexpensive interpolation of discrete data, explicit 4^{th} order Runge-Kutta is used to integrate each carticle's position. When s_j is greater than 1 — and thus has traveled beyond the end of the lane — the carticle is either removed from the simulation altogether (in the case of an external boundary condition) or "pass" it to the next lane and adjust s_j appropriately.

4.3.4.2 Carticles at the continuum level

Carticles represent the positions of vehicles, but in order to have the underlying continuum simulation reflect the position of these vehicles, the simulation must "seed" the discrete cells along each lane with the appropriate density and velocities of each carticle.

When all Δx_j (see Section 4.3.3.1) are greater than the length of any vehicle type, it is certain that the interval associated with each carticle overlaps no more than 2 grid cells. The quantity ρ stored at each grid cell is interpreted as "cars per car length" (as described in Section 4.3.3.3); thus, for each cell i a carticle j with velocity u_j overlaps, the updated density (ρ_i') and velocity (u_i') are computed at i from their original values $[\rho_i,\, u_i]^{\mathrm{T}}$ and the contribution of j:

$$\Delta\rho_i = \frac{o_{i,j}}{\Delta x_{\text{lane}}}$$

$$\rho_i' = \rho_i + \Delta\rho_i \tag{4.30}$$

$$u_i' = \frac{\rho_i u_i + \Delta\rho_i u_j}{\rho_i'} \tag{4.31}$$

Here $o_{i,j}$ is the length (in real, not parametric space) that the carticle j overlaps cell i.

4.3.5 Lane changes and merges

This method handles the movement of vehicles from one lane to another (interchangeably called a lane change or a merge) using a combination of information from carticles and the density/flow data from the continuum model.

This method arises from the following observations:

- A lane change generally takes place on a longer timescale than a single simulation step.
- Once a vehicle begins a lane change, it continues to move into the other lane until the lane change is finished.

Based on these observations, initiate lane changes are initiated on a per-carticle basis. When certain conditions described below are met, a carticle will be marked as being in a either a left or right lane change and the simulation will account for the lateral movement of that carticle and the underlying flux between lanes.

Starting a lane change Lane changes are initiated based on some simple rules that are used to compute a signed *merge factor* that ultimately determines if a vehicle will change lanes.

At a vehicle's position along a lane, there are as many as two adjacent lanes to move to, but it is required that the adjacency on each side continues for a long enough distance forward to make the lane change possible, given the vehicle's current velocity.

Additionally, no vehicles may lie in the potential path of the lane change; the technique searches for carticles in the cells in the immediate vicinity of the cell that neighbors the lane-change candidate. If any vehicles are present with trajectories would overlap the potential lane change path in the next timestep, the path is removed from consideration.

Finally, if there is at least one suitable adjacent lane, the desirability of changing lanes is determined — the aforementioned merge factor. Vehicles change lanes to ensure a certain path is taken (i.e. to be able to make certain turns or take exit ramps), to move into faster traffic, and for a variety of other reasons. This technique does not account for the long-term intent of each vehicle, so lane changes for routing make little sense in this context.

A lane change is made based on the perceived increase in velocity attainable; if the traffic ahead of a vehicle is moving much more slowly than the traffic in a neighboring lane, change to that faster lane is attractive.

The following formula is applied to each adjacent lane $k \in \{l, r\}$ to determine the merge factor m_k:

$$m_k = \frac{u_{\text{adj}_k}}{u_{\text{ahd}}} \tag{4.32}$$

Here u_{adj_k} is the continuum velocity in the cell in the adjacent lane k that neighbors the candidate vehicle and u_{ahd} the velocity in the cell ahead of the candidate vehicle.

If there is more than one candidate lane being considered, the lane with the largest merge factor is selected. Now if this ultimate merge factor exceeds a threshold — I found 1.1 worked well in my experiments — a lane change is initiated.

4.3.5.1 Lane changes at the continuum level

As I have discussed, lane changes are initiated at the carticle level and must be carried to completion. Furthermore, a lane change will generally require several simulation steps to perform. The transition of the vehicle at the (continuum) dynamics level must be accounted for to properly reflect effects of the lane change, but a straightforward transfer of the density and velocity corresponding to the vehicle over the course of the lane change is not sufficient.

A vehicle peforming a lane change effectively occupies a space in *both lanes simultaneously* — its motion dictates the behavior of traffic behind it in both the lane it is leaving and the one it is entering. For this reason, once a vehicle begins to switch lanes, its density is *duplicated* and velocity information in the adjacent cell and proceed with the simulation until the lane change has completed, at which point the density and velocity representing the vehicle in the lane it left is removed. This violates conservation for a brief period of time, but the ultimate result is a superior description of the effects of a lane change and density after the lane change is the same as it was beforehand. I describe how to convert carticles to their corresponding continuum-level information in Section 4.3.4.

4.4 Results

4.4.1 Examples

I have tested my technique on a number of synthetic road networks, and on a large 'clover-leaf' intersection; see Figures 4.1, 4.3, 4.4, and 4.5 for visual depiction. Figure 4.1 shows an overview of a 'cloverleaf' freeway interchange. Figures 4.3 and 4.4 show scenes from traffic traveling along a freeway, and Figure 4.5 shows traffic flowing near an offramp on a highway system.

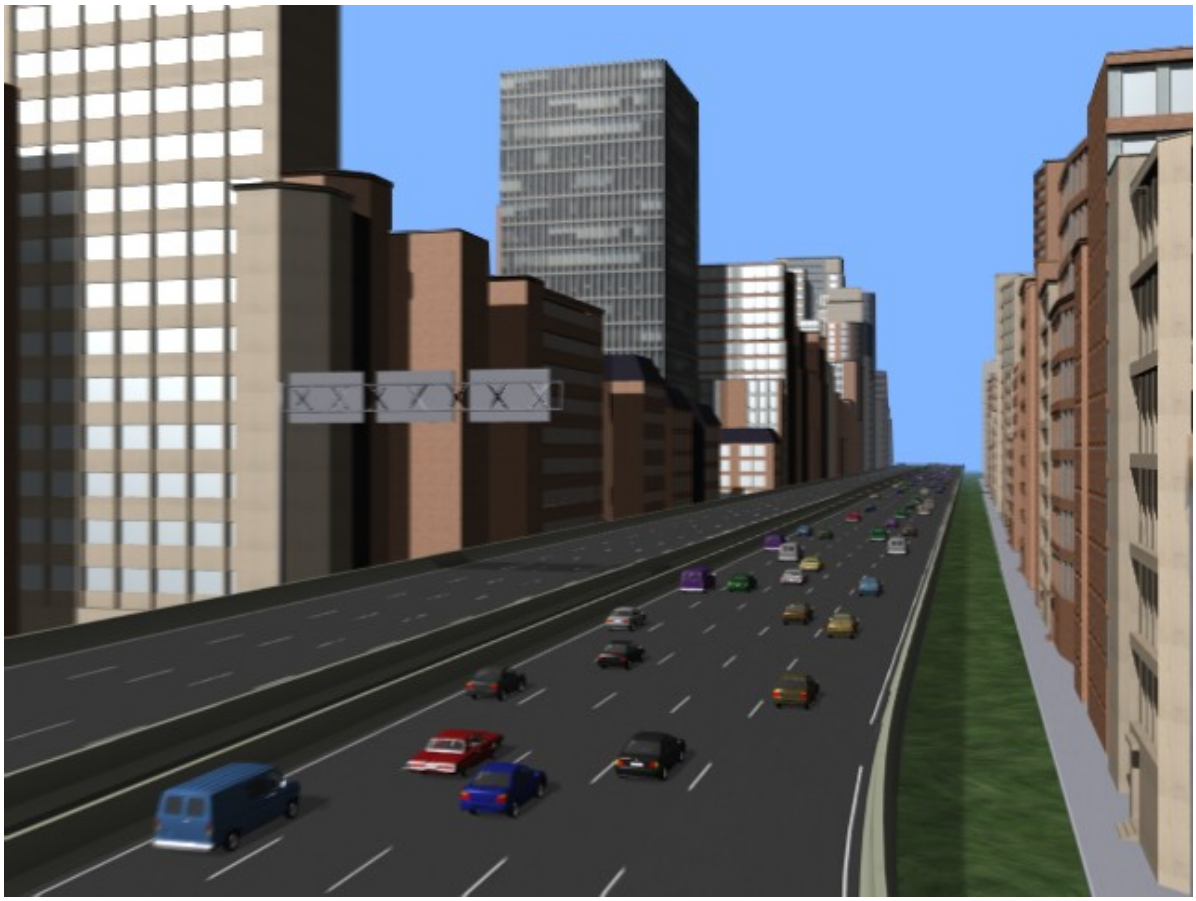


Figure 4.3: A flyover of a freeway

4.4.2 Comparison with agent-based simulation

To get better understanding of the performance of this technique as compared to a microscopic, agent-based simulation method, I have timed simulations using my technique and a popular, state-of-the-art traffic simulator for a variety of scenarios. The Simulation of Urban MObility (SUMO) project [SUMO, 2009] is an open-source traffic simulation package originating from the Centre for Applied Informatics at the University of Cologne and the Institute of Transport Research at the German Aerospace Centre. SUMO is based on a microscopic car-following model of traffic flow, and one would therefore expect its performance to be linear in the number of vehicles in the simulation.

The road network description format for each simulator varies greatly, so I chose

Figure 4.4: A freeway in a city

a simple simulation network as the basis for comparison to ensure that my simulator and SUMO would be operating on precisely the same input. The network is a 6-lane straight stretch of freeway 10km long. I provided input data to each simulation as series of vehicles entering the network; each scenario ran for a constant period of time but varied in the number of vehicles emitted over the interval. The result of this simulation is shown in Figure 4.6. Note that as there is no parallel implementation of SUMO available, I performed timings for both simulators on a single processor.

One observes nearly linear performance in the number of cars for SUMO for scenarios with a small number of cars, but a dramatic drop in performance as the number of cars increases. In contrast, my simulator maintains a nearly linear performance over all

Figure 4.5: Vehicles exiting a freeway

ranges of inputs, but with a larger constant overhead than SUMO for a small number of cars. The qualitative results here are roughly what one would expect: SUMO should have some cost per vehicle being simulated, resulting in performance linear in the number of vehicles in the simulator; while my technique has a constant cost associated with the total network size, regardless of the number of vehicles, as well as a much smaller added cost per vehicle in the network.

4.4.3 Scaling of parallel implementation

Numeric techniques based on hyperbolic equations are frequently very amenable to parallel computation; the majority of the work done in this method involves solving the Riemann problem at each cell, and this can be done independently for each interface

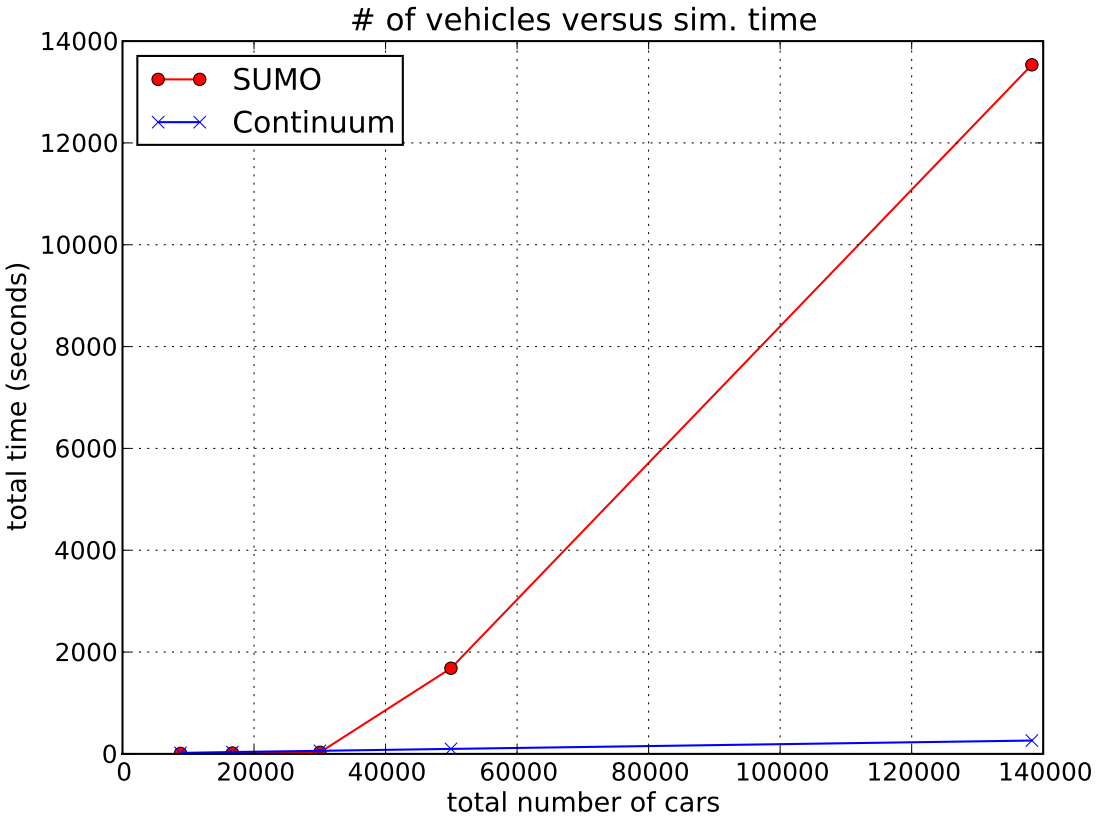


Figure 4.6: Comparison of performance scaling of agent-based SUMO (red, top) vs. my simulator (blue, bottom) as the number of cars increases.

between cells. In practice, it makes sense to divide the work into coarser tasks involving multiple lanes. Detailed discussion of how this technique has been parallelized can be found in Appendix D; an initial parallelization has produced sub-linear but encouraging parallelism — approximately 5x on a 4-core Intel i7 architecture (with SIMT).

4.5 Conclusion

I have presented a method for the generation of realistic traffic animations. The method is based on continuum dynamics with a facility to extract discrete results. I have reported preliminary results on parallelization and demonstrated this method's ability to generate traffic on large, real-world road networks.

4.5.1 Limitations and future work

This technique can handle a wide variety of traffic conditions, but is still limited in the scope of traffic-related phenomena that it can handle. Vehicle collisions, road conditions, and weather, and the distinctions of the vast array of vehicle types and driver types are not currently considered in my prototype system.

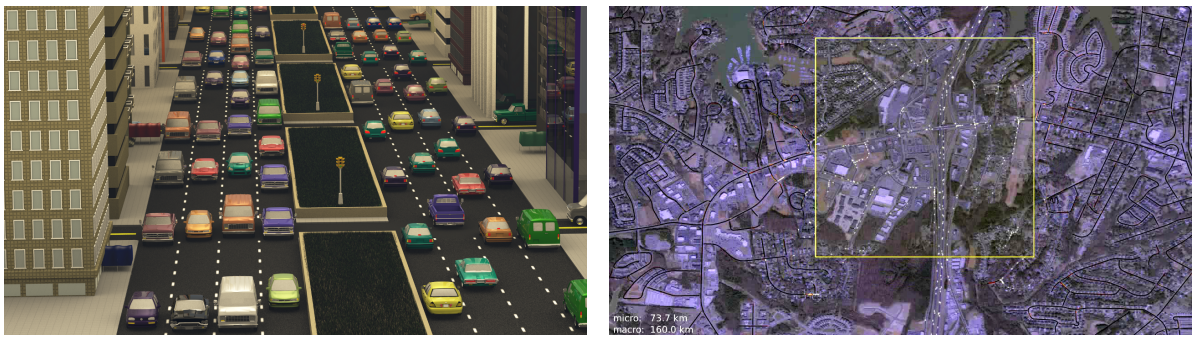
Chapter 5

Hybrid Traffic Simulation

5.1 Introduction

With automobile traffic ubiquitous in developed nations and on the rise in developing ones, traffic simulation techniques such those found in Chapter 4 are of great use in analyzing road usage and for generating visual representations of traffic flow for virtual environments.

However, there are certain limitations of purely continuum (or *macroscopic*) approaches such as that described in Chapter 4; their greatest strength — the ability to describe the motion of many vehicles with a paucity of computational cells — demands that vehicles behave with largely aggregate behavior.



(a) Interactive 3D visualization of urban traffic

(b) Augmenting a satellite image with real-time traffic

Figure 5.1: Scenes of traffic generated with my hybrid technique

In many situations, it is desirable to have individual vehicles act individually; a user may wish to simulate certain drivers with more aggressive or timid behaviors, or to have some vehicles to travel specific routes. Continuum models make this type of behavior difficult to achieve and augmentations directly made to macroscopic methods are likely to concede the very efficiency that makes them useful to begin with.

Agent-based traffic simulations (known alternately as *microscopic* methods) determine the motion of each vehicle individually through a series of rules. These rules are easy to vary on an agent-to-agent basis; thus microscopic simulation techniques are well-suited to vehicles with inhomogeneous governing behaviors.

Microscopic simulation techniques can capture individualistic vehicle behavior while macroscopic simulations maximize efficiency; this method uses a hybrid simulation technique that takes advantage of these complimentary features. I combine the continuum traffic approach described in Chapter 4 with a conventional agent-based traffic simulator, allowing for certain areas of the road network to be handled by the macroscopic simulator and others by the microscopic.

This chapter introduces a multi-method simulation technique that combines the strengths of two classes of traffic simulation to achieve flexible, interactive, high-fidelity simulation on large road networks. I describe my methods for handling the transfer of vehicles between continuum and discrete simulation areas and discuss how the constituent simulation components are adapted to handle this transition. I demonstrate the variety of traffic flows my method can produce and analyze its performance on modern architectures, and measure its ability to match real-world traffic flow with a validation comparison.

5.2 Related Work

The state of the art in traffic simulation, in both the engineering and graphics communities, is reviewed in Section 4.2; here I review some related work on hybrid techniques and road network representations.

5.2.1 Adaptive Simulations

One of the goals for the hybrid technique discussed in this chapter is a method that is able to dynamically trade simulation detail or features for performance. Such techniques have been investigated in graphics for other types of simulation; in particular, the work of [O'Brien et al., 2001] demonstrated automatic level-of-detail refinement for particles systems to achieve large-scale, realistic simulations. [Redon et al., 2005] present a technique for performant articulated-body simulation based on adaptive joint behavior. The technique presented in this chapter can be considered a form of adaptive simulation; rather than modify levels of refinement in the simulation — i.e., using coarser spatial representations — my technique changes the actual type of simulation to a very different class.

5.2.2 Traffic Networks and Road Representations

Traffic networks are complex entities; while polygons and line segments in the plane typically form adequate domains for crowd simulation, and surface geometry with computational grids is suitable for most physically-based simulations, traffic simulation presents unique challenges in representing and acquiring simulation domains. A natural choice for domains for traffic simulation is to use real-world networks; digital representations of real-world networks are widely available in the form of connected polylines. Procedural cities and roads are attractive candidates for domains for many graphical problems; recent work by [Galin et al., 2010] and [Chen et al., 2008], among others, has enabled

the generation of detailed, realistic urban layout and roads.

Numerous spatial representations of curves have been developed over the years — too many to begin to enumerate here. I chose the 'arc road' representation presented in this chapter (and Appendix E) based on a number of reasons:

- Smooth appearance and resemblance to real roads
- Ease of extension from widely available polyline data
- Connection to vehicle kinematics
- Thrift of computation, definition, and representation

[van den Berg and Overmars, 2007] proposed a model of roadmaps for robotics using connected *clothoid* curves. However, while their choice of representation is based solely on the need to generate vehicle motion, for the purposes of my work, the representation must also be suitable for the generation of road surfaces, which are not necessarily clothoids. Additionally, clothoid curves are expensive to compute — requiring the evaluation of Fresnel integrals — whereas our relies solely on coordinate frames and sines and cosines.

[Nieuwenhuisen et al., 2004] use circular arcs, as we do, to represent curves, but as in [van den Berg and Overmars, 2007], they are used to smooth the corners of roadmaps for motion planning. Furthermore, neither of these techniques have been investigated for the case of extracting ribbon-like surfaces, as we do (in Section E.2.5), nor is there an established technique for fitting them to multi-segment, non-planar polylines such as that I present in Section E.2.2.1.

5.3 Method

5.3.1 Overview

In this section, I briefly discuss the data structures used in the simulation and proceed with a description of our hybrid simulation technique.

5.3.2 Road networks

The hybrid technique's simulation domains are based on those described in Section 4.3.2; this method requires the same detailed road network data structure that provides parametrized information about adjacency, road shapes, and speed limits as well as network boundary conditions. The lane-based approach this representation espouses works well with both the microscopic and macroscopic simulation components, as many of their computations are carried out in units of single lanes.

The following describes enhancements to the road network data structure and how they relate to this hybrid simulation technique; in particular, it introduces a new representation for spatial curves ideal for traffic simulation.

5.3.2.1 Intersections

In my data structure, an intersection is where more than two lanes meet and traffic is able to transition between them. Signaled intersections simply have a sequence of states that enumerate what each lane incident on the intersection does; some pairs of lanes may be connected, and others stopped. Each state can have a duration (in the case of a timed signal) or be triggered by the presence of vehicles at an incoming lane.

5.3.2.2 Arc roads

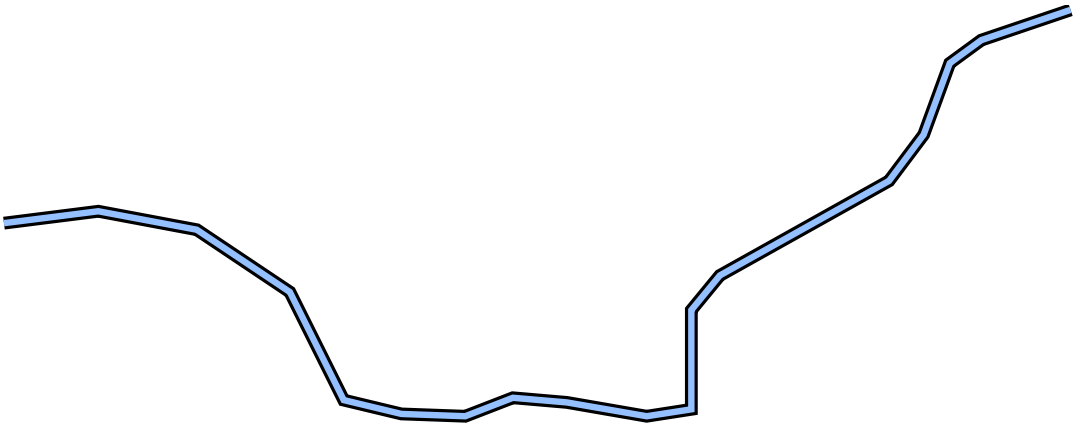
This hybrid simulation technique uses an augmented representation of road geometry. While the simulation technique described in Chapter 4, as well as most digital maps, use

polylines (C^0 series of line segments) to represent road shapes, this generally leads to visible artifacts in the motion of vehicles along these roads; the sudden change in direction between segments (due to the discontinuous derivative of non-degenerate polylines) produces jerky turning motions in vehicles, particularly at sharp turns. The naïve solution to this problem is to simply refine the approximation of the underlying smooth curve with more straight line segments. While this can give acceptable results in certain cases, it leads to a proliferation of data points, which requires more resources to handle. Furthermore, barring extreme levels of refinement, the C^0 nature of the representation is still observable. It should also be noted that there is no clear method by which to use polylines in 3 dimensions to describe vehicle motion with consistent orientation. Figure 5.2a shows a polyline-based road imported from GIS data. I propose *arc roads*, which consist of alternating straight line segments and circular arcs. This representation has numerous advantages over polylines:

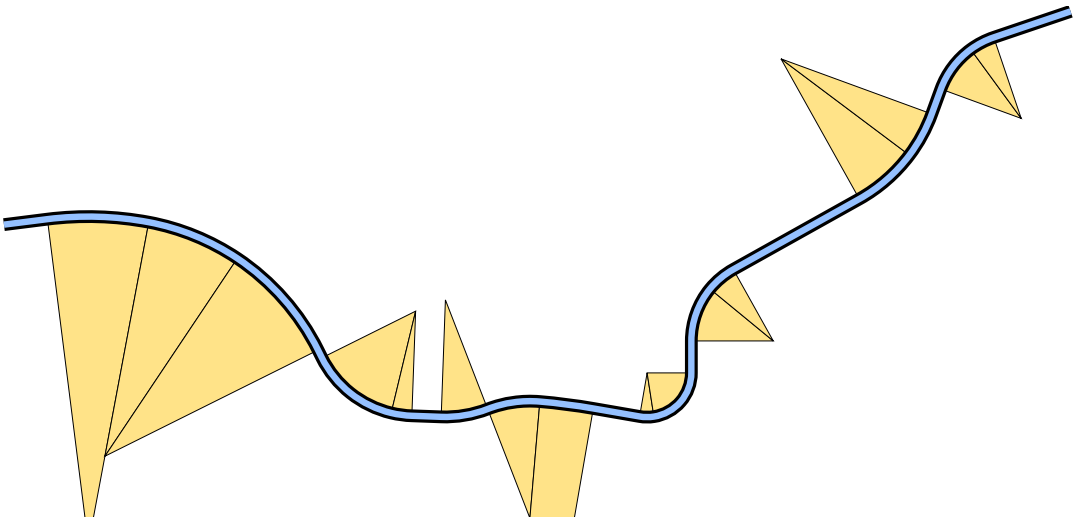
- The curve is C^1.
- Useful and well-defined in 3 dimensions.
- Straightforward to derive from existing polyline data.
- Admits a simple parametrization.
- A consistent Frenet frame is available at each point.
- Computationally-efficient evaluation of position and orientations.
- Allows for much smoother animations of vehicle motion.

Figure 5.2b shows an exemplary arc road derived from the polyline shown in Figure 5.2a.

A detailed description and analysis of arc roads can be found in Appendix E.



(a) Polyline road geometry (from the TIGER® database [U.S. Census Bureau, 2010] via OpenStreetMap [OpenStreetMap community, 2010])



(b) An *arc road* derived from the above polyline. The orange arcs show the center and radius of each arc used to give the road its smooth appearance.

Figure 5.2: A polyline and derived arc road

5.3.3 Overview of simulation methodology

My method of hybrid traffic simulation combines the strengths of both macroscopic and microscopic techniques to be both efficient and flexible. At any given point in simulation, the road network consists of mutually exclusive regions of two types; one where a continuum technique is used to describe vehicle movement and another where a discrete, agent-based technique for simulation dynamics is applied. These regions are not necessarily connected nor static; one may pick either technique to govern a given part of the network based on what the user wishes to observe, the current volume and velocity of traffic in the network, or to enforce certain types of desired behavior.

Terminology I use the terms *upstream* and *downstream* to refer to relative directions — against and with the flow of traffic, respectively. From a technical standpoint, certain waves do indeed travel against the flow of traffic, in which case the meanings could be reversed, but I shall never intend this interpretation.

Continuum and *macroscopic* shall be used interchangeably to refer to the continuous simulation regime and associated regions of the network; *discrete* and *microscopic* similarly refer to the other, agent-based regime and regions. Clearly even the 'continuous' type of simulation is ultimately realized through discrete cells, but I shall avoid confusing these terms.

Hybrid technique A key component of my hybrid simulation technique is how the two different types of simulation are coupled together; continuum traffic expects a density-like quantity of cars per car length with a velocity component, while discrete simulation is carried out with the explicit position and velocity of each vehicle in the network. The partitioning of the network allows for an arbitrary number of interface points where the two types of simulation must be coupled and vehicles under one regime must be passed to the other.

This technique for describing the flow of traffic requires a road network with suitably-defined boundary conditions and an initial state for any vehicles that begin the simulation in the network. I then proceed by taking discrete time steps of varying length Δt, wherein the state of traffic is considered and integrated forward in time.

A time step of this technique goes as follows:

Step 1 — Advance continuum regions:

(a) Determine the rate of traffic flows (depending on both the density and velocity of traffic), also known as *flux*, between each adjacent cell.

(b) Compute the minimum timestep, Δt, using the maximum speed from each cell solution in (a).

(c) Integrate each cell using the 'flux' solution from Step 1a and Δt from Step 1b.

Step 2 — Update *flux capacitors* (see Section 5.4.3.1) and add discrete cars as needed.

Step 3 — Advance agent-based regions (using the same Δt computed in Step 1b).

Step 4 — Aggregate all discrete vehicles (Section 5.4.1) that flow into a continuum region.

In other words, I separately advance the continuum and agent-based simulations, while we manage the transition of vehicles between the different simulation regimes. Below, I describe the basic simulation techniques used in Step 1 and Step 3. Later, in Section 5.4, I present my main contributions — techniques for converting between different types of simulation.

5.3.4 Continuum traffic simulation

The continuum regime of this simulation uses the technique described in Chapter 4; briefly, each lane is divided into discrete computational cells that represent the two

conserved quantities $\mathbf{q} = [\rho,\, y]^{\mathrm{T}}$ from the Aw-Rascle-Zhang system of equations (Equation (4.7)). The solution is then advanced in discrete timesteps with explicit integration of split fluxes obtained through our Riemann solver (see Section 4.3.3 for more details).

The continuum components of the network are advanced in Step 1 of the procedure in Section 5.3.3 — before the microscopic simulation components are handled. This is to ensure that the timestep Δt used in the each component is the largest stable timestep achievable. The macroscopic simulation component has more stringent stability requirements than the micro component; by performing the macro update prior to the micro, I use the Δt computed in Step 1b of the list in Section 5.3.3 in the later microscopic update performed in Step 3 of the same list.

It is worth keeping in mind that while our choice of timestep for the continuum component is restricted for stability reasons, said restriction is hardly stringent in most contexts; the discussion in Section 4.3.3.1 makes the point that the typical timestep for the usual simulation context is on par with the $\frac{1}{30}$s frame length found in most animation scenarios. Indeed, depending on the order of integrator used in the microscopic simulation component, it is possible that the foremost concern will be the accuracy of this integration.

The important distinction between the method presented in the previous chapter and the continuum simulation component under discussion is the non-trivial problem of instantiating discrete vehicles (suitable for use in display or for the discrete simulation component). I discuss my solution to this problem in Section 5.4.3.1.

5.3.4.1 Simulation data structure

The data structure needed for continuum simulation is the same as that used in Chapter 4; to each lane j, I associate a regular 1-dimensional grid of length N^j of unknowns $\mathbf{Q}_i^j = \left[\rho_i^j,\, y_i^j\right]^{\mathrm{T}}$, $i \in [0, N^j - 1]$. The method also stores a grid with each lane a regular 1-dimensional grid of $N^j - 1$ Riemann solutions that are populated during Step 1a and

used to update the Q_i^j during Step 1c. See Section 4.3.3.1 for information on how N^j for each lane is computed.

5.3.5 Microscopic simulation

The regions of the traffic network under the microscopic regime are handled by an agent-based 'car-following' method adapted from the work of [Treiber et al., 2000]. Essentially, at each timestep, each vehicle chooses some acceleration based on the distance to the vehicle ahead of it and their relative velocity. This acceleration is used to update the vehicle's velocity, which is then used to update the vehicle's position. Care must be taken to ensure the vehicles behave correctly at intersections, but this approach is straightforward and reasonably efficient.

5.3.6 Car-following model

Most microscopic simulations are based on these car-following models, wherein each vehicle computes its acceleration for the next timestep. The acceleration value is chosen to satisfy several conditions:

Collision avoidance A vehicle should not run into the vehicle ahead of it.

Acceleration limitations The ability of a vehicle to increase or decrease its speed is limited.

Preferred velocity Each vehicle has a 'preferred' velocity, usually related to the conditions of the road and driver behavior, that the vehicle should try to achieve.

The notion of a car-following model as outlined above requires that each vehicle know only the position and velocity of the vehicle directly ahead of it to compute its acceleration for a time step. In the following paragraphs, I discuss how each vehicle determines what vehicle lies ahead of it, and how I compute acceleration given this information.

5.3.6.1 Simulation data structure

To keep track of the state of the vehicles in each lane j in the micro regime, I maintain a list C^j (of cardinality V^j) in increasing order of the parametric position of each vehicle $i = 0, 1, \ldots, C^j - 1$ in the lane. Vehicles entering the lane are naturally inserted into the front of the list, while vehicles that flow out of the lane are removed from the back.

5.3.6.2 Locating leaders

Fundamental to the operation of the car-following model is the ability for a given vehicle to determine the velocity of and distance to the vehicle ahead of it (its *leader*). The algorithm for locating the leader for a given car, LOCATE-LEADER, is given in Algorithm 5.1.

I also define an *interaction limit distance* $d_{\max}$ on the order of 50–1000 meters beyond which the leader has no effect on the behavior of the follower. If, in the search for the leader, $d_{\max}$ is exceeded, the searched search is terminated and ignore car-following effects in the acceleration computation (described shortly in Section 5.3.6.3).

The algorithm for finding a leader works as follows; when a vehicle is *not* the last in the lane, the next vehicle is trivial to compute from the data structure described in Section 5.3.6.1; the leader of vehicle $i < C^j - 1$ in lane j is simply vehicle $i + 1$.

If a vehicle is the last in the lane, consider the next logical leader: if the lane ends at a stopped intersection, a *virtual* vehicle with zero velocity is created, positioned such that its rear bumper corresponds to the end of the lane; if the lane flows freely into another, the first vehicle in that lane (found with the FIRST-CAR procedure, which trivially returns the first car in the list C^L) is used. Should a lane contain no vehicles, the process is repeated recursively until $d_{\max}$ is exceeded or a leader encountered.

There is one more consideration in this process; if the search for a leader comes upon a lane in the macroscopic simulation regime, a mechanism for reporting the first (if any) vehicle in the lane is needed. My solution uses the *instantiation mechanism*

Algorithm 5.1 LOCATE-LEADER

LOCATE-LEADER(L, c)

```
    // L — a lane, c — the car in L to search from.
1   if c.nextcar   // If there is a car after c in L...
2       return c.nextcar   // return it.
3   elseif L.next  // If lane flows onto another...
4       return LOCATE-LEADER-SUB(L.next, L.length)   // search in that lane.
5   else       // This lane stops at a (stopped) intersection or is a dead-end...
6       return VIRTUAL-CAR(0, L.length)       // return 'virtual' stopped car.
```

LOCATE-LEADER-SUB(L, d_{curr})

```
    // L — a lane, d_curr — accumulated distance in search.
1   if d_curr ≥ d_max // If the search exceeded d_max...
2       return ∅ // return nothing.
3   elseif !EMPTY(L)  // If L has cars...
4       return FIRST-CAR(L)  // return the first car.
5   elseif L.next
6       return LOCATE-LEADER-SUB(L.next, d_curr + L.length)
7   else
8       return VIRTUAL-CAR(0, d_curr + L.length)
```

The LOCATE-LEADER and LOCATE-LEADER-SUB procedures used to find the leader of a given car.

developed to couple macroscopic simulations to microscopic ones, and it is discussed in Section 5.4.3.2.

5.3.6.3 Acceleration computation

At each timestep, vehicles undergoing microscopic simulation must choose how to accelerate to avoid collision with the vehicle ahead of them while continuing to travel at a pace that is reasonable given the conditions (i.e., the speedlimit of the road, behavioral factors associated with the driver or vehicle). The following formula for the acceleration of vehicle i being led by vehicle j is from [Treiber et al., 2000]:

$$a_i = a_i^{\max}\left(1 - \left(\frac{v_i}{v_i^{\text{pref}}}\right)^{\delta} - \left(\frac{s^{\min}(v_i, v_j)}{s_i}\right)^2\right) \tag{5.1}$$

The terms in the equation are explained as follows:

- $a_i^{\max}$ is the maximum forward acceleration vehicle i is capable of.
- v_i is the current velocity of vehicle i.
- v_i^{pref} the preferred velocity of vehicle i (perhaps based on the speedlimit of the current lane).
- δ a positive integer affecting the weight the vehicles preferred velocity has on its acceleration (a value of $\delta = 4$ is suggested by [Treiber et al., 2000]).
- s_i the distance between the i^{th} and the next vehicle in the lane (vehicle j).

Figure 5.3 depicts these quantities graphically. $s^{\min}$ is relation that returns the minimum acceptable distance between two vehicles based on their velocities; I use the formula given in [Treiber et al., 2000]:

$$s^{\min}(v_i,\, v_j) = s_i^0 + s_i^1\sqrt{\frac{v_i}{v_i^0}} + T_i v_i + \frac{v_i\,(v_j - v_i)}{2\sqrt{a_i^{\max} b_i^{\text{pref}}}} \tag{5.2}$$

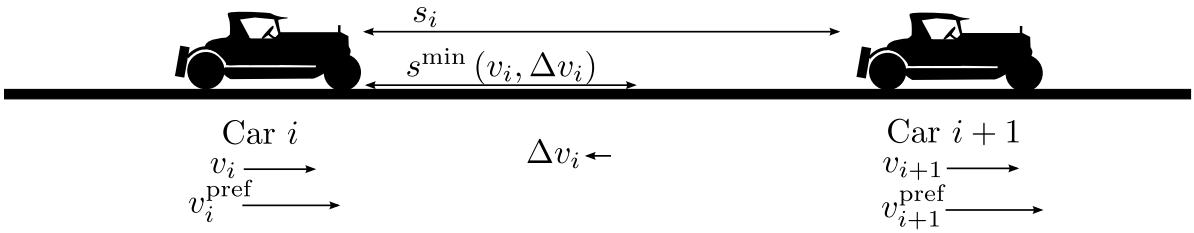


Figure 5.3: Terms in car-following acceleration computation equation

Here, s_i^0, s_i^1, T_i, and b_i^{pref} are parameters specific to the driver/vehicle model being used representing 'jam distances', 'safe headway time', and 'preferred braking acceleration'.

Equation (5.1) represents two distinct factors; a positive or negative acceleration based on the ratio of a vehicle's current velocity v_i to its preferred velocity v_i^{pref} (the first two terms inside the outermost parenthetical) and a zero or negative acceleration based on the ratio of the vehicle's minimum braking distance and actual distance to the vehicle ahead of it.

5.3.6.4 Lane changes

While the lane following model handles traffic within a single lane, real traffic networks contain highways and arterial roads with numerous lanes, and real drivers change lanes to avoid slower moving vehicles. To simulate this behavior, I use a method based in part on [Kesting et al., 2007] and on the method presented in 4.3.5. I use the underlying car-following model to calculate what the gain in acceleration, Δa, is needed for a vehicles $c_{merging}$ to change lanes. If this Δa is greater than a threshold value, the car will attempt to change lanes if it is safe to do so. This safety check is done by finding the car c_{follow} that would be behind $c_{merging}$ if it changed lanes. If the deceleration of c_{follow} after $c_{merging}$ changes lanes is less than a factor $polite \in [0, 1]$ times the maximum deceleration, then the lane change is considered safe. Finally, to preserve the dynamics of the microscopic simulation, the car is considered to be its current lane and the lane it is merging into throughout the merge.

5.4 Transitioning between continuum and agent-based models

To take advantage of agent-based and continuum simulation, we introduce a hybrid, multi-method technique that divides the road network into multiple disjoint (and not necessarily connected) regions — each region is governed by either agent-based simulation or continuum simulation.

These regions in our simulation are dynamic; we can adaptively change the simulation method in a region as needed to observe certain phenomena, meet performance requirements, or to acquiesce to user input. To achieve this, we must be able to convert discrete vehicles from agent-based simulation lanes into the aggregate format necessary for continuum simulation, and we must be able to introduce discrete vehicles corresponding to the distribution of density in continuum data.

Sections 5.4.1 and 5.4.2 establish the fundamentals of the conversion process, and described how whole regions are converted from one regime to the other. Later, in Section 5.4.3, we present how traffic *flowing* from one region into another is handled.

5.4.1 Conversion of microscopic regions to macroscopic through averaging

Vehicle support functions It is straightforward to compute a continuum representation from a list of discrete vehicles; the $\mathbf{Q}_k = [\rho_k, y_k]^{\mathrm{T}}$ stored for macroscopic simulation are averaged quantities. For each discrete vehicle C_i with front-bumper position p_i, length l_i, and velocity v_i, define a boxcar-like support function as follows:

$$S_i(x) = H(x - p_i + l_i) - H(x - p_i) \tag{5.3}$$

where $H(x)$ is the Heaviside function. The support of all n vehicles is then given by the sum:

$$D(x) = \sum_{0}^{n-1} S_i(x) \tag{5.4}$$

An exemplary plot of this function is depicted in Figure 5.4. Assume, without loss of

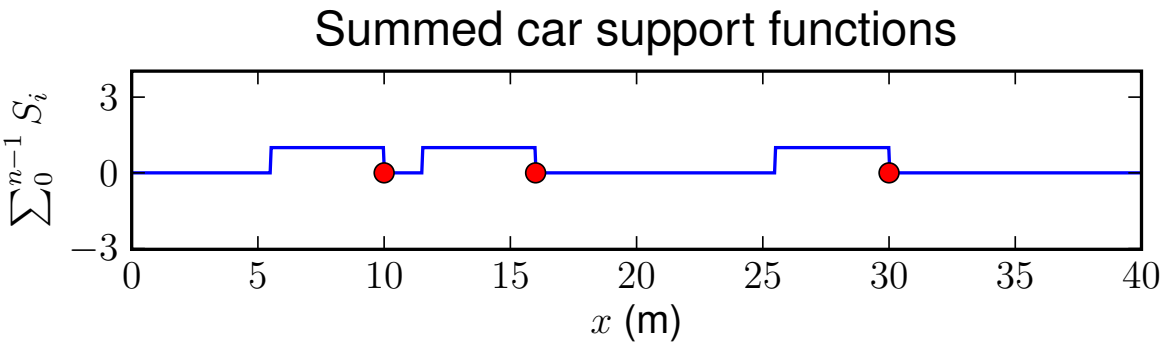


Figure 5.4: The car support function $D(x)$ for a series of cars with front bumpers at $x = 10$, 16, and 30.

generality, that the continuum cells are uniformly spaced by Δx; then the ρ_k for each cell of the continuum is:

$$\rho_k = \frac{1}{\Delta x} \int_{k\Delta x}^{(k+1)\Delta x} D(x)\ dx \tag{5.5}$$

Given n vehicles to discretize. The velocity u_k is

$$u_k = \frac{1}{\Delta x \int_{k\Delta x}^{(k+1)\Delta x} D(x)\ dx} \int_{k\Delta x}^{(k+1)\Delta x} \sum_{0}^{n-1} v_i S_i\, dx = \frac{1}{\Delta x^2 \rho_k} \int_{k\Delta x}^{(k+1)\Delta x} \sum_{0}^{n-1} v_i S_i\, dx \tag{5.6}$$

To compute the primitive-variable quantity y_k for each cell, I simply apply Equation (4.8).

Averaging algorithm In practice, the $\mathbf{Q}_k$ can be computed in an efficient manner — linear in the number of cars n — by iterating over each vehicle C_i, $i \in [0, n-1]$. For each i, I compute the intersection of the nonzero portions of S_i with the continuum grid and apply Equations (5.5), (5.6), and (4.8). So long as $\Delta x = \Omega(l_i) \forall l_i$, each vehicle will cover a constant number of grid cells and the averaging process is $O(n)$.

5.4.2 Conversion of macroscopic regions to microscopic through Poisson instantiation

The process of initializing a microscopic region given a macroscopic one is much more complicated than the reverse, described in Section 5.4.1. This is necessarily so; while microscopic to macroscopic conversion effects a *decrease* in information, macroscopic to microscopic requires that the information in the system somehow *increase*. I propose a method inspired by the kinetic theory of gases and Poisson processes that delivers suitable results while remaining simple and efficient.

5.4.2.1 Poisson processes

The Poisson process is a stochastic procedure used to model occurrences ('arrival times') of independent events t_0, t_1, t_2,.... The number of events occurring in a given time interval is modeled by the Poisson distribution, which has the following probability mass function (pmf):

$$p(\eta, \lambda) = \frac{e^{-\lambda}\lambda^{\eta}}{\eta!} \tag{5.7}$$

where $x \in \mathbb{Z}[0, \infty)$ is the number of events observed and λ a *rate parameter* specifying the number of expected occurrences per time interval; essentially, Equation (5.7) gives the probability that exactly η events occur in over some time interval.

I use a Poisson-like process to determine the location of discrete vehicles in a continuum region given the piecewise-constant density cells ρ_k that comprise the unknowns in that region. Rather than determining how discrete *events* are distributed in *time*, I model where discrete *vehicles* are distributed in *space* — specifically, how they are specified along the 1-dimensional space of the region.

5.4.2.2 Generating events in a homogeneous Poisson process

There are a number of techniques for generating events in a Poisson process; I shall briefly introduce the general principles and present an efficient technique tailored to this problem.

To begin with, it can be shown [Devroye, 1986] that the time between events t_i, t_{+1} in a *homogeneous* Poisson process satisfy an exponential distribution with probability density function (pdf):

$$p(x) = \lambda e^{-\lambda x} \tag{5.8}$$

Precisely, Equation (5.8) gives the probability, for each $x \in \mathbb{R}[0, \infty)$, that $x = t_{i+1} - t_i$ — the time successive events in the process. So a simple algorithm for generating the i^{ith} event in a (homogeneous) Poisson process is to generate a random variable V from an exponential distribution and set $t_i = V + t_{i-1}$, where I define the base case $t_{-1} = 0$.

Exponential random variables V can be efficiently generated through the *inversion* process; that is, uniform random variables U are combined with the inverted cumulative density function (cdf) of the exponential distribution:

$$\begin{aligned} c(x) &= \int_0^x p(t)\, dt \\ &= \int_0^x \lambda e^{-\lambda t}\, dt \\ &= -e^{-\lambda t}\Big|_0^x \\ &= 1 - e^{-\lambda x} \end{aligned} \tag{5.9}$$

Setting this equal to a uniform random variable U and solving for x, we get an

exponentially-distributed random variable:

$$\begin{aligned} U &= 1 - e^{-\lambda x} \\ 1 - U &= e^{-\lambda x} \\ -\frac{\ln 1 - U}{\lambda} = -\frac{\ln U}{\lambda} &= x \end{aligned} \tag{5.10}$$

Here I have replaced $1 - U$ by U, since they are identically distributed.

5.4.2.3 Generating events in an inhomogeneous Poisson process

To properly account for the continuum data ρ_k in the lane, it is desirable to model an *inhomogeneous* Poisson process — that is, where the λ in Equations (5.8) and (5.9) is no longer constant. Indeed, it is desirable to have $\lambda(x) = \frac{1}{l}\rho(x)$, where l reflects a representative car length; this scaling converts $\rho(x)$ from cars per car length to cars per meter (i.e., to the spatial units of x).

Probability density function Above, I described how to generate successive events in a *homogeneous* Poisson process. To capture the variation in ρ that generally occurs in a continuum lane, the distribution of the separation between successive events is no longer described by Equation (5.8), but by the following pdf [Devroye, 1986] reflecting the inhomogeneous case:

$$p(x) = \lambda(\tau + x)\, e^{-(\Lambda(\tau+x)-\Lambda(\tau))} \tag{5.11}$$

Here τ is the time of the last event and $\Lambda(x) = \int_0^x \lambda(t)\, dt$.

Cumulative density function To extend the technique presented above for generating events in a homogeneous Poisson processes to the inhomogeneous case, I compute

the cdf:

$$\begin{aligned} c(x) &= \int_0^x p(t)\,dt \\ &= \int_0^x \lambda(\tau+t)\, e^{-(\Lambda(\tau+t)-\Lambda(\tau))}\,dt \\ &= -e^{-(\Lambda(\tau+t)-\Lambda(\tau))}\Big|_0^x \\ &= 1 - e^{-(\Lambda(\tau+x)-\Lambda(\tau))} \end{aligned} \tag{5.12}$$

Here I have assumed that $\lim_{x\to\infty} \Lambda(x) = \infty$.

Inversion As with the homogeneous case, I invert the cdf to achieve the formula for generating exponentially-distributed random variables the match the given $\lambda(x)$:

$$\begin{aligned} U &= 1 - e^{-(\Lambda(\tau+x)-\Lambda(\tau))} \\ 1-U &= e^{-(\Lambda(\tau+x)-\Lambda(\tau))} \\ -\ln(1-U) &= \Lambda(\tau+x) - \Lambda(\tau) \\ \Lambda(\tau) - \ln U &= \Lambda(\tau+x) \\ \Lambda^{-1}(\Lambda(\tau) - \ln U) - \tau &= x \end{aligned}$$

Recall that the above equation gives the separation time between two events with the first occurring at τ; in general, I am interested in the actual time of the new event rather than this difference:

$$\tau_i = \Lambda^{-1}(\Lambda(\tau_{i-1}) - \ln U) \tag{5.13}$$

Events in an inhomogeneous Poisson process can be generated with rate function $\lambda(x)$ provided one can compute $\Lambda(x)$ and $\Lambda^{-1}(\nu)$. For general $\lambda(x)$, this may require numerical methods for integration and inversion — computations that may be expensive and prone to issues of numerical stability. In these cases, the *thinning method*

[Lewis and Shedler, 1979] may be applied with success; this technique uses a secondary rate function $\mu(x) > \lambda(x)$ to which the above inversion process may be applied and then uses a rejection-like process to select only the random variables that satisfy the process with rate function $\lambda(x)$.

Because the rate function $\lambda(x) = \frac{1}{l}\rho(x)$ is actually given by discrete cells ρ_k, our rate function is piecewise-constant — its integral $\Lambda(x)$ and integral inverse $\Lambda^{-1}(\nu)$ are simple to compute. I will shortly give an efficient algorithm for computing the positions of discrete vehicles given a continuum region with a piecewise-constant density function $\rho(x) = \rho_k$; first I establish details of the integral and integral inverse of a piecewise-constant function.

5.4.2.4 Integral and inverted integral of continuum densities

Given a continuum region with n discrete cells $\mathbf{Q}_k = [\rho_k, y_k]^{\mathrm{T}}, k \in \mathbb{Z}[0, n-1]$ (where each cell is Δx in length):

$$\begin{aligned}\Lambda(s) &= \int_0^s \frac{1}{l}\rho(x)\,dx \\ &= \frac{1}{l}\Delta x \sum_{k=0}^{i-1}\rho_k + \frac{1}{l}\int_{i\Delta x}^{s}\rho(x)\,dx \\ &= \frac{1}{l}\left(\Delta x \sum_{k=0}^{i-1}\rho_k + \rho_i(s - i\Delta x)\right)\end{aligned} \tag{5.14}$$

where $i = \sup\{j \in \mathbb{Z}[0,n] | \Delta j < s\}$; i.e., the index of the cell 'containing' s (or one past the last grid cell, if $s \geq \Delta x n$). I restrict $s \geq 0$ and define $\rho(t) = 0$ for $t \geq \Delta x n$, and also that $\rho_n = 0$. See Figure 5.5 for a plot of $\lambda(x)$ and $\Lambda(x)$.

To use Equation (5.13) to generate events that correspond to the density ρ in a continuum region, one must invert Λ from Equation (5.14).

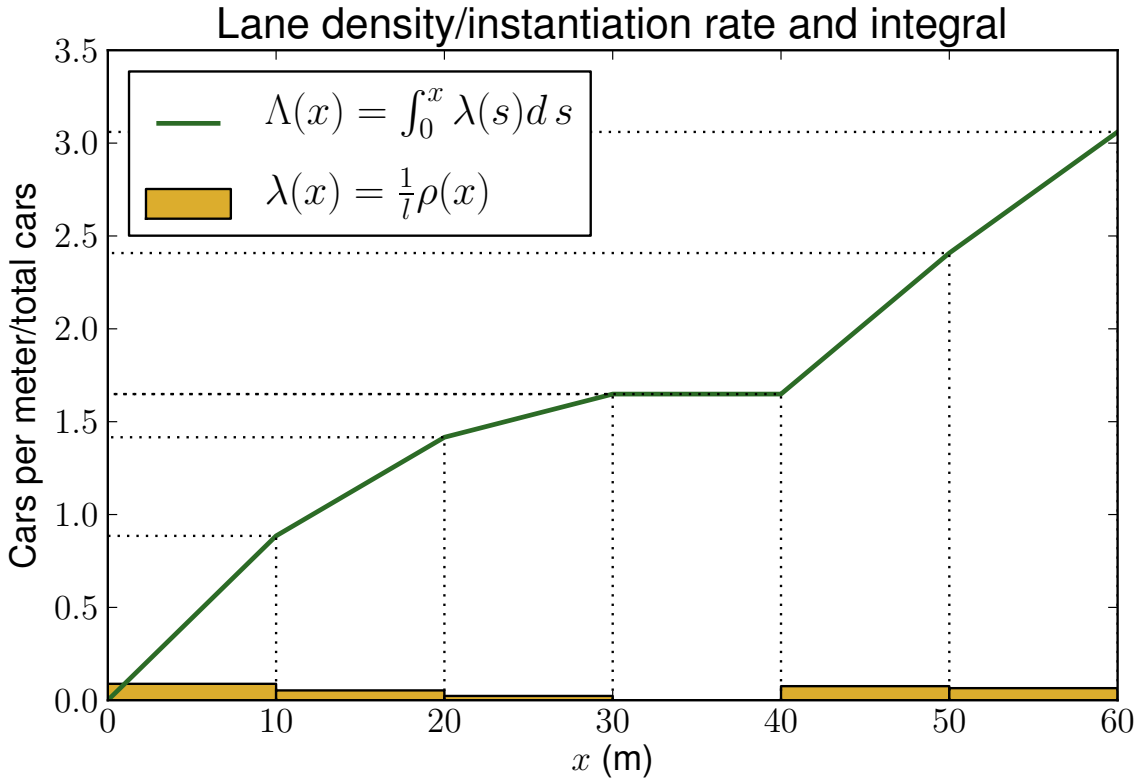


Figure 5.5: Plot of $\frac{1}{l}\rho_k$ for a lane and its integral. Exponentially-distributed random variables are mapped to the y-axis and used to locate the x-value of an event (vehicle).

Asymptotic behavior Consider this $\Lambda(x)$; in Equation (5.12), I assumed that $\lim_{x\to\infty} \Lambda(x) = \infty$. This is important because the argument to Λ^{-1} in Equation (5.13) takes its value in the range $(0, \infty)$, so the domain of $\Lambda^{-1}(\nu)$ ought to match. However, for the $\Lambda(x)$ in Equation (5.14), $\lim_{x\to\infty} \Lambda(x) = \frac{1}{l}\Delta x \sum_{k=0}^{n-1} \rho_k \neq \infty$ — this is because each continuum lane has finite length and obviously contains a finite number of vehicles. In our technique, when $\Lambda(\tau) - \ln U > \frac{1}{l}\Delta x \sum_{k=0}^{n-1} \rho_k$ in Equation (5.13) (as it must eventually, given that the events (vehicles) have strictly increasing value), I simply stop the instantiation process; this is the termination condition.

Montonicity Finally, while the $\Lambda(x)$ in Equation (5.14) is monotone (because $\frac{1}{l}\rho_k \geq 0\,\forall k$), it is not *strictly* increasing — $\Lambda^{-1}(\nu)$ is not well defined in the traditional sense. However, given the application, this issue is easily dealt with. A 'flat' spot on $\Lambda(x)$

corresponds to one or more adjacent cells $i+0, i+1, \ldots, i+m$ ($i, m \geq 0$ and $i+m < n$) where $\rho_i = 0$; conceptually, *no vehicles should be instantiated here.* Whenever $\mathbb{X} = \{\Lambda^{-1}(\nu)\}$ for any ν has cardinality $|\mathbb{X}| > 1$, define $x = \sup \mathbb{X} = \Delta x\,(i+m+1)$.

5.4.2.5 Discrete car instantiation algorithm

The process for generating discrete cars given n cells of density ρ_k, $k \in \mathbb{Z}[0, n-1]$ with spacing Δx is based on Equations (5.13) and (5.14); given a previous vehicle position p_{i-1} and a uniformly distributed random variable U, add $-\ln U$ to the integrated rate $\Lambda(p_{i-1})$ and look up the x value of this sum in Λ^{-1}; this gives p_i. When a value that exceeds the inverse integral Λ^{-1} is generated, the length of the region has been exceeded and the process terminates. The algorithm for this process is given in Algorithm 5.2;

Algorithm 5.2 INSTANTIATE-VEHICLES

```
INSTANTIATE-VEHICLES(ρ[n], Δx)
    // ρ[n] — an array of n density values, Δx — the length of each grid cell
 1  p = [ ]
 2  Λ_last, i, σ = 0
 3  while true
 4      U = UNIFORM-RANDOM-NUMBER((0, 1])
 5      Λ_last = Λ_last − ln U
 6      while i < n and σ + (1/l)ρ[i]Δx < Λ_last
 7          σ = σ + (1/l)ρ[i]Δx
 8          i = i + 1
 9      p_cand = ((Λ_last − σ) / ((1/l)ρ[i])) + iΔx
10      if p_cand > nΔx
11          return p
12      p = p + [p_cand]
```

An algorithm for vehicle instantiation from continuum data

it is based on several observations on the nature of Equations (5.13) and (5.14). First, note that the τ in $\Lambda(\tau)$ in Equation (5.13) is the argument to Λ^{-1} from the previous event — $\Lambda(\Lambda^{-1}(\Lambda(\tau_{i-1}) - \ln U_{i-1})) = \Lambda(\tau_{i-1}) - \ln U_{i-1}$. The result of this is that one never needs to explicitly compute $\Lambda(\tau)$; it was computed in the previous iteration (in

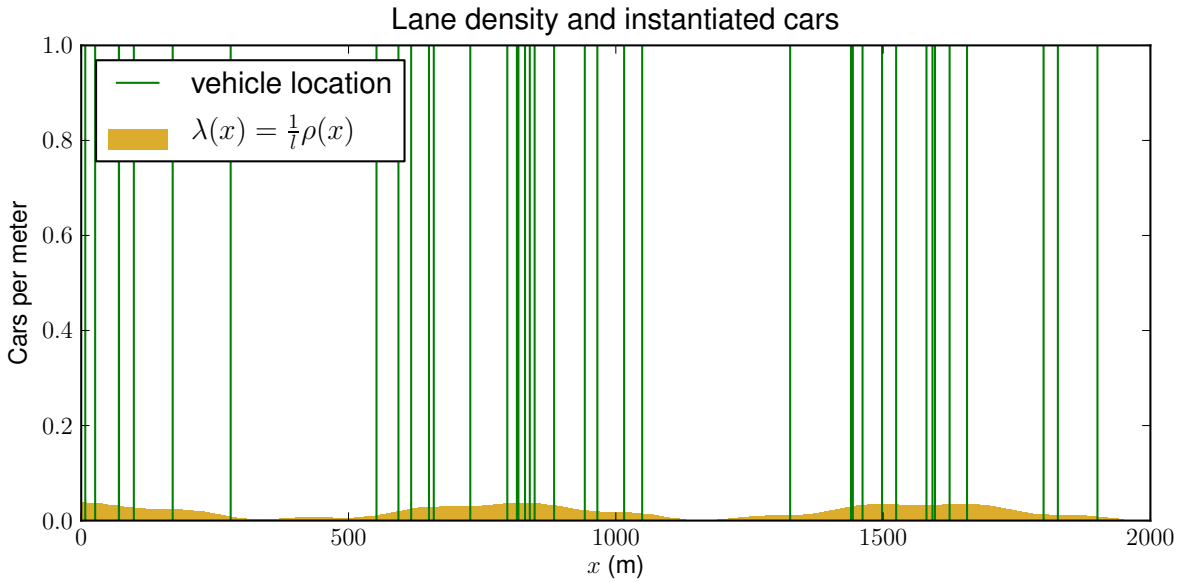


Figure 5.6: The results of running INSTANTIATE-VEHICLES() on a continuum lane; green vertical lines represent vehicle positions.

the base case, from Equation (5.14) ensures that $\Lambda(0) = 0$). A similar algorithm for generating samples is presented in [Knuth, 1997].

Furthermore, instantiated vehicles have strictly increasing x values. This is a general property of Poisson processes, and it is readily confirmed by the fact that $-\ln U > 0$ and that $\Lambda(\tau)$ is monotone and increasing. This simplifies the computation of each Λ^{-1}, since each Λ_{last} will take its value as $i\Delta x < \Lambda^{-1}(\Lambda_{\text{last}}) < \infty$, where i is index of the grid cell the last instantiation occurred in. Figure 5.6 shows the result of running INSTANTIATE-VEHICLES on a continuum lane.

Analysis The performance of INSTANTIATE-VEHICLES is $O(n + k)$, where n is the number of grid cells in the continuum region and k is then number of instantiated vehicles. The outer while loop spanning lines 3–12 in Algorithm 5.2 is clearly executed k times, and the inner loop spanning lines 6–8 will iterate no more than n times in the course of the entire execution of INSTANTIATE-VEHICLES.

It remains to bound the value of k for a call of INSTANTIATE-VEHICLES; this is a

randomized algorithm and k may theoretically be arbitrarily large (although subject to some upper bound based on floating-point arithmetic). A conservative expected value can be established as follows: consider the case where $\rho[n]$ is such that $\rho[i] = 1 \, \forall i \in \mathbb{Z}[0, n-1)$ — clearly an upper bound on any real continuum region. Then $\lambda(\tau) = \frac{1}{l}$, and since the expected value for a homogeneous exponential distribution with rate parameter λ is $\frac{1}{\lambda}$, the average is an instantiated vehicle every l meters — bumper-to-bumper traffic that precisely matches saturation of density. The total expected number of vehicles k is then $\frac{n\Delta x}{l}$, which itself is $O(n)$. Given that the estimate for k just developed is an upper bound, the expected runtime of INSTANTIATE-VEHICLES is $O(n)$.

5.4.3 Coupling

The interfaces between the continuum and discrete regions that are found in the networks can be characterized in two ways: continuum flowing into discrete, and discrete flowing in to continuum. The method must account for how 'upstream' regions affect the 'downstream' regions they flow into, and vise versa.

However, traffic is an *isotropic* phenomena — vehicles are strongly biased towards moving forward — and these two configurations must be dealt with separately: vehicles passing from one regime to another must be converted to the representation used in the destination regime, and the flow of traffic in each region must affect the region that precedes it.

The processes described in Sections 5.4.1 and 5.4.2 describe how one can *instantaneously* convert one type of region to the other, but part of the challenge here is that conversion must be performed dynamically. As I shall show, much of the previously developed material will be useful, but key modifications must be made to properly describe the transfer of vehicles.

In Section 5.4.3.1, discusses the continuum-to-discrete configuration; I introduce 'flux capacitors' to convert continuum flow to discrete agents and show how the downstream

discrete traffic is made to influence the upstream continuum. Section 5.4.3.2 describes the discrete-to-continuum arrangement: how I adapt the car averaging procedure from Section 5.4.1 to handle discrete vehicles that flow into macroscopic regions, and how discrete vehicles determine their behavior from the continuum they flow into.

5.4.3.1 Flux capacitors

Given two adjacent cells in a continuum lane, the methods described in Chapter 4 are used to solve the Riemann problem at the interface between these two cells; doing so computes the value of the unknowns at the interface: $\mathbf{q}_0$. Substituting this into $\mathbf{f}$ from Equation (4.7) gives the *flux* between the cells. When a continuum lane flows into a discrete one, flux can be used to convert density flowing out of the continuum region into discrete vehicles entering the microscopic lane, and to imbue these vehicles with velocity.

Outflow boundary conditions To compute the flux at the outflow boundary of a continuum region — which is used to create discrete vehicles leaving the region — I use a 'virtual' cell of continuum data based on vehicles near the start of the adjoining microscopic region. Precisely, temporary cell of length Δx is created at the start of the microscopic region and apply Equations (5.5) and (5.6) from Section 5.4.1. This temporary cell is used for q_r when solving the Riemann problem at the end of the continuum lane, and the resulting flux is used to send discrete vehicles into the following microscopic region.

Accumulating density In the dimensionless interval between the end of a continuum region and the start of a discrete one, I monitor the flux of vehicles and accumulate density until a sufficient quantity has been retained that a vehicle can be emitted — I call this a *flux capacitor*. Formally, the accumulated cars V_{tot} at a flux capacitor at time

t is given by the following:

$$V_{\text{tot}} = \int_0^t \frac{\rho_0(\tau)u_0(\tau)}{l}\, d\tau \tag{5.15}$$

where $\rho_0 u_0$ is the ρ-component of the flux of the intermediate state as computed in the Riemann problem (computed from $\mathbf{q}_0 = [\rho_0, y_0]^{\mathrm{T}}$, found as per Section 4.3.3.4, and substituted into $\mathbf{f}$ of Equation (4.7)). l is vehicle length; this term is necessary to scale the integrand from cars per car length (ρ) times meters per second (u) to cars per second. Because $\mathbf{q}_0$ is considered to be constant during a timestep, Equation (5.15) simplifies to:

$$V_{\text{tot}} = \sum_{i=0}^{n-2} \int_{t_i}^{t_{i+1}} \frac{\rho_0(\tau)u_0(\tau)}{l}\, d\tau = \frac{1}{l}\sum_{i=0}^{n-2} \rho_0(t_i)u_0(t_i)\,(t_{i+1} - t_i) \tag{5.16}$$

given a (nonuniform) sequence of n distinct timesteps $[t_0 = 0, t_1, \ldots, t_{n-1}]$.

Incremental flux accumulation Of course, rather than simply count their number, I wish to introduce these 'accumulated' vehicles as they occur. To this end, I maintain a V_{acc} that is updated each at the end of each timestep t_i by

$$\Delta V_{\text{acc}} = \frac{1}{l}\rho_0(t_i)u_0(t_i)\,(t_{i+1} - t_i) \tag{5.17}$$

Then, if $V_{\text{acc}} \geq 1$, emit a vehicle and set $V_{\text{acc}} = \{V_{\text{acc}}\}$, where $\{x\}$ is the sawtooth function $\{x\} = x - \lfloor x \rfloor$.

This vehicle begins with its rearmost point (the rear bumper) aligned with the start of the macroscopic lane, and its velocity is set to be u_0 during the timestep of the interval wherein the vehicle was emitted.

Preventing multiple-vehicle emission The procedure above does not account for situations where $\Delta V_{\text{acc}} > 1$; in this case, the method may be obligated to emit multiple vehicles in a single timestep. To do so, it is necessary to emit the vehicles at some reasonable distance from one another, and this brings of issue of locality — one or more

of the emitted vehicles would start well into the microscopic region, and it is difficult to ensure that there is sufficient space to avoid overlapping vehicles.

It is necessary to have $\Delta V_{\text{acc}} \leq 1$; to get a better idea of this value, Equation (5.17) can be simplified using what is known about $t_{i+1} - t_i = \Delta t_i$, which is computed according to Equation (4.6). Δt_i is the smallest ratio of the cell spacing Δx to the maximum speed $\lambda_{\max}$ across all lanes; and $u_0 \leq \lambda_{\max}$, because u is a wave speed (See Equation 4.14). Putting it all together results in following rough upper bound for ΔV_{acc}:

$$\Delta V_{\text{acc}} \leq \frac{\Delta x}{l} \rho_0(t_i) \tag{5.18}$$

This has an intuitive explanation; $\Delta x / l$ is the number of cars that can possibly fit in a cell in the current region, and ρ_0 is how 'full' of vehicles such a cell is. Given that $\Delta x >> l$ in general (the cell spacing is approximately some multiple of the vehicle length l), ΔV_{acc} is certainly not restricted to be ≤ 1.

To ensure that no more than one vehicle may be emitted per flux capacitor in a timestep, I add an additional restriction to that given in Equation (4.6):

$$\Delta t < \min_{j \in \mathbb{L}, k \in \mathbb{FC}} \left\{ \frac{\Delta x_j}{\lambda_{\max j}}, \frac{l}{\Delta x_k \rho_{0,k} u_{0,k}} \right\} \tag{5.19}$$

where $\mathbb{L}$ is the set of all continuum lanes and $\mathbb{FC}$ is the set of all flux capacitors.

In practice, the additional restriction imposed on Δt in Equation (5.19) will rarely result in a Δt smaller than that produced by from the conventional stability requirements given in Equation (4.6); there is a relationship between ρ_0 and u_0 given by Equation (4.9) that limits the maximum flux ρu that can be achieved for a given γ.

5.4.3.2 Flow averaging

The previous discussion on 'flux capacitors' describes how to translate the flow from a continuum region into discrete vehicles that may be used for microscopic simulation,

and how to account for the downstream microscopic region's effect on the upstream continuum region Now, I describe the converse: how vehicles in discrete regions that flow *into* continuum regions may be converted into the appropriate macroscopic quantities, and how the state of this continuum region can be accounted for in the upstream microscopic region.

Finding leaders in continuum regions I have discussed how a vehicle at the end of a discrete lane locates the leading vehicle that dictates its acceleration in Section 5.3.6.2. Should one encounter a continuum lane in the search, a suitable distance and velocity must be provided — the qualities of a hypothetical leading vehicle. This mechanism will allow microscopic regions that flow into continuum ones to account for the downstream dynamics.

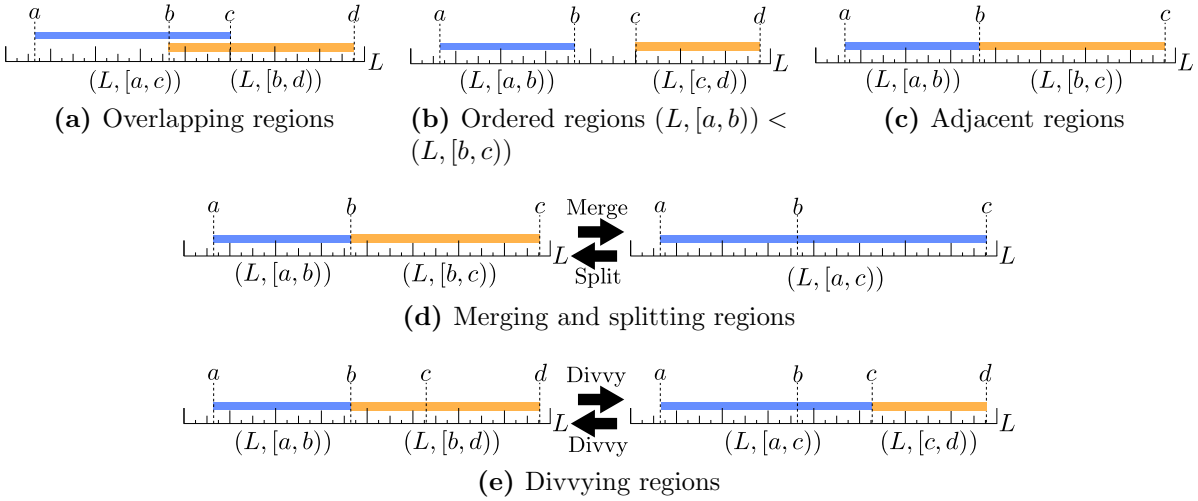
I use the vehicle instantiation procedure described in Section 5.4.2.5, except that it now terminates after finding just one vehicle This vehicle's velocity is determined by the continuum data at its location, and it is position and velocity are reported back as per the algorithm described in Section 5.3.6.2.

Not that this vehicle is not persistent; one simply uses the instantiation algorithm to determine the position and velocity of the first vehicle in the lane, report it to the leader-locating algorithm, and discard the temporary vehicle's information.

Transfer of discrete vehicles into continuum regions When discrete regions flow into continuum ones, one must convert discrete vehicles into continuum data as they pass into the new regime. I achieve this in a similar fashion to how I account for downstream discrete vehicles affect the outflow of continuum regions in Section 5.4.3.1.

A single 'virtual' grid cell at the end of the microscopic region is filled according to the averaging procedure in Section 5.4.1; this is then used as the left grid cell q_r when solving the Riemann problem associated the start of the continuum lane.

When the motion of a vehicle carries it from a discrete region to a continuum one,



(a) Overlapping regions **(b)** Ordered regions $(L, [a, b)) < (L, [b, c))$ **(c)** Adjacent regions

(d) Merging and splitting regions

(e) Divvying regions

Figure 5.7: Region operations

simply remove it from the simulation; the vehicle is already accounted for in the continuum representation and flux computation just described.

5.4.4 Simulation region selection and refinement

So far, I have discussed how I simulate traffic flow with disparate continuum and discrete methods and how the various disjoint simulation regions can be coupled to each other. I now present how these regions are chosen and evolve throughout a simulation.

Model distinctions Recall that the motivation for this simulation technique, discussed in Section 5.1, is that the two simulation techniques are complimentary. Microscopic simulation allows for easy visualization of individual vehicles and makes simulation of an inhomogeneous collection of vehicles simple, while macroscopic can be vastly more efficient (see Figure 4.6, for example).

5.4.4.1 Characterization of regions

I have frequently discussed simulations as occurring in 'regions', which may invoke images of meandering polygons or winding, closed curves. In fact, given that the simulation

domain is a network of (one-dimensional) lanes, and that handle boundary conditions must be handled on a per-lane basis (to properly account for the dynamic states of intersections, among other reasons), regions are simple linear intervals along lanes. These may well be defined as the intersection of the network with a more complicated region of the plane, but the mechanics of this technique is agnostic of these higher-level details.

Each region can be described as a tuple $(L, [a, b))$, where L indicates one of the lanes in the network and $[a, b)$ a half-open subinterval of $[0, 1)$ that represents the parametric range of L that the region covers. This simple description helps defines some simple concepts and operations with tuples, which I list here and depict visually in Figure 5.7:

Overlap Two regions $(L, [a, b))$ and $(R, [c, d))$ *overlap* iff $L = R$ and $[a, b) \cap [c, d)$ is not empty. See Figure 5.7(a).

Order We say that the pair of regions $(L, [a, b))$, $(R, [c, d))$ is *ordered* $(L, [a, b)) < (R, [c, d))$ iff $L = R$, they do not overlap, and $a < b \leq c < d$. Example shown in Figure 5.7(b).

Adjacency A pair of regions $(L, [a, b))$, $(R, [c, d))$ is *adjacent* iff $L = R$ and $b = c$; see Figure 5.7(c). Clearly such an adjacent pair of regions is in order as $(L, [a, b)) < (R, [c, d))$.

Merge We can *merge* an adjacent pair of regions $(L, [a, b))$, $(L, [b, c))$; this results in the single region $(L, [a, c))$. Shown, along with its inverse, in Figure 5.7(d).

Split A region $(L, [a, b))$ can be *split* into an adjacent pair of regions $(L, [a, c))$, $(L, [c, b))$ for any $c \in (a, b)$. This is the inverse of the aforementioned *merge* operation; see Figure 5.7(d). We can use this operation recursively to break a region into any number of sub-regions.

Divvy Given an adjacent pair of regions $(L, [a, b))$, $(L, [b, d))$, we can *divvy* them, enlarging one and shrinking the other. We choose any $c \in (a, d)$ and end up with

the adjacent pair of resized regions $(L, [a, c))$, $(L, [c, d))$. Were we to allow $c = a$ or $c = d$ and discard the degenerate region, this can be viewed as a generalization of the merge operation. See Figure 5.7(e) for an example.

Each simulation type has a set of these region tuples associated with it; I label these sets $\mathcal{C}$ (for the continuum regime) and $\mathcal{D}$ (for the discrete regime). I require that no two regions in $\mathcal{C} \cup \mathcal{D}$ overlap, that $\mathcal{C} \cap \mathcal{D} = \emptyset$, and that $\mathcal{C} \cup \mathcal{D}$ completely cover the road network.

Basic constraints on regions For practical reasons, I impose certain requirements on the minimum length of these regions in real space. The parametric length $b - a$ for a given region $(L, [a, b))$ can be mapped to spatial length given the length of the lane $|L|$; for obvious reasons, I require that this spatial region length $(b - a)|L|$ be greater than the length of a vehicle l.

For a region $(L, [a, b))$ in $\mathcal{C}$, I am more stringent and require that $a = j\Delta x_L/|L|$ and $b = k\Delta x_L/|L|$ $j, k \in \mathbb{N}, j < k$ — the region must begin and end on (distinct) computational cell boundaries. With this, it is unnecessary to deal with fractional grid cells in the continuum solvers, and additionally ensures that $(b - a)|L| \geq \Delta x_L >> l$.

These are merely general lower bounds on the lengths of regions; in practice, one will generally wish for continuum regions to cover many grid cells to maximize the efficiency gain of continuum simulation.

Mechanism The operations and material above deals with geometric and topographic changes to regions, but there is an obvious further considerations to be made; one must be able to move regions between $\mathcal{C}$ and $\mathcal{D}$ — that is, convert the simulation type of a region. I have shown how the simulation data is converted in Sections 5.4.1 and 5.4.2; this is the *convert* operation, which uses these techniques to change continuum regions to discrete ones, and vise versa.

One can transition between any two configurations $\mathcal{N} = (\mathcal{C}, \mathcal{D})$ and $\mathcal{N}' = (\mathcal{C}', \mathcal{D}')$ through successive applications of these *merge*, *split*, *divvy*, and *convert* operations.

5.4.4.2 Criteria

I have described above in Section 5.4.3 how two distinct simulation methodologies for traffic simulation can be coupled to one another. This is desirable because one is then able to take advantage the generally complimentary nature of these two simulation regimes.

Recall that continuum techniques can efficiently handle both large and dense areas of networks at the expense of being coarse-grained and incapable of describing truly individual vehicles, whereas agent-based simulators are highly customizeable and flexible but are typically slower than continuum methods.

There are numerous criteria one may wish to consider when choosing which portions of the network should be simulated with macroscopic or microscopic techniques. Here, we discuss some of the qualities that make a hybrid technique desirable and present such criteria may be used to select which portions of the network will undergo various modes of simulation.

Spatial When concerned with visualizing vehicles in the network, one must consider what areas of the network are visible and ensure that only microscopic simulation is used there — recall that there is no 'carticle' mechanism for capturing discrete vehicles in continuum areas as in Chapter 4, and therefore no time-coherent way to depict discrete vehicles given the density in a continuum region.

To this end, I force the visible portion of the network to be covered by microscopic regions in my simulations. Considering the largely planar character of road networks, the visible portion of the network can range from a rectangular area associated with an overhead 'bird's-eye' view to a trapezoid associated with a view frustum. To properly

handle secondary light transmission effects such as reflection and refraction, the visible portion of the network can grow to be noncompact and arbitrarily complex, but the difficulty there lies in visibility calculations.

Performance Based on the performance needs of the simulation, regions that are particularly expensive to compute in one regime can be converted to the opposing. More formally, every region can be assigned an estimated computational cost of the form:

$$w_T(r) = \alpha_T |r| + \beta_T C^r \tag{5.20}$$

where r is a region, $|r|$ the region's length, and C^r the number of vehicles in that region. α_T and β_T are weights associated with the computational regimes $T = \{\text{macro}, \text{micro}\}$. These can be determined empirically during computation; based on what the know of the two regimes, at least $\alpha_{\text{macro}} >> \beta_{\text{macro}}$ and $\beta_{\text{micro}} >> \alpha_{\text{micro}}$. Given a pair of weights $w_{\text{macro}}(r)$ and $w_{\text{micro}}(r)$ for each r and a computational 'budget', we can assign convert regions using an inexpensive partitioning scheme, such as a greedy algorithm or polynomial approximation to the minimum makespan problem (see Appendix D for a brief discussion of such a scheme).

Equation (5.20) is certainly just an estimate of the cost of simulating a region with a particular regime and a simple one at that; if more detailed information on the expense of computation is available — perhaps with regard to the movement of traffic and the associated future costs — such information can be incorporated in the model. What I have demonstrated here aims to strike a balance between utility and simplicity; a very complicated weight function could itself expensive to evaluate and work contrary to our performance-minded goals.

Feature-capturing Another useful class of criteria arises from the desire to capture specific features and types of flow with the simulation technique. An example of this is

when one wishes to track the movement of a specific vehicle or group of vehicles through the network. Because the macroscopic regime aggregates the behavior of vehicles, were a specific vehicle to be absorbed into the continuum regime, either through the transfer process described in Section 5.4.3.2 or the region conversion process described in Section 5.4.3.2, the specific information associated with that vehicle would lost. Similarly, should we wish to capture the behavior of inhomogeneous vehicles, we must resort to the microscopic model, which allows varied vehicle behavior.

To effect this, any regions containing such features must not be converted, and furthermore, whenever such a feature of interest is about to transition into a region governed by the other (unsuitable) regime, that downstream region must be converted to the upstream regime.

Additionally, there are times when we wish to satisfy certain numerical conditions governing the nature of the simulation itself. For example, in the kinetic theory of gases, there is a dimensionless quantity known as the *Knudsen number* that can be used to classify fluid behavior as a function of the fluid state itself and the scale of observation. Precisely, the Knudsen number is given by

$$Kn = \frac{\lambda}{L} \tag{5.21}$$

where λ is the mean free path of a particle and L the characteristic length scale of the problem. The prevailing in wisdom gas simulation is that continuum models are valid in the range $Kn < 0.01$, statistical models are valid when $Kn < 0.1$, and discrete models are valid at all scales; see [Hirschfelder et al., 1964] for details.

A rigorous investigation of the applicability of the Knudsen number to continuum traffic simulation has not been performed, but should the user wish to ensure that the above scheme is satisfied, it is straightforward to compute the Knudsen number for each lane and enforce those that have sufficiently large Knudsen numbers to obey the

microscopic regime.

5.5 Results

5.5.1 Benchmarks

I have demonstrated my hybrid technique for interactive visual simulation of large-scale traffic on the following scenarios:

Virtual downtown of a metropolitan area I recreated a 'virtual downtown' based on the "The Grid" sequence in Godfrey Reggio's film *Koyaanisqatsi* [Reggio, 1982]; a particular shot in this film shows traffic moving along a busy city street, punctuated by the rhythmic cycling of traffic signals and cross-traffic. This scene is located 52 minutes, 3 seconds into the theatrical release of the film, and can be viewed on YouTube: `http://www.youtube.com/watch?v=Sps6C9u7ras#t=0h52m03s`. A still of my animation can be seen in Figure 5.1.

Augmented satellite street maps One of my impetuses for this work was to be able to interactively visualize real-world traffic simulated using live traffic data to augment online virtual worlds, such as Google Earth, Microsoft Virtual Earth, or Second Life, or to enhance mobile geographic information systems, such as car navigation systems for PDAs or on-vehicle GPS systems.

In Figure 2.3, I show some example road networks created using my in-house geometric modeling system, which are then used as the simulation domain in my real-time visual simulation of metropolitan-scale traffic — typically at scales of tens of thousands of vehicles. The models illustrated here were created using GIS data from the Open Street Map website. My hybrid technique is able to recreate real-time traffic flows and individual vehicle motion on complex, metropolitan-scale road networks, which can then be visualized atop aerial imagery.

(a) A low-angle view

(b) A top-down view

Figure 5.8: A city scene filled with traffic simulated with my technique

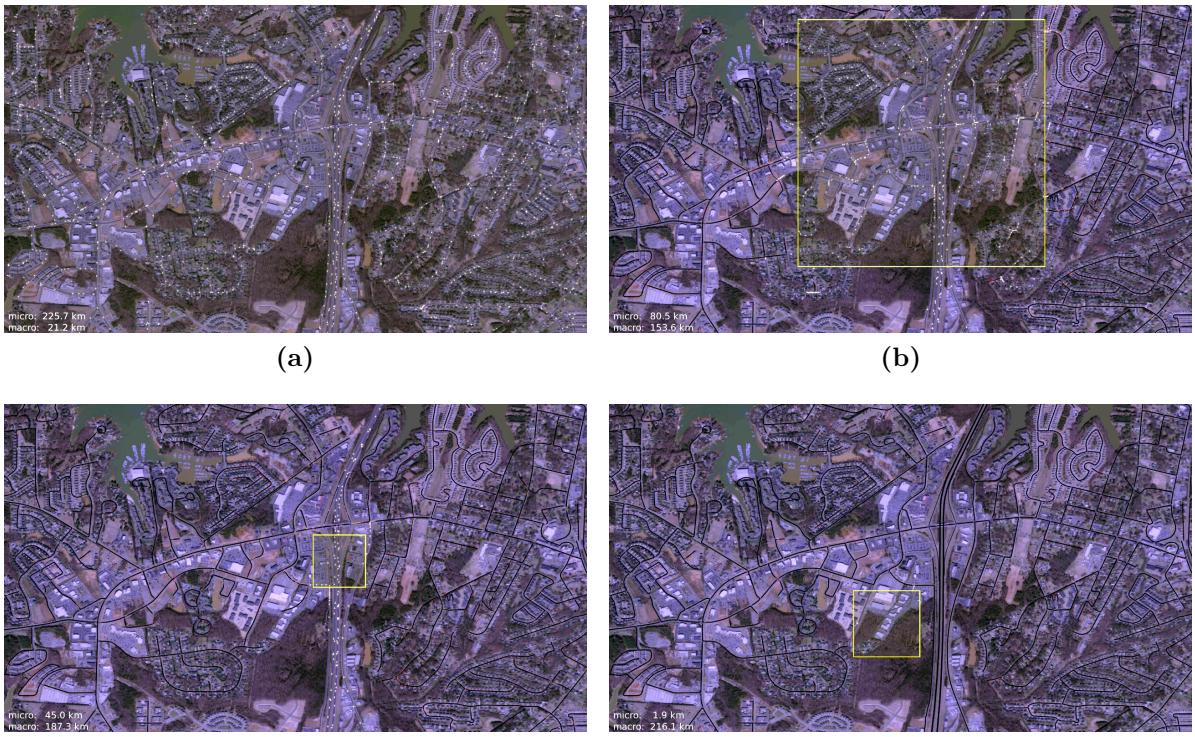
Figures 5.9(a)–(d) and the corresponding sequence in the accompanying video show dynamic region refinement based on the movement of a 'region of interest' — a yellow rectangle. Typically, this region of interest would be the visible portion of the scene, allowing my technique to perform simulation in the off-camera areas without incurring the expense of agent-based simulation everywhere.

Although this concept of augmented street maps bears close resemblance to some features in the work by [Kim et al., 2009], their approach to augment aerial earth maps with traffic information requires setting up many video cameras closely located on freeways in order to reproduce the spatial extent and aerial coverage of traffic visualization that I am able to recreate here. With my approach, commonly available live traffic data from sparsely located cameras or merely cheap in-road sensors (e.g. inductive loops) would be sufficient to initialize my simulation method for real-time traffic visualization. The two approaches are, however, complimentary; and my work can be easily integrated into their overall Augmented Reality framework.

5.5.2 Performance

One of key objectives for this work is to facilitate the cooperative use of disparate simulation strategies — agent-based and continuum traffic simulation — in a traffic network for extensive metropolitan areas. There are numerous reasons that this is desirable: continuum techniques have performance advantages over agent-based simulations in many situations — their computational cost is proportional to size of the network, not to the traffic therein. Furthermore, the limited and regular memory access patterns of continuum algorithms are much more amenable to scalable parallelism than those found in agent-based algorithms.

Figure 5.10 shows the performance of single-thread agent-based, continuum simulation, and my hybrid technique where for a city road network with a variety of vehicle densities. Purely continuum simulation outperforms purely agent-based by 2.4x-9.2x and



(a) **(b)**

(c) **(d)**

Figure 5.9: A sequence of images illustrating simulation region refinement. (a): Initially, the whole road network is simulated with agent-based techniques. (b)-(d): Later, only roads whose bounding box intersects the yellow box are simulated with agent-based techniques — continuum techniques are used elsewhere. The averaging and instantiation methods of Sections 5.4.1 and 5.4.2 handle changes due to the movement of the rectangle, while the coupling techniques described in Section 5.4.3 seamlessly integrate the dynamics of different simulation regimes.

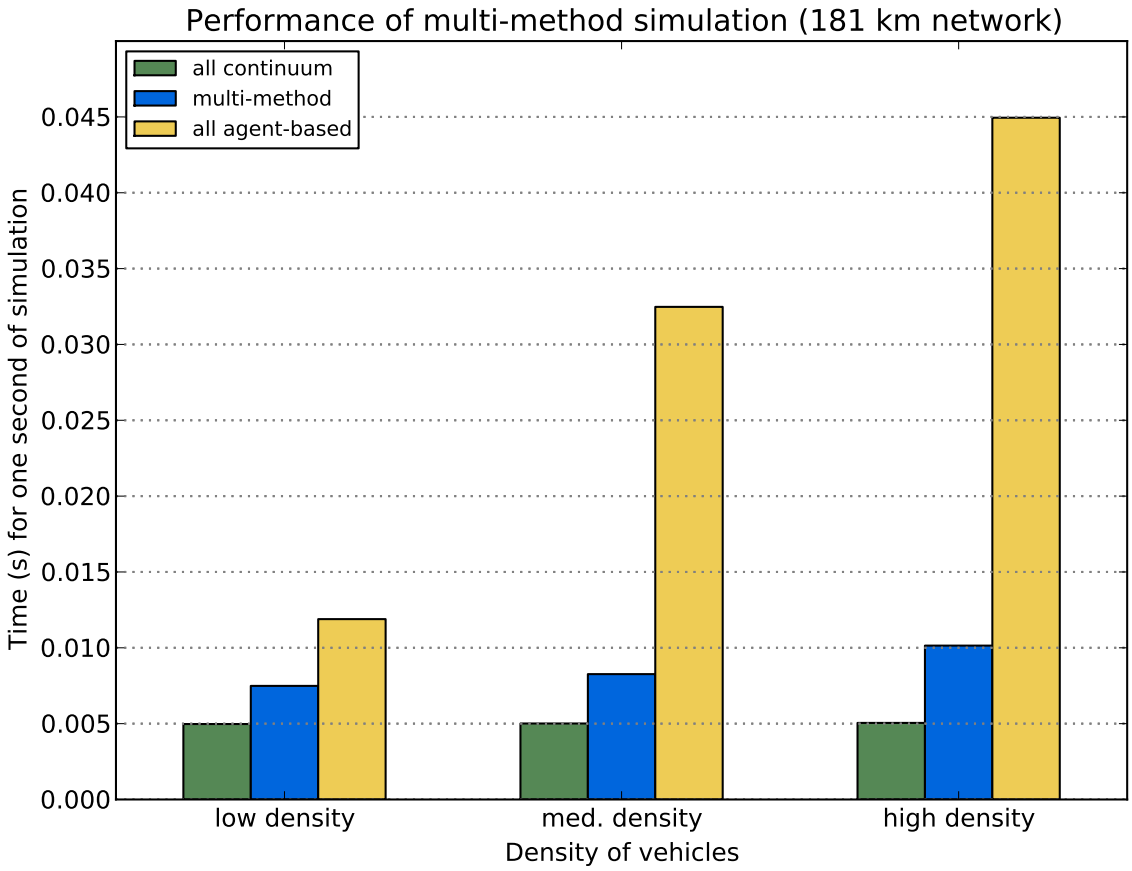


Figure 5.10: Performance of all-continuum simulation, my hybrid technique, and wholly agent-based simulation for various densities on a road network with 181 km of roads

my hybrid scheme (in this case, a constant 10% of the road network is simulated with agent-based, and the rest with continuum) is 1.5x-4.4x faster than pure agent-based. These results were collected on an Intel Core i7 980X 'Westmere' processor running at 3.33GHz.

5.5.3 Comparison with real-world data

It is useful to understand how the traffic motion produced by the method described in this chapter compares to the motion of real-world traffic. However, this comparison must be performed with deliberation and care, as there are numerous subtleties at play. Here, I explain the type of real-world data that is available, discuss the deficiencies

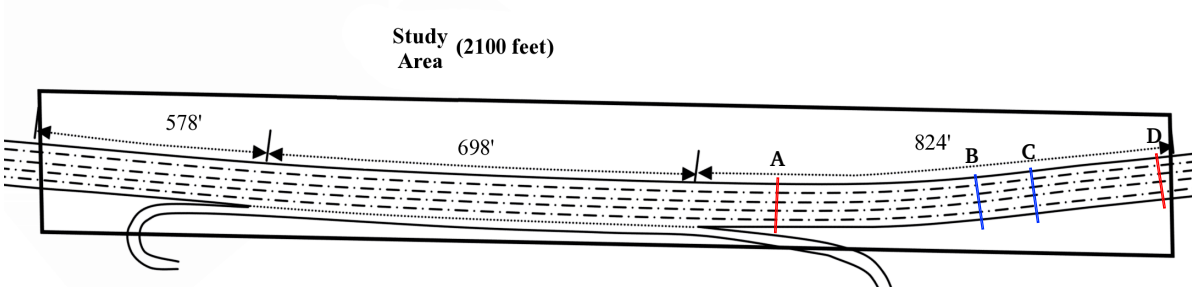


Figure 5.11: Section of Highway 101 near Los Angeles. The black rectangle indicates the region where the NGSIM project has recorded trajectories. The red lines (the interval $\overline{AD}$) denote the clipped region used in this comparison, while the blue lines (interval $\overline{BC}$) denote the macroscopic simulation region used in the hybrid simulation test.

therein as well as the limitations of my method in the presence of real-world data, and present my methodology and results for comparing my simulation technique with said real-world data.

5.5.3.1 The NGSIM project

Despite the ubiquity of traffic in the real world, data suitable for comparison with simulation techniques can be hard to find. Data must be collected for reasonably long periods of time — several minutes at the very least — and correspondingly large spatial scales. It is challenging to set up sensors for these scales, and the problem is made all the more difficult the necessity of obtaining accurate and precise measurements.

Fortunately, reasonably high-quality data has been made available by the Next-Generation Simulation project (NGSIM) [Alexiadis et al., 2007]. This is a joint project between the Federal Highway Administration and a number of partners created to provide useful data to serve the needs of the simulation common. The data provided is per-vehicle trajectory information for relatively short segments of major highways in California.

Data format The NGSIM highway data is organized as a series of time entries each specifying the position, the magnitude of velocity, and the magnitude of acceleration of

a given vehicle, among other quantities. A unique identification number associated with each frame is given to identify which vehicle the data describes. Curiously, each frame also specifies the length and width of the vehicle, as well as a discrete 'type' specifying if the vehicle is a motorcycle, truck, or passenger car. For for the data sets I have examined, these remained (blessedly) constant. Each frame also identifies a discrete 'lane number' that the vehicle is traveling in.

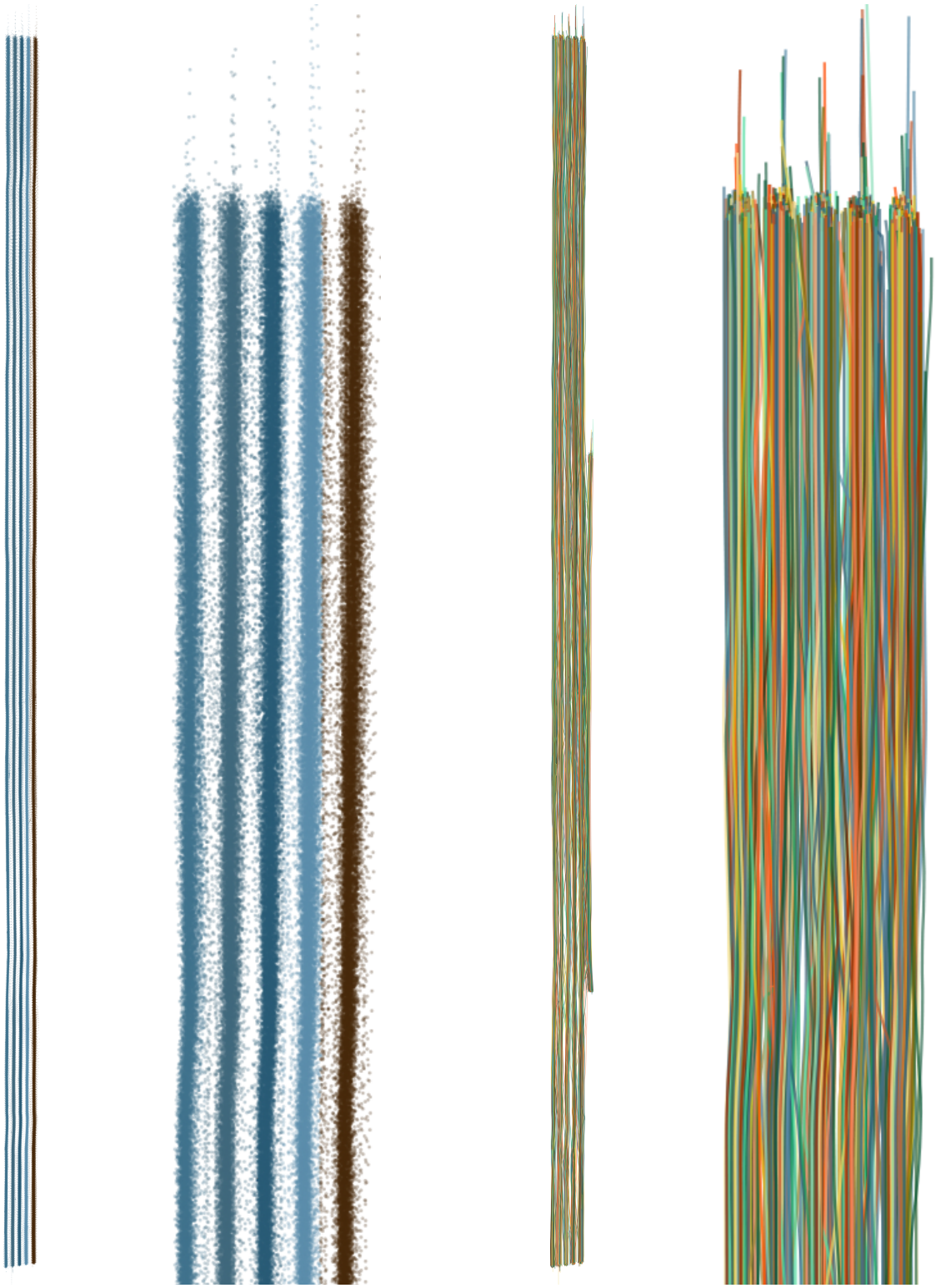
All data is given at 10 frames per second, and this is aligned globally — the position of each vehicle in the system is updated every $\frac{1}{10^{\text{th}}}$s of a second of a global clock. The data for each highway is broken up into several 15-minute intervals. Position information is given in feet to a precision of three digits, while velocity and acceleration data is given to a precision of two digits.

Figure 5.11 shows the layout of a section of highway 101 near Los Angeles; Figure 5.12 visualizes the data in for a segment of US-101 in Los Angeles, CA: Figures 5.12a and 5.12b shows each spatial sample in the dataset colored by lane — some quantization is apparent in the data. Figures 5.12c and 5.12d show the actual trajectories for each vehicle plotted in a variety of colors.

Issues It is very useful to have the NGSIM data, although it presents several challenges as well. First, from a validation standpoint, a notion of the accuracy of the measurements in the data would be useful. Unfortunately, no such estimate is given. Additionally, while the given spatial precision of three digits in feet is more than enough, the frame rate of 10 frames per second is quite poor for highway traffic. As mentioned in Section 5.3.4, the timestep required for simulation can easily be $\frac{1}{30^{\text{th}}}$ of a second.

Furthermore, the format of the acceleration and velocity data — as scalar magnitudes rather than vector data — is somewhat limiting, since vehicles changing lanes and merging have non-trivial velocity and acceleration vectors.

Finally, while the black box in Figure 5.11 shows the region where the NGSIM project



(a) Scatter plot of position samples

(b) Scatter plot of position samples (detail)

(c) Trajectories

(d) Trajectories (detail)

Figure 5.12: NGSIM US-101 dataset

recorded data, not all vehicle trajectories begin or end where the box actually intersects the highway; some start as much as 30 meters from the beginning of the highway. In order to perform a meaningful comparison, it is necessary to clip all trajectories to the smallest region of the highway where there is data for all vehicles, which somewhat reduces the total amount of trajectory data to work with.

5.5.3.2 Methodology

As discussed previously in Section 5.5.3.1, there is real-world data readily available for comparison with simulation techniques, albeit with some deficiencies. Here, I discuss my approach to achieving a meaningful comparison of the results of my simulation technique and real-world data.

Comparison strategies Numerous approaches to validation are taken in the simulation community; these are often developed based on the quality and completeness of real-world data with which to form a comparison. Given a timeseries of real-world data $\{\eta_0, \eta_1, \eta_2, \ldots\}$, the most straightforward approach is to initialize the simulator with some initial condition $\{\eta_0\}$ and generate the simulated successor states $\{\eta'_0 = \eta_0, \eta'_1, \eta'_2, \ldots\}$.

Such a simple scheme falls prey to a number of complications, ranging from granularity and measurement error in the initial data set to known deficiencies with the simulation technique. Additionally, many classes of phenomena are highly chaotic — even small perturbations in the initial conditions are greatly magnified in successive time steps. Such phenomena is exceedingly difficult to directly compare in a meaningful fashion.

Often, specific qualities of phenomena are tested; simple initial conditions are chosen that are expected to evince certain characteristic features of phenomena — for example, the 'lid-driven cavity' for fluid dynamics described by [Burggraf, 1966], or 'arching' behavior in crowd dynamics (see [Guy et al., 2010]).

For agent simulation techniques, such as traffic or crowd simulation, direct trajectory comparisons are usually not performed due to the chaotic nature of the phenomena. Rather, comparisons of averaged velocity and traffic volume over time are common, as in [Treiber et al., 2000]. A cross-sectional region of the road is designated as a 'detector' and the number of vehicles crossing said region are counted along with their average velocities over a series of non-overlapping time spans. It is also sometimes useful to measure the total *flux* of vehicles — the product of the average velocities for all vehicles and the total number of vehicles over an interval. In fact, this is the only quantity that can be directly extracted from macroscopic simulation; the corresponding results in Section 5.5.3.3 plot flux in addition to the other primitive quantities.

Measuring effectiveness of coupling The hybrid simulation technique presented in this chapter consists of both microscopic and macroscopic traffic simulation coupled together through a mechanism described in Section 5.4.3.

The extent that the base simulation methods — microscopic and macroscopic — are able to match the real-world data gives a baseline for comparison. Combining these with my hybrid technique will have some effect on the quality of this match — this change indicates the effect the hybrid method has on a basic simulation.

To gauge its effect on the ability of the underlying simulation methods to match real-world input, I have performed three separate simulations, each of which is compared against the real-world data.

Microscopic Pure agent-based simulation is carried out on a section of highway corresponding to the domain of the real-world data — $\overline{AD}$ in Figure 5.11. Vehicles are introduced to the simulation according to the time in which the enter the corresponding section of the real highway, with the appropriate lane and velocity information. Detectors spaced along the highway — as described above — record crossing times and velocities for the simulated agents (vehicles), which are then

accumulated over short intervals.

Macroscopic The same area of highway used for agent-based simulation in the previous simulation is instead used for continuum simulation. Vehicles are introduced as densities and velocities in the appropriate lanes according to the crossing times. Similarly, detectors are employed to compare results against real-world data. However, while agent-based and the real-world trajectory data lend themselves to direct computation of vehicle crossings, the nature of macroscopic simulation dictates that we measure the *flux* of vehicles across detectors — that is, $f = \rho u$. Averaged over intervals, this may be meaningfully compared against the product of crossing densities and velocities from trajectory data.

Hybrid The highway segment used for the simulations above is now divided into *three* segments — a microscopic region including the incoming boundary, an adjacent macroscopic region that occupies the central potion of the highway segment, and an abutting additional microscopic region that includes the outgoing boundary — respectively, $\overline{AB}$, $\overline{BC}$, and $\overline{CD}$ in Figure 5.11. Vehicles are introduced to the first region just as in the microscopic simulation setup, and my hybrid technique converts vehicles into the neighboring macroscopic region, where they eventually are converted back into discrete vehicles in the succeeding microscopic region. A detector in this final region records crossing times and velocities as in the above microscopic setup.

Boundary conditions Simply supplying initial conditions to a simulator is generally not sufficient if one hopes to achieve a meaningful comparison; we must consider what forces outside the simulation are affecting it. The shape of the highway certainly has an effect; thankfully this is fairly clear from the original data and easily accommodated in the simulation techniques being examined.

Of larger concern is the traffic that is *not* present in the real-world trajectory data;

the NGSIM data provided is simply a snapshot of real traffic on a highway — there is necessarily a beginning and an end, and in particular, the leading vehicles in the provided data are behaving according, to some extent, on the behavior of vehicles unknown to us — vehicles that occupy the highway at the start of the simulation, or vehicles further 'down' the highway than the domain captures. It is important to somehow capture the effect of these vehicles on the simulation; in both the agent-based and macroscopic simulation techniques examined in this thesis, vehicles with unlimited headway — no vehicles ahead of them — will simply accelerate until they reach their maximum velocity; generally, the speedlimit of the road.

To account for these unknown leading vehicles, I simply set the velocity of the leading vehicle in each lane in the simulation (leading computational cells, in a macroscopic simulation) to the velocity of the leading vehicle of the corresponding lane in the real-world data.

5.5.3.3 Comparison results

The histograms from the above simulations/real-world data are found in Figures 5.13(a)–(g). Interpretations of these plots follow:

Microscopic Overall, the microscopic simulation matches the NGSIM data quite well; for $x = 429$ (Figure 5.13(a)), we expect all quantities to match quite well because we are close to the incoming boundary condition. Detectors further down the highway ($x = 440$–542; Figures 5.13(b)–(e)) continue to match the vehicle count well, but it is evident that the microscopic simulation technique we are using has a tendency to aggressively adjust vehicles to their preferred velocity. The final two detectors ($x = 598$ and $x = 620$; Figures 5.13(f) and 5.13(g)) again match velocity well as we approach the outgoing boundary condition, where we impose the outgoing velocity of the leading vehicle from the NGSIM data. In these final stages, we see some time shifts in features we identify from the NGSIM data.

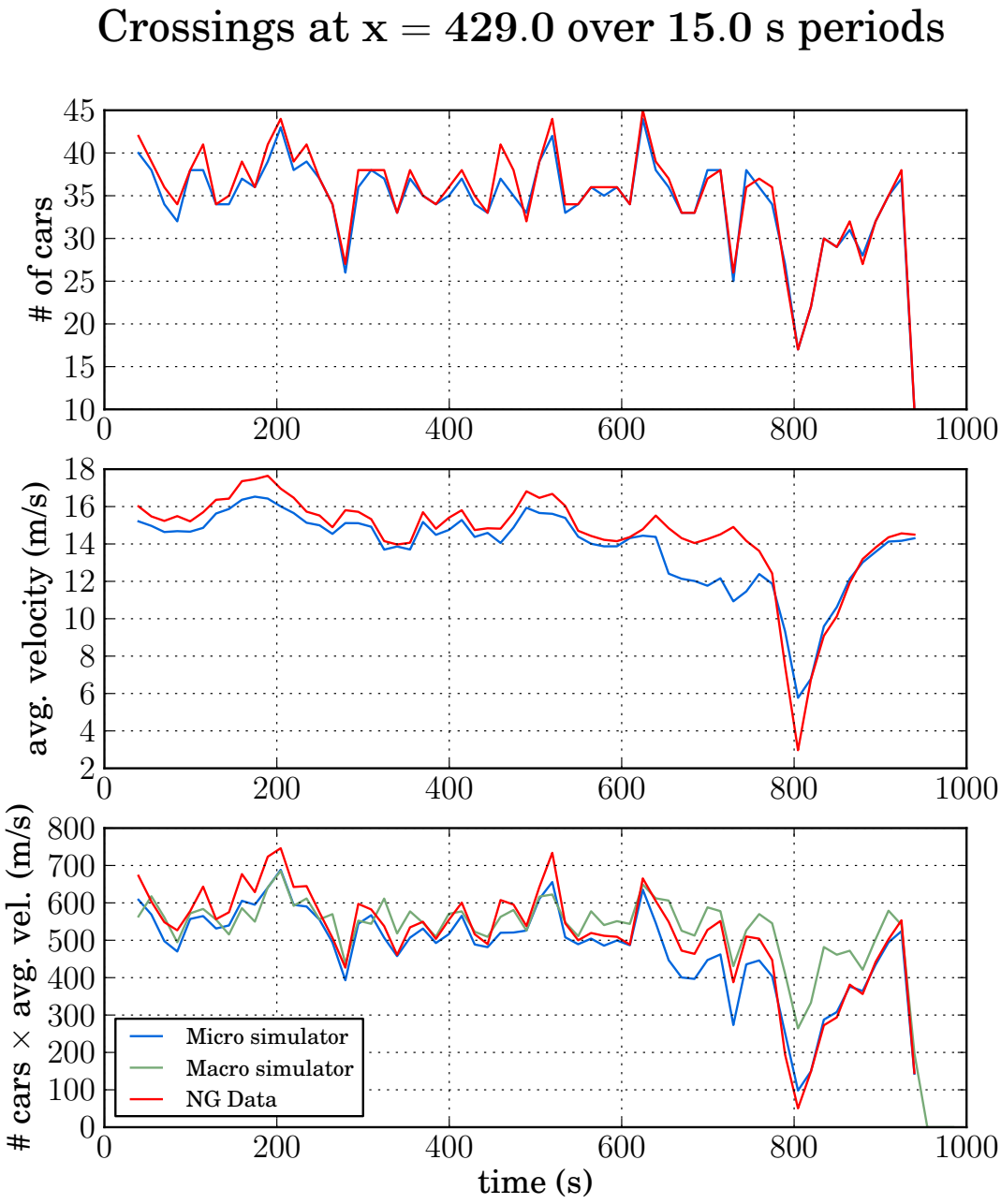
These can be attributed to the differing preceding velocity values.

Macroscopic The results of the macroscopic simulation are fluxes. As with microscopic we see good matches for the initial detectors and drift accumulating further down the highway. However, despite shifts in overall magnitude and phase, the overall patterns remain, showing that the primary features of the flow are preserved — in particular, the trough in flux that occurs around $t = 800$ at all detectors is preserved.

Hybrid Because of the final leg of the hybrid validation scheme is microscopic, we have data for the number of cars, velocity, and flux for the final detector ($x = 620$, Figure 5.13(g)). Velocity matches quite well as with the pure microscopic case on account of the outgoing boundary condition. The vehicle counts demonstrate the same features as those found in the original NGSIM data, albeit with slight variations in magnitude.

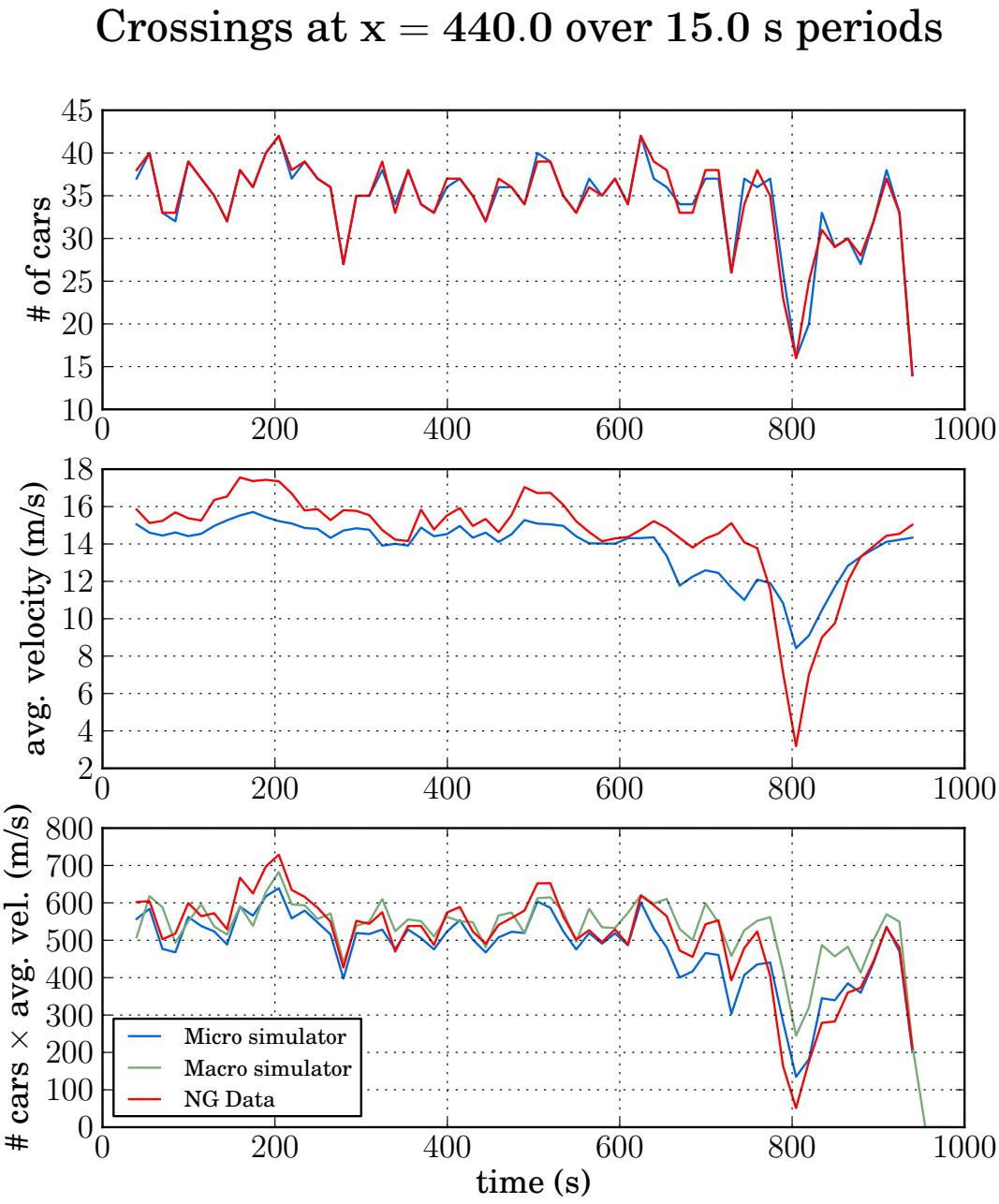
Sequence comparison Computing the number of vehicles and average velocities for the real-world data and results of the corresponding simulation results in two sets of timeseries data — number of cars, average velocity, and flux for both real-world and simulation. Examining these visually can be illuminating, but we would like a more quantitative way of comparing these data. A standard infinity– or 2– norm might seem an obvious choice, but these will rather harshly penalize noisy and time-shifted data. As discussed in [Chen et al., 2005] and [Morse and Patel, 2007], suitably modified string distance metrics such as longest common subsequence and edit distance are very useful when comparing timeseries.

As the name implies, given two sequences R and S — not necessarily of the same length — longest common subsequence (LCSS) reports the length of the largest subsequence they share. By normalizing this subsequence length by the length of the shorter



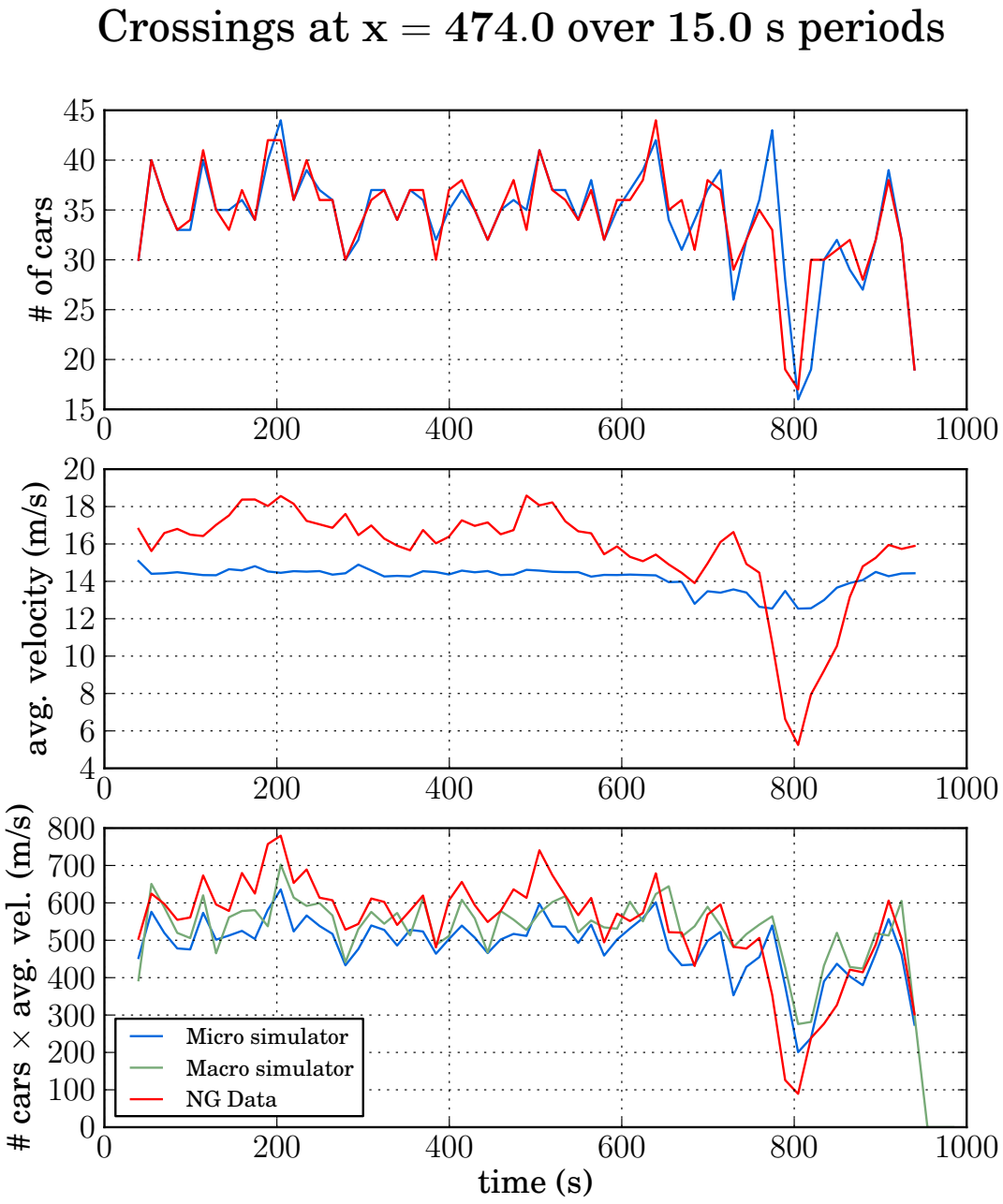
(a) Crossings at $x = 429$

Figure 5.13: Validation results, histogram width 15



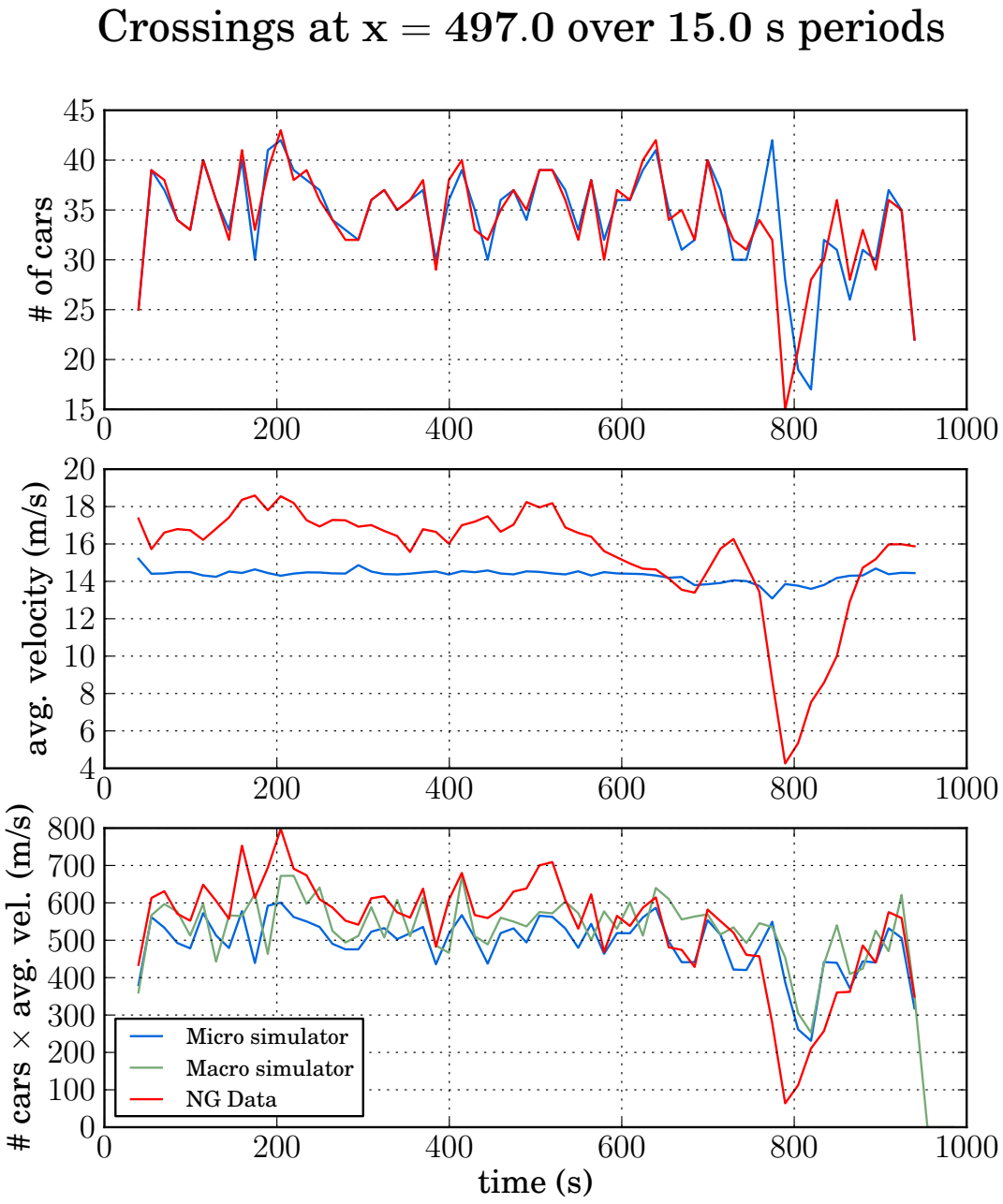
(b) Crossings at $x = 440$

Figure 5.13: Validation results, histogram width 15



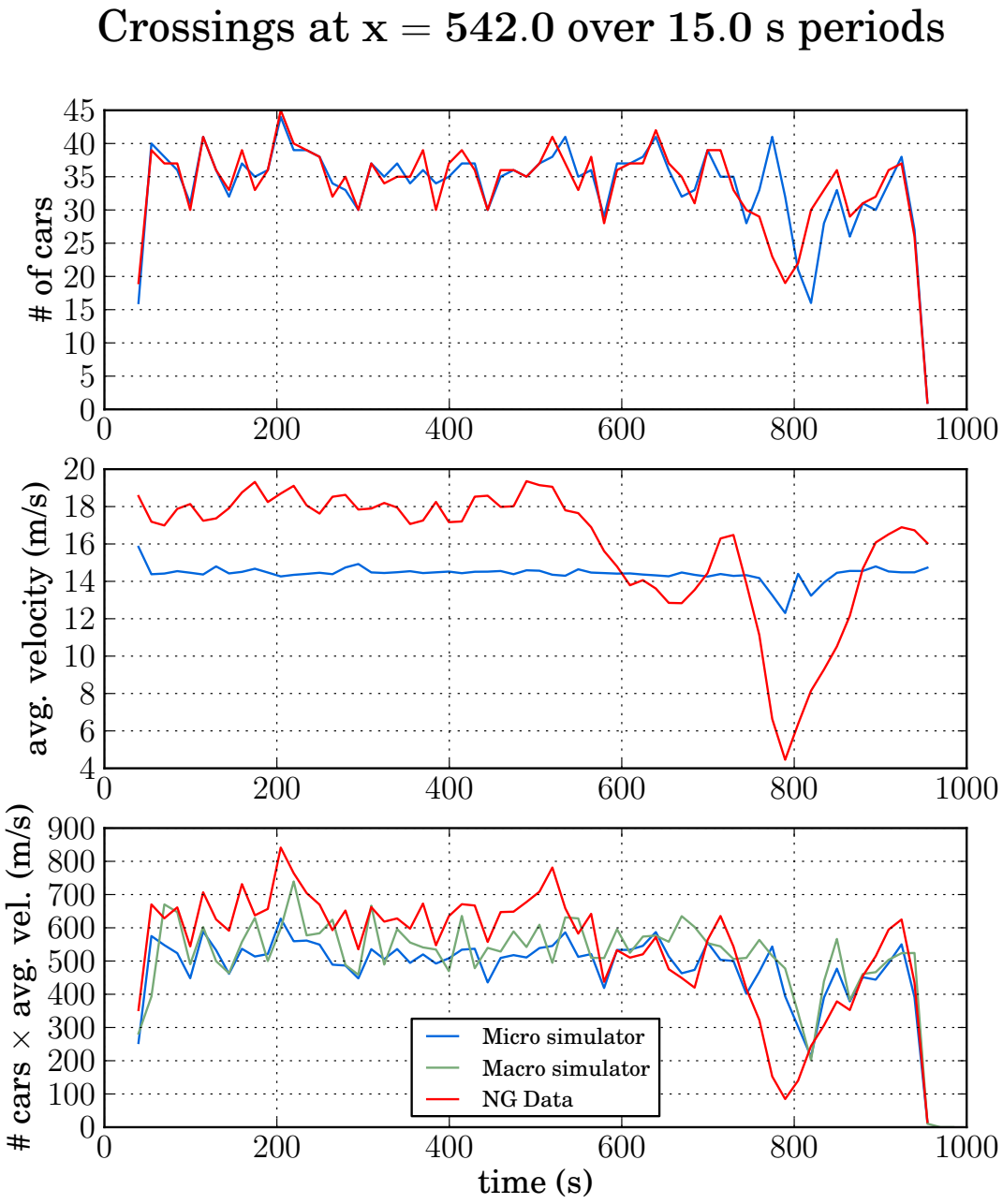
(c) Crossings at $x = 474$

Figure 5.13: Validation results, histogram width 15



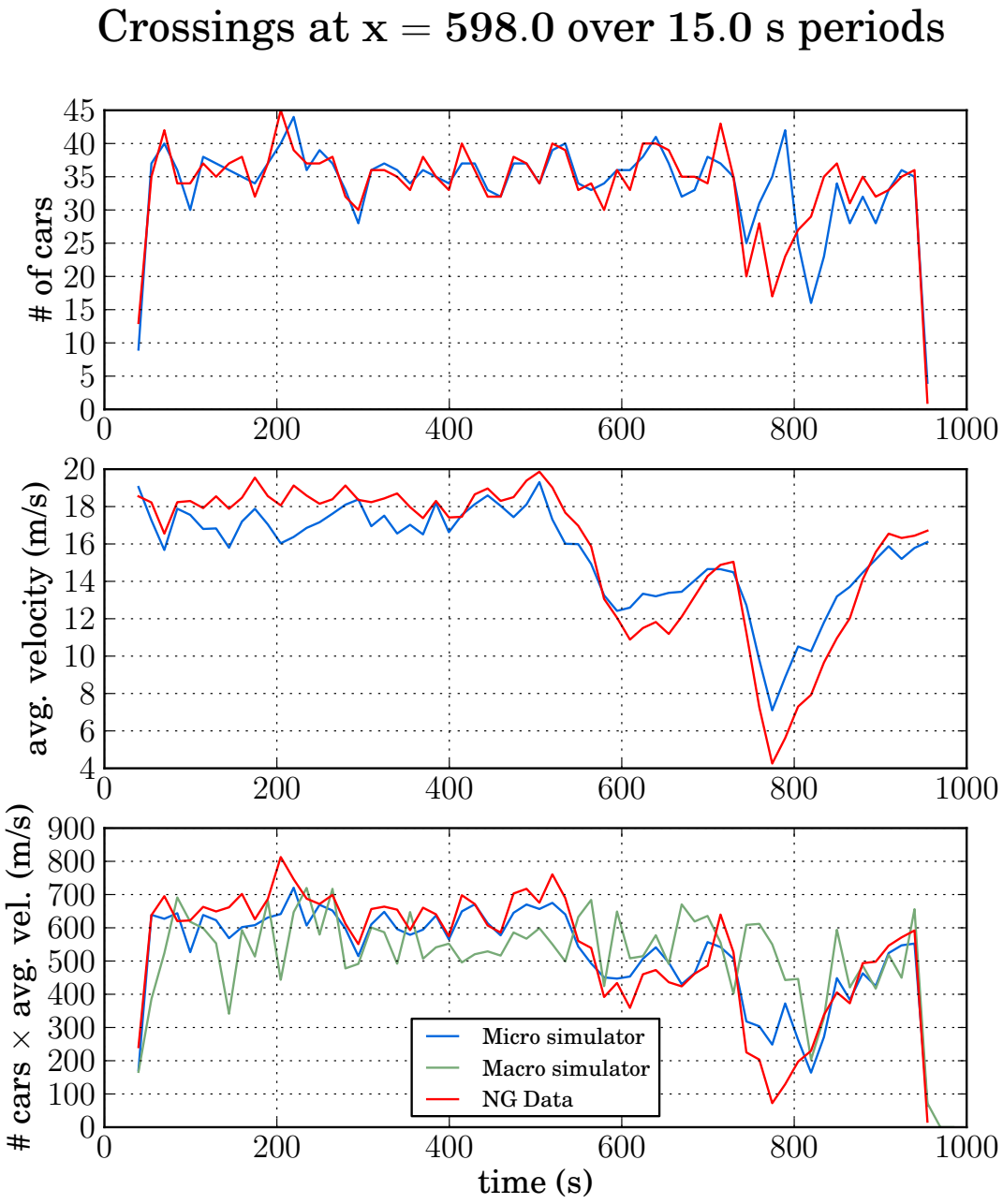
(d) Crossings at $x = 497$

Figure 5.13: Validation results, histogram width 15



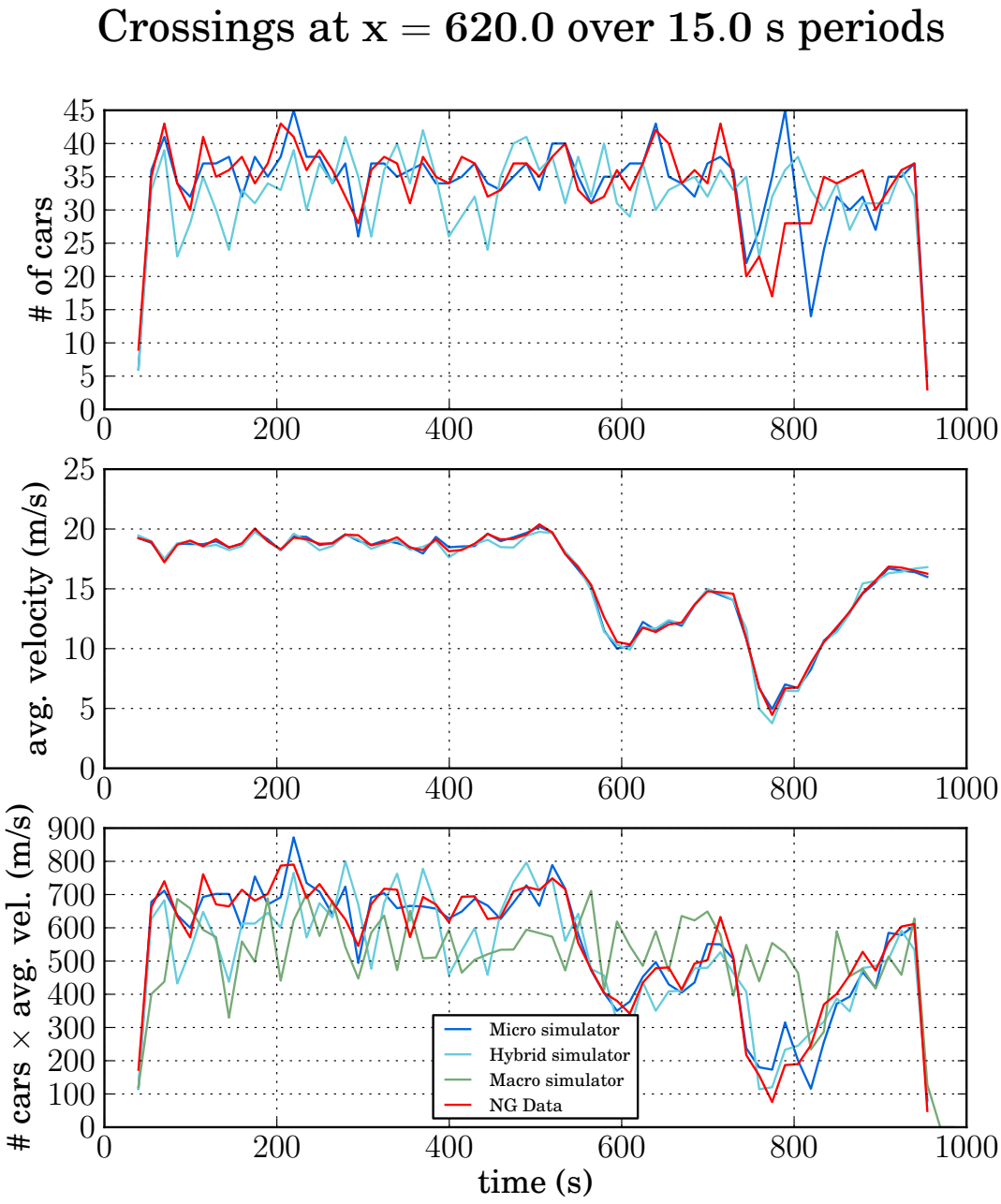
(e) Crossings at $x = 542$

Figure 5.13: Validation results, histogram width 15



(f) Crossings at $x = 598$

Figure 5.13: Validation results, histogram width 15



(g) Crossings at $x = 620$

Figure 5.13: Validation results, histogram width 15

x	metric	micro	macro	hybrid
429	flux LCSS	0.833	0.850	
	flux EDR	0.900	0.870	
440	flux LCSS	0.783	0.850	
	flux EDR	0.858	0.862	
474	flux LCSS	0.533	0.800	
	flux EDR	0.675	0.813	
497	flux LCSS	0.533	0.800	
	flux EDR	0.700	0.813	
542	flux LCSS	0.475	0.672	
	flux EDR	0.680	0.750	
598	flux LCSS	0.836	0.607	
	flux EDR	0.877	0.710	
620	flux LCSS	0.934	0.541	0.820
	flux EDR	0.951	0.685	0.861

Table 5.1: Flux sequence comparison results

of R and S; we can assign a 'score', or distance, to the similarity of two sequences. Numerous techniques fit into the aegis of edit distance; the common theme is scoring the similarity of R and S by the 'effort' required to transform one to the other. Varieties of edit distance are distinguished by how they define the weight of transformations — [Chen et al., 2005] propose 'edit distance with real penalties' (EDR).

These algorithms typically operate on sequences consisting of elements from a finite set — for example, Latin characters in a string. When dealing with sequences of real numbers (velocity and flux in this case), [Morse and Patel, 2007] suggest identifying two real numbers $x \in R$ and $y \in S$ as 'equal' in the LCSS or EDR sense when $|x - y| < \epsilon = \sigma_{\min}/2$, where $\sigma_{\min}$ is the lesser of the standard deviations of R and S.

For each of the simulation types (microscopic, macroscopic, hybrid), I have compared the resulting flux timeseries with the corresponding NGSIM data — see Table 5.5.3.3. As the data in the $x = 620$ row shows, my hybrid simulation technique only slightly decreases the score for micro simulation.

5.6 Conclusion

I have introduced a technique for hybrid traffic simulation that combines the strengths of continuum-level macroscopic and discrete-level microscopic traffic simulations methodologies.

The hybrid traffic simulation is built on the continuum traffic simulation technique discussed in Chapter 4 and a state-of-the-art agent-based technique; the different regimes govern mutually exclusive dynamic regions of the domain road network. The two mechanisms are coupled together with a flow averaging scheme for microscopic to continuum connections and a novel instantiation scheme I call 'flux capacitors' for continuum to discrete interactions.

I also introduce a new method for instantiating discrete vehicles according to an underlying continuum representation based on inhomogeneous Poisson processes; this is used, along with a car averaging procedure, to convert regions in of one simulation type to the other, allowing for dynamic regions. I propose several criteria for region determination based on user-dictated needs — these broadly cover problems of visualization, performance, and feature preservation.

I demonstrate how this method can be used to simulate traffic on large-scale rural and urban networks with dynamically changing simulation regime regions, I have performed comparisons of microscopic and macroscopic simulations with real-world data, measured the effect of my hybrid coupling technique on the efficacy of these techniques, and shown it to be small.

5.6.1 Limitations

There are limits to the way regions are defined in this method; in particular, while the technique is capable of lane changes, these may only occur between regions in the same regime. I currently do not have a scheme for instantiation from macroscopic to

microscopic regimes as it applies to merging vehicles. While a mechanism for effecting this transition is conceivable, it suffers from the problem of the macroscopic regime being unable to consider information in adjacent lanes; this could result colliding vehicles.

The continuum to discrete instantiation process is not guaranteed to conserve cars. Like the underlying Poisson process it is based on, the scheme is stochastic and is only expected — in the statistical sense — to produce a number of vehicles equal to the corresponding integral of density. For large flows, this is not a significant problem, but for sparsely-populated regions, the effect is more noticeable. One potential workaround is to apply a rejection-type approach to the instantiation process and simply re-run the procedure until the expected number of vehicles is emitted. Like all rejection processes, this is potentially expensive.

The Poisson process used to model arrival times assumes that vehicles arrive independently of one another. This assumption can be improved upon with a more sophisticated model that takes into account the effect of *dependence* between vehicles; the general algorithms and techniques presented in this chapter would still apply, but the more sophisticated distribution could be used to predict arrival times.

5.6.2 Future work

I have looked at how this hybrid coupling mechanism can be applied to a specific continuum model and a specific agent-based scheme; many such schemes exist. It would be illuminating to see how this scheme can be applied to other such models and evaluate its generality.

Much work on traffic simulation has been on well-behaved drivers carefully avoid collisions. In truth, real drivers make errors every day, and automobile accidents are an often-tragic feature of our lives — I hope to investigate how this hybrid technique can be used to simulate behavior where collisions do occur, to better capture the real-world character of automobile traffic and potentially assist study of accident causes on large

scales.

Furthermore, preliminary results have shown that aspects of the hybrid technique are highly parallel; I hope to develop the technique to effectively utilize today's many-core parallel architectures and tomorrow's nascent parallel hardware.

Finally, this approach has many promising avenues for future work. In particular, while I have only investigated how my coupling technique may be applied to two models for simulating and visualizing traffic flows, I would like to examine if the coupling technique may also be applicable to other domains, such as crowd simulations. In addition, it would be interesting to explore using my technique for real-time traffic prediction to examine how the behaviors of individual vehicles at the arterial street level affect the overall traffic flows on freeways; this would be useful to calculate better routing for individual vehicles and more effective remediation for traffic congestion.

In the next and final chapter, I summarize my contributions and present general limitations and avenues for future work.

Chapter 6

Discussion

In this dissertation, I have presented techniques for generating animations of compressible gas flow as well as automobile traffic motion on large road networks. These techniques take advantage of hyperbolic models to describe underlying phenomena, in order to achieve a broad range of results. I demonstrate their scalability on modern parallel computer architectures and show that physics-based animations of these complex phenomenon can be synthesized efficiently.

6.1 Summary of results

I have adapted residual distribution schemes (RDS) to computer graphics as a method for efficiently simulating compressible fluids on modern architectures. I demonstrate that RDS are computationally attractive for animation in several regards – for example, they can effectively model multi-physics phenomena, such as two-way coupling between fluids and solids. They offer a natural balance between efficiency and accuracy, and the method also takes advantage of adaptive mesh refinement to focus computational efforts on areas of visual and physical significance, and are able to deform the computational domain and avoid inaccuracies due to inverted computational cells.

I described a method for efficiently generating animations of supersonic flows in compressible, inviscid fluids. I have demonstrated the ability of this method to capture the

behavior of shocks and to handle complex, bidirectional object-shock interactions stably. I have also demonstrated an effective parallelization scheme based on architectural considerations that achieves near-linear scaling on modern multi-core architectures.

I presented a method for the generation of realistic traffic animations based on continuum dynamics with my own 'carticles' that enable the visualization of discrete vehicles. I have demonstrated this method's ability to generate traffic on large, real-world road networks, and shown that its performance for large numbers of vehicles greatly exceeds that of agent-based methods.

Finally, I presented a technique that combines the strengths of continuum and discrete traffic simulation techniques through a novel coupling scheme that can dynamically transition vehicles' representations between regimes, as well as convert large swaths of road networks between the two simulation methodologies. I suggest several criteria that may be used to guide the selection of regimes for various portions of the road network; these criteria focus on visualization, performance, and preservation of features.

6.2 Limitations

I discuss the limitations of each method in detail its corresponding chapter; here I address limitations common to my general approach. While I have demonstrated the strengths of applying hyperbolic method to animation and simulation problems, it is important to understand the limitations and boundaries of these techniques.

A hyperbolic model can be developed for most physical phenomena, but there are cases when the benefits of such a model are outweighed by other considerations. For example, in nearly all graphics applications, light propagation is assumed to occur instantaneously — cf. the rendering equation [Kajiya, 1986], which is elliptic. Of course, the speed of light is well-known and finite, but for most applications, the temporal and spatial scales of observation are such that infinite-speed propagation is a reasonable

assumption. A similar example (although less extreme) is the incompressible model of fluids popular in graphics; the elliptic divergence-free condition $\nabla \cdot \mathbf{x} = 0$ found in the incompressible formulation of Navier-Stokes is quite reasonable for liquids at scales consistent with human experience; a hyperbolic method applied to such a problem may have to take unreasonably small time steps to successfully describe the evolution of such a system.

As the material in Chapter 5 touches upon, there are times when a technique based on a continuum — be it hyperbolic or not — may not be suitable. While the extension of the theory behind the Knudsen number to traffic is somewhat nebulous, it has a clear place in compressible gas flow; indeed, it is the phenomena that inspired the development of said number! It is not clear if methods for the animation of highly rarefied gases — when the Knudsen number for a compressible fluid exceeds 0.01 — are of great interest, but if so, the material in Chapters 2 and 3 will need a hybrid model with a coupling scheme similar in spirit to that presented in Chapter 5.

6.2.1 Future work

There are a number of exciting areas of future work based on this thesis. While many specific ideas are enumerated in each chapter with the associated material, here I list some areas of future work that apply to the general thesis.

There are many phenomena in the real world we would like to see brought to virtual ones; an interesting direction for future work is the investigation of the applicability of this thesis to other classes of phenomena. Elasticity — particularly the nonlinear plastic regime thereof — is potentially a good candidate for this type of approach. The Saint Venant, or 'shallow water' equations have a natural hyperbolic form that is potentially amenable to this approach, and while incompressible fluids are listed above as potentially unfeasible candidates for the methodology presented in this thesis, it is possible that either by eschewing true incompressiblity or examining large scales, a

hyperbolic approach would suit the needs of certain problems in animation.

This thesis has explored the possibilities of realizing hyperbolic models on modern parallel architectures, but the computing landscape continues to change. It would be very interesting to examine how the methods presented in this thesis map to the various architectures on the near horizon. In particular, the detailed discussion of the parallel shockwave simulator in Section 3.4 examines the effects of the memory hierarchy assume a uniform memory model. As manufacturers introduce commodity architectures with non-uniform memory access (NUMA), many previous assumptions of data management based on uniformity will break down.

Furthermore, while the focus of parallelization in this thesis has been on thread-level parallelization (or a similar abstraction, in the case of GPUs), current and future architectures — such as Intel's Sandy Bridge and Many Integrated Core (MIC, alternately known as Larrabee or Knights Ferry) — are making wide single-instruction, multiple-data (SIMD) architectures available to the user and open exciting possibilities for data-level parallelism.

Appendix A

Residual distribution details

A.1 Conservative → primitive variable transformation matrix

$$\mathbf{M} = \frac{\partial \mathbf{q}}{\partial \mathbf{Q}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ u & \rho & 0 & 0 & 0 \\ v & 0 & \rho & 0 & 0 \\ w & 0 & 0 & \rho & 0 \\ \mathbf{V}^2/2 & \rho u & \rho v & \rho w & 1/\gamma_s \end{bmatrix} \tag{A.1}$$

with $\mathbf{V}^2 = u^2 + v^2 + w^2$, $\gamma_s = \gamma - 1$ ($\mathbf{M}^{-1}$ is easily computed analytically).

The primitive variable Euler equations are

$$\mathbf{Q}_t + \mathbf{F}_\mathbf{Q} \cdot \nabla \mathbf{Q} = f \tag{A.2}$$

with the Jacobian components $\mathbf{F}_\mathbf{Q} = A\mathbf{e}_x + B\mathbf{e}_y + C\mathbf{e}_z$

$$\mathbf{A} = \begin{bmatrix} u & \rho & 0 & 0 & 0 \\ 0 & u & 0 & 0 & \frac{1}{\rho} \\ 0 & 0 & u & 0 & 0 \\ 0 & 0 & 0 & u & 0 \\ 0 & a & 0 & 0 & u \end{bmatrix} \tag{A.3}$$

$$\mathbf{B} = \begin{bmatrix} v & 0 & \rho & 0 & 0 \\ 0 & v & 0 & 0 & 0 \\ 0 & 0 & v & 0 & \frac{1}{\rho} \\ 0 & 0 & 0 & v & 0 \\ 0 & 0 & a & 0 & v \end{bmatrix} \tag{A.4}$$

$$\mathbf{C} = \begin{bmatrix} w & 0 & 0 & \rho & 0 \\ 0 & w & 0 & 0 & 0 \\ 0 & 0 & w & 0 & 0 \\ 0 & 0 & 0 & w & \frac{1}{\rho} \\ 0 & 0 & 0 & a & w \end{bmatrix} \tag{A.5}$$

with

$$a = \left[\frac{p}{\rho} - \rho \left(\frac{\partial e}{\partial \rho} \right)_p \right] \left(\frac{\partial e}{\partial p} \right)_\rho^{-1} = \gamma p \tag{A.6}$$

the last equality being valid for a perfect gas.

A.2 Roe average

The Roe average for the Euler equations is a weighted average of the vertex velocities (u_R, v_R, w_R) and enthalpy h_R, the quantities that appear in the eigenmodes. The weights are defined by the square root of the vertex density.

$$u_R = \frac{\sum_{i=1}^{5} \sqrt{\rho_i} u_i}{\sum_{i=1}^{5} \sqrt{\rho_i}}, \tag{A.7}$$

with similar expressions for the other variables.

A.3 K Matrix Decomposition Cases

1. $(\mathbf{v}\cdot\mathbf{n}_i + c) > \mathbf{v}\cdot\mathbf{n}_i > (\mathbf{v}\cdot\mathbf{n}_i - c) > 0$. All eigenvalues are positive indicating inflow of all wave modes. The splitting is trivial: $\mathbf{K}_i^+ = \mathbf{K}_i, \quad \mathbf{K}_i^- = 0$.

2. $(\mathbf{v}\cdot\mathbf{n}_i + c) > \mathbf{v}\cdot\mathbf{n}_i > 0 \geq (\mathbf{v}\cdot\mathbf{n}_i - c)$. We have $\mathbf{K}_i^+ = \mathbf{K}_{2i}^+$ with $\mathbf{K}_{2i}^+$ given in Appendix Equation (A.8) and $\mathbf{K}_i^- = \mathbf{K}_i - \mathbf{K}_i^+$.

3. $(\mathbf{v}\cdot\mathbf{n}_i + c) > 0 \geq \mathbf{v}\cdot\mathbf{n}_i > (\mathbf{v}\cdot\mathbf{n}_i - c)$. We have $\mathbf{K}_i^+ = \mathbf{K}_{3i}^+$ with $\mathbf{K}_{3i}^+$ given in Appendix Equation (A.13) and $\mathbf{K}_i^- = \mathbf{K}_i - \mathbf{K}_i^+$.

4. $0 \geq (\mathbf{v}\cdot\mathbf{n}_i + c) > \mathbf{v}\cdot\mathbf{n}_i > (\mathbf{v}\cdot\mathbf{n}_i - c)$. All eigenvalues are negative indicating outflow of all wave modes: $\mathbf{K}_i^+ = 0, \quad \mathbf{K}_i^- = \mathbf{K}_i$.

A.4 Inflow/outflow splitting, case 2, 3 inflow matrices

$$\mathbf{K}_{2i}^+ = \begin{bmatrix} k_{2i}^{(1)} & k_{2i}^{(2)} & k_{2i}^{(3)} & k_{2i}^{(4)} & k_{2i}^{(5)} \end{bmatrix} \tag{A.8}$$

$$k_{2i}^{(1)} = \begin{bmatrix} \mathbf{v}\cdot\mathbf{n}_i & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{A.9}$$

$$\begin{bmatrix} k_{2i}^{(2)} & k_{2i}^{(3)} & k_{2i}^{(4)} \end{bmatrix} = \tag{A.10}$$

$$\frac{1}{2}\begin{bmatrix} \rho n_{ix}\lambda_+/c & \rho n_{iy}\lambda_+/c & \rho n_{iz}\lambda_+/c \\ 2V_{ni} - n_{ix}^2\lambda_- & -n_{ix}n_{iy}\lambda_- & -n_{ix}n_{iz}\lambda_- \\ -n_{ix}n_{iy}\lambda_- & 2V_{ni} - n_{iy}^2\lambda_- & -n_{iy}n_{iz}\lambda_- \\ -n_{ix}n_{iz}\lambda_- & -n_{iy}n_{iz}\lambda_- & 2V_{ni} - n_{iz}^2\lambda_- \\ c\rho n_{ix}\lambda_+ & c\rho n_{iy}\lambda_+ & c\rho n_{iz}n_{iy}\lambda_+ \end{bmatrix} \tag{A.11}$$

$$k_{2i}^{(5)} = \begin{bmatrix} -\frac{\lambda_-}{2c^2} & \frac{n_{ix}\lambda_+}{2\rho c} & \frac{n_{iy}\lambda_+}{2\rho c} & \frac{n_{iz}\lambda_+}{2\rho c} & \frac{\lambda_+}{2} \end{bmatrix} \tag{A.12}$$

$$\mathbf{K}_{3i}^{+} = \frac{\lambda_{+}}{2} \begin{bmatrix} 0 & \frac{\rho n_{ix}}{c} & \frac{\rho n_{iy}}{c} & \frac{\rho n_{iz}}{c} & \frac{1}{c^2} \\ 0 & n_{ix}^2 & n_{ix}n_{iy} & n_{ix}n_{iz} & \frac{n_{ix}}{c\rho} \\ 0 & n_{ix}n_{iy} & n_{iy}^2 & n_{iy}n_{iz} & \frac{n_{iy}}{c\rho} \\ 0 & n_{ix}n_{iz} & n_{iy}n_{iz} & n_{iz}^2 & \frac{n_{iz}}{c\rho} \\ 0 & c\rho n_{ix} & c\rho n_{iy} & c\rho n_{iz} & 1 \end{bmatrix} \tag{A.13}$$

Appendix B

The Aw-Rascle-Zhang system

In this appendix, I give an expanded analysis of the Aw-Rascle-Zhang system of equations used for macroscopic traffic simulation in Chapters 4 and 5. In the interest of flow and clear exposition, some material from earlier chapters is repeated here.

B.1 The system of equations

The ARZ model can be written as a conservation law of the form

$$\begin{aligned} &\mathbf{q} + \mathbf{f}(\mathbf{q})_x = 0, \\ &\text{where } \mathbf{q} = \begin{bmatrix} \rho \\ y \end{bmatrix} \text{ and } \mathbf{f}(\mathbf{q}) = \begin{bmatrix} \rho u \\ y u \end{bmatrix} \end{aligned} \tag{B.1}$$

Here ρ is the density of traffic, i.e. "cars per car length", u is the velocity of traffic, and y the "relative flow" of traffic.

y, ρ, and u are related by the equation:

$$y(\rho, u) = \rho(u - u_{\text{eq}}(\rho)) \tag{B.2}$$

where $u_{\text{eq}}(\rho)$ is the "equilibrium velocity" for ρ. There is a single criterion on u_{eq} that must be satisfied (see Section B.2.3.1) on this function; the following works well:

$$u_{\text{eq}}(\rho) = u_{\max}(1 - \rho^{\gamma}) \tag{B.3}$$

where $\gamma > 0$. The first derivative of Equation (B.3) is significant as well:

$$u'_{\text{eq}}(\rho) = -u_{\max}\gamma\rho^{\gamma-1} \tag{B.4}$$

In the above equations, $u_{\max}$ is the speed limit of the road. Using Equations (B.2) and (B.3), one can write u in terms of y and ρ:

$$u(\rho, y) = \frac{y}{\rho} + u_{\text{eq}}(\rho) \tag{B.5}$$

In what follows, I shall interchangeably use $u_{\text{eq}}(\rho)$ and u_{eq} as well as $u(\rho, y)$ and u.

B.2 Waves and speeds

The eigenstructure of the Jacobian of the flux matrix $\mathbf{f}(\mathbf{q})$ from Equation (B.1) reveals the fundamental waves of the system and their speeds.

$$\mathbf{J_f} = \begin{bmatrix} \frac{\partial(\rho u)}{\partial \rho} & \frac{\partial(\rho u)}{\partial y} \\ \frac{\partial(yu)}{\partial \rho} & \frac{\partial(yu)}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial(y+\rho u_{\text{eq}})}{\partial \rho} & \frac{\partial(y+\rho u_{\text{eq}})}{\partial y} \\ \frac{\partial\left(\frac{y^2}{\rho}+yu_{\text{eq}}\right)}{\partial \rho} & \frac{\partial\left(\frac{y^2}{\rho}+yu_{\text{eq}}\right)}{\partial y} \end{bmatrix} = \begin{bmatrix} u_{\text{eq}} + \rho u'_{\text{eq}} & 1 \\ yu'_{\text{eq}} - \frac{y^2}{\rho^2} & \frac{2y}{\rho} + u_{\text{eq}} \end{bmatrix} \tag{B.6}$$

B.2.1 Eigenvalues (Speeds)

The eigenvalues of Equation (B.6) are given by the characteristic polynomial

$$\lambda^2 - (A + D)\lambda + AD - BC = 0 \tag{B.7}$$

Where A, B, C, and D are the elements of $\mathbf{J_f}$:

$$\mathbf{J_f} = \begin{bmatrix} u_{\text{eq}} + \rho u'_{\text{eq}} & 1 \\ yu'_{\text{eq}} - \frac{y^2}{\rho^2} & \frac{2y}{\rho} + u_{\text{eq}} \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{B.8}$$

The solution to Equation (B.7) is given by the quadratic equation:

$$\lambda_0,\, \lambda_1 = \frac{A + D \pm \sqrt{(A+D)^2 - 4\,(AD - BC)}}{2} \tag{B.9}$$

To compute the above, the following is needed:

$$A + D = \rho u'_{\text{eq}} + 2u_{\text{eq}} + \frac{2y}{\rho} = \rho u'_{\text{eq}} + 2u \tag{B.10}$$

$$(A+D)^2 = \rho^2 u'^2_{\text{eq}} + 4u^2_{\text{eq}} + 4\frac{y^2}{\rho^2} + 4\rho u_{\text{eq}} u'_{\text{eq}} + 4y u'_{\text{eq}} + 8\frac{y u_{\text{eq}}}{\rho} \tag{B.11}$$

$$\begin{aligned} 4\,(AD - BC) &= 4\left(u_{\text{eq}} + \rho u'_{\text{eq}}\right)\left(2\frac{y}{\rho} + u_{\text{eq}}\right) - 4\left(y u'_{\text{eq}} - \frac{y^2}{\rho^2}\right) \\ &= 8\frac{y u_{\text{eq}}}{\rho} + 4u^2_{\text{eq}} + 4y u'_{\text{eq}} + 4\rho u_{\text{eq}} u'_{\text{eq}} + 4\frac{y^2}{\rho^2} \end{aligned} \tag{B.12}$$

$$(A+B)^2 - 4\,(AD - BC) = \rho^2 u'^2_{\text{eq}} \tag{B.13}$$

Substituting Equations (B.10) and (B.13) into Equation (B.9):

$$\lambda_0,\, \lambda_1 = \frac{\rho u'_{\text{eq}} + 2u \pm \rho u'_{\text{eq}}}{2} \tag{B.14}$$

$$\lambda_0 = u + \rho u'_{\text{eq}} \tag{B.15}$$

$$\lambda_1 = u \tag{B.16}$$

If one chooses a strictly decreasing u_{eq} (as per Equation (B.3)), then $u'_{\text{eq}} < 0$ (see Equation (B.4)) and the following holds:

$$u + \rho u'_{\text{eq}} = \lambda_0 \leq \lambda_1 = u \tag{B.17}$$

Equation (B.17) is a strict inequality only in the nonphysical state $\rho = 0$.

B.2.2 Eigenvectors (Waves)

The shape of the propagating waves is given by the eigenvectors, which can be determined by the eigenvalues λ_i

$$\mathbf{J_f}\mathbf{r}_i = \lambda_i \mathbf{r}_i \tag{B.18}$$

We can combine Equation (B.18) with λ_0 (from Equation (B.15)) to obtain the corresponding eigenvector $\mathbf{r}_0$

$$\begin{bmatrix} u_{\text{eq}} + \rho u'_{\text{eq}} & 1 \\ y u'_{\text{eq}} - \frac{y^2}{\rho^2} & 2\frac{y}{\rho} + u_{\text{eq}} \end{bmatrix} \begin{bmatrix} r_0^0 \\ r_0^1 \end{bmatrix} = \left(u + \rho u'_{\text{eq}}\right) \begin{bmatrix} r_0^0 \\ r_0^1 \end{bmatrix} \tag{B.19}$$

Writing Equation (B.19) out:

$$\left(u_{\text{eq}} + \rho u'_{\text{eq}}\right) r_0^0 + r_0^1 = \left(u + \rho u'_{\text{eq}}\right) r_0^0 \tag{B.20}$$

$$\left(y u'_{\text{eq}} - \frac{y^2}{\rho^2}\right) r_0^0 + \left(2\frac{y}{\rho} + u_{\text{eq}}\right) r_0^1 = \left(u + \rho u'_{\text{eq}}\right) r_0^1 \tag{B.21}$$

We can solve Equation (B.20) for r_0^1 to obtain:

$$r_0^1 = \left(u - u_{\text{eq}}\right) r_0^0 \tag{B.22}$$

Thus, the eigenvector corresponding to λ_0 is:

$$\mathbf{r}_0 = \begin{bmatrix} 1 \\ u - u_{\text{eq}} \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{y}{\rho} \end{bmatrix} \tag{B.23}$$

We can similarly combine Equation (B.18) with λ_1 (from Equation (B.16)) to obtain the corresponding eigenvector $\mathbf{r}_1$

$$\begin{bmatrix} u_{\text{eq}} + \rho u'_{\text{eq}} & 1 \\ y u'_{\text{eq}} - \frac{y^2}{\rho^2} & 2\frac{y}{\rho} + u_{\text{eq}} \end{bmatrix} \begin{bmatrix} r_1^0 \\ r_1^1 \end{bmatrix} = u \begin{bmatrix} r_1^0 \\ r_1^1 \end{bmatrix} \tag{B.24}$$

Writing Equation (B.24) out:

$$\left(u_{\text{eq}} + \rho u'_{\text{eq}}\right) r_1^0 + r_1^1 = u r_1^0 \tag{B.25}$$

$$\left(y u'_{\text{eq}} - \frac{y^2}{\rho^2}\right) r_1^0 + \left(2\frac{y}{\rho} + u_{\text{eq}}\right) r_1^1 = u r_1^1 \tag{B.26}$$

Now one can use Equation (B.25) and get:

$$r_1^1 = \left(u - u_{\text{eq}} - \rho u'_{\text{eq}}\right) r_1^0 \tag{B.27}$$

Now the eigenvector corresponding to λ_1 can be computed:

$$\mathbf{r}_1 = \begin{bmatrix} 1 \\ u - u_{\text{eq}} - \rho u'_{\text{eq}} \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{y}{\rho} - \rho u'_{\text{eq}} \end{bmatrix} \tag{B.28}$$

B.2.3 Field classification

We can classify each of the two waves as *genuinely nonlinear* or *linearly degenerate* with the following formula:

$$\nabla \lambda_i \cdot \mathbf{r}_i \tag{B.29}$$

If the quantity in Equation (B.29) is zero, one calls the wave i *linearly degenerate* — that is to say, it behaves as the waves in a linear equation and propagates without changing shape. Otherwise, the wave is termed *genuinely nonlinear* — it manifests as shocks or rarefactions.

B.2.3.1 First family of waves

For λ_0 and $\mathbf{r}_0$:

$$\nabla\lambda_0 = \left\langle \frac{\partial\left(u+\rho u'_{\text{eq}}\right)}{\partial\rho}, \frac{\partial\left(u+\rho u'_{\text{eq}}\right)}{\partial y} \right\rangle = \left\langle \frac{\partial\left(\frac{y}{\rho}+u_{\text{eq}}+\rho u'_{\text{eq}}\right)}{\partial\rho}, \frac{\partial\left(\frac{y}{\rho}+u_{\text{eq}}+\rho u'_{\text{eq}}\right)}{\partial y} \right\rangle$$
$$= \left\langle -\frac{y}{\rho^2}+2u'_{\text{eq}}+\rho u''_{\text{eq}}, \frac{1}{\rho} \right\rangle \tag{B.30}$$

We can substitute Equations (B.30) and (B.23) into Equation (B.29):

$$\begin{aligned}\nabla\lambda_0\cdot\mathbf{r}_0 &= \left\langle -\frac{y}{\rho^2}+2u'_{\text{eq}}+\rho u''_{\text{eq}}, \frac{1}{\rho} \right\rangle \cdot \left\langle 1, \frac{y}{\rho} \right\rangle \\ &= -\frac{y}{\rho^2}+2u'_{\text{eq}}+\rho u''_{\text{eq}}+\frac{y}{\rho^2} \\ &= 2u'_{\text{eq}}+\rho u''_{\text{eq}}\end{aligned} \tag{B.31}$$

So long as u_{eq} is such that the quantity in Equation (B.31) $\neq 0$, the first family of waves is genuinely nonlinear. Solutions of genuinely nonlinear families are either shocks or rarefactions.

B.2.3.2 Second family of waves

For λ_1 and $\mathbf{r}_1$:

$$\nabla\lambda_1 = \left\langle \frac{\partial u}{\partial\rho}, \frac{\partial u}{\partial y} \right\rangle = \left\langle \frac{\partial\left(\frac{y}{\rho}+u_{\text{eq}}\right)}{\partial\rho}, \frac{\partial\left(\frac{y}{\rho}+u_{\text{eq}}\right)}{\partial y} \right\rangle = \left\langle -\frac{y}{\rho^2}+u'_{\text{eq}}, \frac{1}{\rho} \right\rangle \tag{B.32}$$

Equations (B.32) and (B.28) can be substituted into Equation (B.29):

$$\begin{aligned}\nabla\lambda_1 \cdot \mathbf{r}_1 &= \left\langle -\frac{y}{\rho^2} + u'_{\text{eq}}, \frac{1}{\rho} \right\rangle \cdot \left\langle 1, \frac{y}{\rho} - \rho u'_{\text{eq}} \right\rangle \\ &= -\frac{y}{\rho^2} + u_{\text{eq}} + \frac{y}{\rho^2} - u_{\text{eq}} \\ &= 0 \end{aligned} \tag{B.33}$$

The second family of waves is linearly degenerate; solutions associated with this family are simple *contact discontinuities* — jumps in the solution that propagate as waves in linear phenomena.

B.2.4 Riemann invariants

Any quantity ω_i is invariant across wave i if it satisfies the following equation:

$$\nabla\omega_i \cdot \mathbf{r}_i = 0 \tag{B.34}$$

B.2.4.1 First family of waves

The invariant for the first wave (corresponding to λ_0 and $\mathbf{r}_0$) can be computed by first substituting Equation (B.23) into Equation (B.34):

$$\left\langle \frac{\partial\,\omega_0}{\partial\,\rho}, \frac{\partial\,\omega_0}{\partial\,y} \right\rangle \cdot \left\langle 1, \frac{y}{\rho} \right\rangle = \frac{\partial\,\omega_0}{\partial\,\rho} + \frac{\partial\,\omega_0}{\partial\,y}\frac{y}{\rho} = 0 \tag{B.35}$$

Equation (B.35) is a partial differential equation that can be solved with separation of variables. Assuming a solution of the form $\omega_0 = G(\rho) H(y)$:

$$\begin{aligned}
\frac{\partial \omega_0}{\partial \rho} + \frac{\partial \omega_0}{\partial y}\frac{y}{\rho} &= 0 \\
\frac{\partial \left(G(\rho) H(y)\right)}{\partial \rho}\rho + \frac{\partial \left(G(\rho) H(y)\right)}{\partial y} y &= 0 \\
G'(\rho) H(y) \rho + G(\rho) H'(y) y &= 0 \\
\frac{G'(\rho)}{G(\rho)}\rho + \frac{H'(y)}{H(y)} y &= 0
\end{aligned} \tag{B.36}$$

The two terms on the left hand-side of Equation (B.36) can now be written as ODEs and solved with standard techniques

$$\begin{aligned}
\frac{G'(\rho)}{G(\rho)}\rho &= k & \qquad \frac{H'(y)}{H(y)} y &= -k \\
\frac{G'(\rho)}{G(\rho)} &= \frac{k}{\rho} & \qquad \frac{H'(y)}{H(y)} &= \frac{-k}{y} \\
G(\rho) &= \rho^k & \qquad H(y) &= y^{-k}
\end{aligned}$$

Then $\omega_0 = G(\rho) H(y) = \rho^k y^{-k}$ for any k; one can choose $k = -1$ and obtain

$$\omega_0 = \frac{y}{\rho} = u - u_{\text{eq}} \tag{B.37}$$

B.2.4.2 Second family of waves

The second family of waves is linearly degenerate, as shown in Section B.2.3.2; λ_1 already satisfies Equation (B.34) (see Equation (B.33)). Thus,

$$\omega_1 = \lambda_1 = u \tag{B.38}$$

B.3 Riemann problem

Given initial constant states q_l (with components which I term ρ_l, y_l, and u_l and ρ_r, y_r, and u_r, respectively) for $x < 0$ and q_r for $x > 0$, what happens for $t > 0$? In the case of the Aw-Rascle-Zhang model, there are several distinct possibilities. To begin with, the speeds of the system (Equations (B.15) and (B.16)) are distinct for $\rho_l > 0$; in this case, there are three distinct regions of the solution: q_l for $x \leq \lambda_0 t$, q_m for $\lambda_0 < \frac{x}{t} < \lambda_1$, and q_r for $x \geq \lambda_1 t$. In the case where $\rho_l = 0$, $\lambda_0 = \lambda_1$ and q_m vanishes. I deal with cases where $\rho_l = 0$ or $\rho_r = 0$ separately below.

B.3.1 Intermediate state

To compute the left- and right-going fluctuations in the update scheme, one needs to compute the value of q_0 — that is to say, the value of q at $x = 0$ for $t > 0$, given q_l and q_r. In general, q_0 can be q_l, q_r, or q_m. For the ARZ model under consideration, $\lambda_1 = u_r > 0$ and therefore q_0 cannot be q_r. It is left to determine if q_0 is q_l or q_m, and if q_m what the value of q_m along $x = 0$ is.

Since $\omega_0 = u - u_{\text{eq}}$ is conserved across the 0-wave connecting q_l and q_m and that $\omega_1 = u$ is conserved across the 1-wave connecting q_m and q_r, the following holds:

$$u_l - u_{\text{eq}}(\rho_l) = u_m - u_{\text{eq}}(\rho_m) \tag{B.39}$$

and

$$u_m = u_r \tag{B.40}$$

Substituting Equation (B.40) into Equation (B.39):

$$
\begin{aligned}
u_l - u_{\text{eq}}(\rho_l) &= u_r - u_{\text{eq}}(\rho_m) \\
u_l - u_r - u_{\text{eq}}(\rho_l) &= -u_{\text{eq}}(\rho_m) \\
u_r - u_l + u_{\text{eq}}(\rho_l) &= u_{\text{eq}}(\rho_m) \\
u_r - u_l + u_{\max}(1 - \rho_l^\gamma) &= u_{\max}(1 - \rho_m^\gamma) \\
\frac{u_r - u_l}{u_{\max}} - \rho_l^\gamma &= -\rho_m^\gamma \\
\rho_l^\gamma + \frac{u_l - u_r}{u_{\max}} &= \rho_m^\gamma \qquad \text{(B.41)} \\
\left(\rho_l^\gamma + \frac{u_l - u_r}{u_{\max}}\right)^{\frac{1}{\gamma}} &= \rho_m \qquad \text{(B.42)}
\end{aligned}
$$

B.3.1.1 Existence of solutions

Clearly, the manipulations in Equation (B.42) are only valid for arbitrary γ when $\rho_m > 0$. The quantity on the left hand side of Equation (B.41) must be > 0; from this, the following condition for ρ_m to exist can be derived:

$$
\begin{aligned}
\rho_l^\gamma + \frac{u_l - u_r}{u_{\max}} &> 0 \\
u_{\max}\rho_l^\gamma + u_l - u_r &> 0 \\
u_{\max}\rho_l^\gamma + u_l &> u_r \qquad \text{(B.43)}
\end{aligned}
$$

B.3.2 Classification of solutions

The speeds (λ_{0l}, λ_{0m}) associated with the first family are (using Equations (B.15) and (B.42)):

$$\lambda_{0l} = u_l - u_{\max}\gamma\rho_l^\gamma \qquad \text{(B.44)}$$

$$\begin{aligned}\lambda_{0m} &= u_m - u_{\max}\gamma\rho_m^\gamma \\ &= u_r - u_{\max}\gamma\left(\rho_l^\gamma + \frac{u_l - u_r}{u_{\max}}\right) \\ &= u_r - u_{\max}\gamma\rho_l^\gamma + \gamma\left(u_r - u_l\right)\end{aligned} \tag{B.45}$$

When $\lambda_{0l} < \lambda_{0m}$, the solution is a rarefaction, and when $\lambda_{0l} > \lambda_{0m}$, the solution is a shock. This reduces to the following:

$$\begin{aligned}\lambda_{0l} &> \lambda_{0m} \\ u_l - u_{\max}\gamma\rho_l^\gamma &> u_r + \gamma\left(u_r - u_l\right) - u_{\max}\gamma\rho_l^\gamma \\ 0 &> u_r - u_l + \gamma\left(u_r - u_l\right) \\ 0 &> \left(1+\gamma\right)\left(u_r - u_l\right) \\ u_l &> u_r\end{aligned}$$

So

$$\lambda_{0l} > \lambda_{0m} \iff u_l > u_r \tag{B.46}$$

For physical inputs to the Riemann problem (that is to say $\rho_l > 0$ and $\rho_r > 0$), four distinct types of solutions can be identified, based on q_l and q_r:

Case 0 $u_r = u_l$

This is a degenerate case, but bears consideration since the assumptions in the remaining cases are based on strict inequalities. In this case, Equation (B.42) predicts $\rho_m = \rho_l$, so $q_m = q_l$. Clearly, there is no wave in the 0-family; the usual contact discontinuity in the 1-family remains.

B.3.2.1 q_o for case 0

Since $q_m = q_l$,

$$q_o = q_l \tag{B.47}$$

regardless of the sign of λ_{0l}.

Case 1 $u_r < u_l$

As per Equation (B.46), the characteristics in the 0-family are colliding and a shock appears in the 0-family. The shock represents in increase in density (because $\rho_m > \rho_l$; see Fig. B.1) and a decrease in velocity (because $u_r < u_l$). The 1-family has a contact discontinuity in ρ as always.

B.3.2.2 Shock speed

In general, a shock's speed is given by:

$$\lambda_s = \frac{\mathbf{f}(\mathbf{q}_+) - \mathbf{f}(\mathbf{q}_-)}{\mathbf{q}_+ - \mathbf{q}_-} \tag{B.48}$$

Here I have slightly abused notion and divided two vectors; what I mean to say by this is that the equation is valid for each component of the vector. For the ARZ model, this gives the following:

$$\lambda_s = \frac{\rho_+ u_+ - \rho_- u_-}{\rho_+ - \rho_-} = \frac{y_+ u_+ - y_- u_-}{y_+ - y_-} \tag{B.49}$$

And in this context, $\rho_+ = \rho_m$, $u_+ = u_m = u_r$, $\rho_- = \rho_l$, and $u_- = u_l$; therefore:

$$\lambda_s = \frac{\rho_m u_m - \rho_l u_l}{\rho_m - \rho_l} \tag{B.50}$$

B.3.2.3 q_o for case 1

Following the above discussion, q_0 for case 1 depends on the sign of Equation (B.50):

$$q_0 = \begin{cases} q_l & \lambda_s \geq 0, \\ q_m & \lambda_s < 0 \end{cases} \tag{B.51}$$

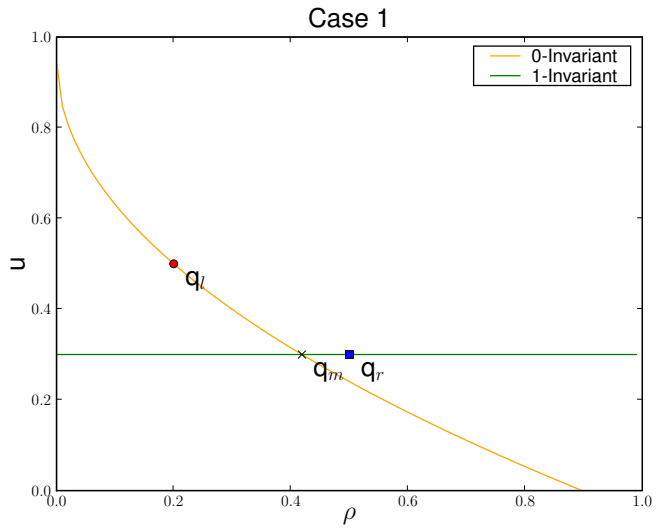


Figure B.1: Case 1 in the ARZ Riemann problem

Case 2 $u_r - u_{\max}\rho_l^{\gamma} < u_l < u_r$

This case satisfies Equation (B.43), so $\rho_m > 0$. Since $u_l < u_r$, Equation (B.46) predicts a rarefaction wave in the 0-family. The 1-family is a contact discontinuity, as always. The intermediate density ρ_m is less than ρ_l (see Fig. B.2).

B.3.2.4 Centered Rarefactions

Rarefactions represent regions of smooth variation and cannot be described with a single speed. λ_{ol} and λ_{om} (see Equations (B.44) and (B.45)) determine if the

rarefaction is to the left or right of q_o (in which case q_0 is q_m or q_l, respectively) or if q_o is inside the rarefaction. If this is the case, it is a *centered* or *transonic* rarefaction; some extra work must be to determine what the structure of the rarefaction is and specifically evaluate it at $x = 0$.

Following [Leveque, 2002], q in a centered rarefaction is given by the following system of differential equations:

$$\tilde{q}'(\xi) = \frac{r_p(\tilde{q}(\xi))}{\nabla\lambda_p(\tilde{q}(\xi)) \cdot r_p(\tilde{q}(\xi))} \tag{B.52}$$

with boundary conditions

$$\begin{aligned} \xi_0 &= \lambda_{0l} \quad \tilde{q}(\xi_0) = q_l \\ \xi_1 &= \lambda_{0m} \quad \tilde{q}(\xi_1) = q_r \end{aligned} \tag{B.53}$$

for $p = 0$, 1 are the various solution families. For $p = 0$, the denominator of Equation (B.52) is Equation (B.31), and r_o is given by Equation (B.23). Substituting these into Equation (B.52):

$$\tilde{q}'(\xi) = \frac{1}{2\tilde{u}'_{\text{eq}} + \tilde{\rho}\tilde{u}''_{\text{eq}}} \begin{bmatrix} 1 \\ \tilde{u} - \tilde{u}_{\text{eq}} \end{bmatrix} \tag{B.54}$$

where $\tilde{u}_{\text{eq}}$, $\tilde{u}$, and $\tilde{\rho}$ are all functions of ξ. The denominator above can be expanded as follows:

$$\begin{aligned} 2\tilde{u}'_{\text{eq}} + \tilde{\rho}\tilde{u}''_{\text{eq}} &= -2u_{\max}\gamma\tilde{\rho}^{\gamma-1} - u_{\max}\tilde{\rho}\gamma(\gamma-1)\tilde{\rho}^{\gamma-2} \\ &= -2u_{\max}\gamma\tilde{\rho}^{\gamma-1} - u_{\max}\gamma(\gamma-1)\tilde{\rho}^{\gamma-1} \\ &= -(2+(\gamma-1))\gamma u_{\max}\tilde{\rho}^{\gamma-1} \\ &= -\gamma(\gamma+1)u_{\max}\tilde{\rho}^{\gamma-1} \end{aligned} \tag{B.55}$$

Equation (B.55) can be substituted into Equation (B.52) to obtain the following ODEs:

$$\tilde{\rho}' = \frac{-1}{\gamma\left(\gamma+1\right)u_{\max}\tilde{\rho}^{\gamma-1}} \tag{B.56}$$

and

$$\tilde{y}' = \frac{\tilde{u}_{\text{eq}} - \tilde{u}}{\gamma\left(\gamma+1\right)u_{\max}\tilde{\rho}^{\gamma-1}} \tag{B.57}$$

Solution for $\tilde{\rho}$ Equation (B.56) is solved using standard techniques:

$$\begin{aligned}
\frac{d\,\tilde{\rho}}{d\,\xi} &= \frac{-1}{\gamma\left(\gamma+1\right)u_{\max}\tilde{\rho}^{\gamma-1}} \\
\tilde{\rho}^{\gamma-1}d\,\tilde{\rho} &= \frac{-d\,\xi}{\gamma\left(\gamma+1\right)u_{\max}} \\
\int \tilde{\rho}^{\gamma-1}d\,\tilde{\rho} &= \int \frac{-d\,\xi}{\gamma\left(\gamma+1\right)u_{\max}} \\
\frac{\tilde{\rho}^{\gamma}}{\gamma} + C_{\rho} &= \frac{-\xi}{\gamma\left(\gamma+1\right)u_{\max}} + C_{\xi} \\
\tilde{\rho}^{\gamma} &= \frac{-\xi + C}{\left(\gamma+1\right)u_{\max}} \\
\tilde{\rho} &= \left(\frac{-\xi + C}{\left(\gamma+1\right)u_{\max}}\right)^{\frac{1}{\gamma}}
\end{aligned} \tag{B.58}$$

A C must be found that satisfies Equation (B.58) subject to Equation (B.53); that is:

$$\tilde{\rho}\left(\lambda_{0l}\right) = \left(\frac{-\lambda_{0l} + C}{\left(\gamma+1\right)u_{\max}}\right)^{\frac{1}{\gamma}} = \rho_l \tag{B.59}$$

and

$$\tilde{\rho}\left(\lambda_{0m}\right) = \left(\frac{-\lambda_{0m} + C}{\left(\gamma+1\right)u_{\max}}\right)^{\frac{1}{\gamma}} = \rho_m \tag{B.60}$$

The above equations can be expanded to find C; for Equation (B.59), we have (using Equation (B.44)):

$$\begin{aligned}
\left(\frac{-\lambda_{0l}+C}{(\gamma+1)\,u_{\max}}\right)^{\frac{1}{\gamma}} &= \rho_l \\
\frac{u_{\max}\gamma\rho_l^{\gamma}-u_l+C}{(\gamma+1)\,u_{\max}} &= \rho_l^{\gamma} \\
u_{\max}\gamma\rho_l^{\gamma}-u_l+C &= (\gamma+1)\,u_{\max}\rho_l^{\gamma} \\
\gamma u_{\max}\rho_l^{\gamma}-u_l+C &= \gamma u_{\max}\rho_l^{\gamma}+u_{\max}\rho_l^{\gamma} \\
C &= u_l+u_{\max}\rho_l^{\gamma}
\end{aligned} \tag{B.61}$$

And for Equation (B.60), we have (using Equation (B.45)):

$$\begin{aligned}
\left(\frac{-\lambda_{0m}+C}{(\gamma+1)\,u_{\max}}\right)^{\frac{1}{\gamma}} &= \rho_m \\
\frac{u_{\max}\gamma\rho_l^{\gamma}-\gamma\,(u_r-u_l)-u_r+C}{(\gamma+1)\,u_{\max}} &= \rho_l^{\gamma}+\frac{u_l-u_r}{u_{\max}} \\
u_{\max}\gamma\rho_l^{\gamma}+\gamma\,(u_l-u_r)-u_r+C &= u_{\max}\,(\gamma+1)\left(\rho_l^{\gamma}+\frac{u_l-u_r}{u_{\max}}\right) \\
u_{\max}\gamma\rho_l^{\gamma}+\gamma\,(u_l-u_r)-u_r+C &= (\gamma+1)\,(u_{\max}\rho_l^{\gamma}+u_l-u_r) \\
\gamma u_{\max}\rho_l^{\gamma}+\gamma\,(u_l-u_r)-u_r+C &= \gamma u_{\max}\rho_l^{\gamma}+\gamma\,(u_l-u_r)+u_{\max}\rho_l^{\gamma}+u_l-u_r \\
C &= u_l+u_{\max}\rho_l^{\gamma}
\end{aligned} \tag{B.62}$$

Clearly Equations (B.61) and (B.62) are in agreement. Thus, the structure of a rarefaction wave given left and right states q_l and q_m is:

$$\tilde{\rho}\,(\xi) = \left(\frac{u_l+u_{\max}\rho_l^{\gamma}-\xi}{(\gamma+1)\,u_{\max}}\right)^{\frac{1}{\gamma}} \tag{B.63}$$

In the context of determining q_0 in the Riemann problem, we wish to evaluate Equation (B.63) at $\xi = 0$:

$$\tilde{\rho}\left(0\right) = \left(\frac{u_l + u_{\max}\rho_l^{\gamma}}{\left(\gamma+1\right)u_{\max}}\right)^{\frac{1}{\gamma}} \tag{B.64}$$

Solution for $\tilde{y}$ Rather than solve Equation (B.57) directly, $\tilde{u}$ (and therefore, with $\tilde{\rho}$ from Equation (B.63), $\tilde{y}$) can be determined using $\tilde{\rho}$ and the Riemann invarient ω_o from Equation (B.37), which must hold across the rarefaction.

$$u_l - u_{\text{eq}}\left(\rho_l\right) = \tilde{u}\left(\xi\right) - u_{\text{eq}}\left(\tilde{\rho}\left(\xi\right)\right) \tag{B.65}$$

$$u_l - u_{\text{eq}}\left(\rho_l\right) = \tilde{u}\left(\xi\right) - u_{\max}\left(1 - \tilde{\rho}\left(\xi\right)^{\gamma}\right) \tag{B.66}$$

$$u_l - u_{\text{eq}}\left(\rho_l\right) = \tilde{u}\left(\xi\right) - u_{\max}\left(1 - \frac{u_l + u_{\max}\rho_l^{\gamma} - \xi}{\left(\gamma+1\right)u_{\max}}\right) \tag{B.67}$$

$$\tilde{u}\left(\xi\right) = u_l - u_{\text{eq}}\left(\rho_l\right) + u_{\max}\left(1 - \frac{u_l + u_{\max}\rho_l^{\gamma} - \xi}{\left(\gamma+1\right)u_{\max}}\right) \tag{B.68}$$

$$\tilde{u}\left(\xi\right) = u_l - u_{\max}\left(1 - \rho_l^{\gamma}\right) + u_{\max} - \frac{u_l + u_{\max}\rho_l^{\gamma} - \xi}{\gamma+1} \tag{B.69}$$

$$\tilde{u}\left(\xi\right) = u_l + u_{\max}\rho_l^{\gamma} - \frac{u_l + u_{\max}\rho_l^{\gamma} - \xi}{\gamma+1} \tag{B.70}$$

$$\tilde{u}\left(\xi\right) = \frac{\left(\gamma+1\right)\left(u_l + u_{\max}\rho_l^{\gamma}\right) - \left(u_l + u_{\max}\rho_l^{\gamma}\right) + \xi}{\gamma+1} \tag{B.71}$$

$$\tilde{u}\left(\xi\right) = \frac{\gamma\left(u_l + u_{\max}\rho_l^{\gamma}\right) + \xi}{\gamma+1} \tag{B.72}$$

As with $\tilde{\rho}$, in the context of determining q_0 in the Riemann problem, we wish to evaluate Equation (B.72) at $\xi = 0$:

$$\tilde{u}\left(0\right) = \frac{\gamma}{\gamma+1}\left(u_l + u_{\max}\rho_l^{\gamma}\right) \tag{B.73}$$

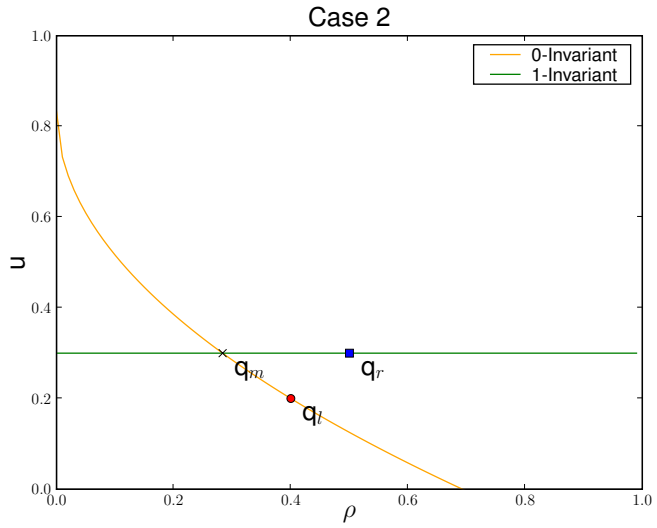


Figure B.2: Case 2 in the ARZ Riemann problem

B.3.2.5 q_0 for case 2

For case 2, q_0 depends on the signs of both Equation (B.44) and Equation (B.45):

$$q_0 = \begin{cases} q_l & \lambda_{0l} \geq 0 \\ q_m & \lambda_{0m} \leq 0 \\ \tilde{q}\left(0, q_l, q_m\right) & \lambda_{0l} < 0 \text{ and } \lambda_{0m} > 0 \end{cases} \tag{B.74}$$

Here $\tilde{q}$ corresponds to the centered rarefaction state described above in Section B.3.2.4.

Case 3 $u_l \leq u_r - u_{\max}\rho_l^{\gamma}$

Now Equation (B.42) cannot be evaluted; the density of the intermediate state ρ_m is 0 (see Fig. B.3).

So $\rho_m = 0$; to observe the Riemann invariants ω_0 and ω_1 from Section B.2.4, u_m

is given by the following (starting with Equation (B.39)):

$$\begin{aligned} u_l - u_{\text{eq}}(\rho_l) &= u_m - u_{\text{eq}}(0) \\ u_l - u_{\max}(1 - \rho_l^\gamma) + u_{\max} &= u_m \\ u_l + u_{\max}\rho_l^\gamma &= u_m \end{aligned} \tag{B.75}$$

and

$$\begin{aligned} \lambda_{0m} &= u_m - \rho u'_{\text{eq}}(\rho_m) \\ &= u_l + u_{\max}\rho_l^\gamma \end{aligned} \tag{B.76}$$

In this case, $\lambda_{0l} = u_l - u_{\max}\gamma\rho_l^\gamma < \lambda_{0m} = u_l + u_{\max}\rho_l^\gamma$; q_l and q_m are connected through a rarefaction wave. Now there is a jump in both ρ and u to get to q_r; this can be imagined as a fictitious velocity wave where $u_m = u_l + u_{\max}\rho_l^\gamma$ jumps to u_r followed by the usual 1-wave — a contact discontinuity in ρ from $\rho_m = 0$ to ρ_r.

The rarefaction wave discussed above will be centered if $\lambda_{0l} < 0$ (since $\lambda_{0m} > 0$). In this case, the same approach used for Case 2 applies (see Section B.3.2.4). However, while the analysis of the boundary conditions for the integrated equation Equation (B.58) is valid for ρ_l, in Equation (B.59), a specific definition of q_m with $u_m = u_r$ and $\rho_m > 0$ given by Equation (B.42) was considered; revisiting Equation (B.60) for the new q_m shows if the previous choice of $C = u_l + u_{\max}\rho_l^\gamma$ is still valid.

$$\begin{aligned} \left(\frac{-\lambda_{0m} + C}{(\gamma+1)\,u_{\max}}\right)^{\frac{1}{\gamma}} &= \rho_m \\ \left(\frac{-(u_l + u_{\max}\rho_l^\gamma) + C}{(\gamma+1)\,u_{\max}}\right)^{\frac{1}{\gamma}} &= 0 \\ C &= u_l + u_{\max}\rho_l^\gamma \end{aligned} \tag{B.77}$$

This is consistent with the C determined by Equation (B.61), so it remains to use Equation (B.64) and Equation (B.73) for q_0. Regardless of q_m, the value of q_o depends on the sign of λ_{0l} (just as with case 2).

B.3.2.6 q_0 for case 3

q_0 for this case is identical to that of case 2, except that the possibility of $q_0 = q_m$ can be eliminated, since $\lambda_{0m} > 0$:

$$q_0 = \begin{cases} q_l & \lambda_{0l} \geq 0 \\ \tilde{q}\left(0, q_l, q_m\right) & \lambda_{0l} < 0 \end{cases} \tag{B.78}$$

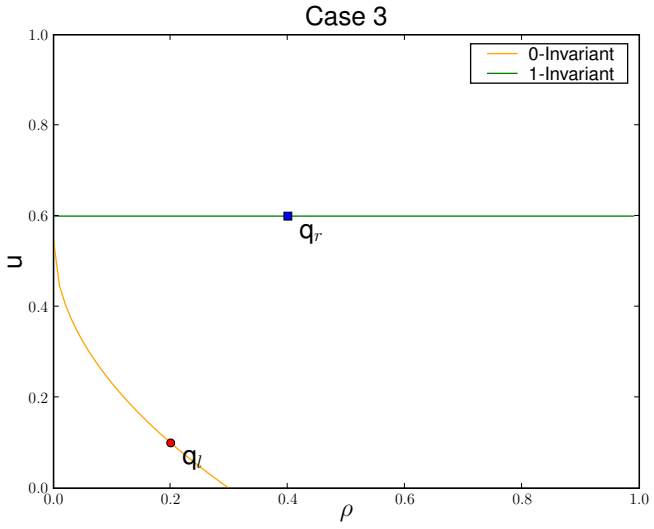


Figure B.3: Case 3 in the ARZ Riemann problem

For robustness and realism, it is desirable to properly handle the absence of traffic (i.e. when ρ_l or ρ_r are 0). In fact, since u is not well-defined for $\rho = 0$, there are only two cases for problems involving "vacuum" states, and they can be treated as sub-cases of case 3:

Case 4 $\rho_l = 0$, $\rho_r > 0$ Clearly, the absence of traffic on the left should have no effect the traffic on the right (drivers in front will not change their behavior if there are no cars behind them); no q_m reasonably exists. There are no waves of the first family; the usual contact discontinuity in the second family remains.

B.3.2.7 q_o for case 4

$$q_0 = q_l = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{B.79}$$

Case 5 $\rho_l > 0$, $\rho_r = 0$ This case is treated the same way as the first family of waves in case 3; we expect a rarefaction wave to appear. There is no jump in ρ, however, so no wave from the second family appears.

B.3.2.8 q_0 for case 5

q_o for this case is identical to case 3 (see Equation (B.78)).

Appendix C

Processing GIS Data

C.1 Geometric Data

GIS road network data contains geometric information for *roads*. The *roads* that GIS data describes can differ from the everyday usage of the word. Here, each *road* is a list of points that may or may not correspond uniquely to a real-world road. Often many of these *road* segments are required to define one real road. There is topological information present in the network in the form of road *intersections*. These are not described explicitly; they are only present in that *road* segments will start or stop at the same points. Again, there is no one-to-one mapping between these *intersections* and those seen on real roads. The reason for this is that, in the GIS file, *roads* do not cross each other. If the real roads cross each other, then the GIS *roads* will meet at an intersection, and two new *roads* will be created. If one of the roads is a highway, however, the *intersection* created would not correspond to a real intersection: there could be an overpass or something similar. Further information present in the geometric data is the direction the *road* is defined in. This is particularly important for roads that are one-way, as often the direction in which the *road* is defined is the only specification of its direction.

C.2 Non-geometric Data

The GIS road network data is accompanied by a database file, in DBF format. The specific data that are included varies from dataset to dataset. However, often, the data

will include fields for road class, road name, speed limit, one-way, number of lanes, and other data.

C.2.1 Desired Output

The processed GIS data needs to be in the format discussed above for the simulation to proceed. This process focuses primarily on creating the topological information needed for the simulation, which the geometric information, such as where the road and intersection boundaries are can be created in a post-processing step

C.2.2 Processing

Intersections First, some region of *roads* of interest is chosen. Then the *intersections* between the *roads* are determined, and the membership data – including which roads are incoming and outgoing based on the direction their geometric data — is defined. These intersections are then classified based on their member roads, as shown in Figure C.1. If an intersection has at least one ramp, it is classified as a *ramp* intersection. Otherwise, if it has at least one highway, it is classified as a *highway* intersection. All other intersections are left as the default class. The intersections of type *ramp* and *highway* are deleted, and the reads crossing at those points are joined. If the names of the *roads* are available, then the *roads* are joined based on the names. If the names are not available, the *roads* are joined based on the similarity of their orientations. Joining *roads* consists of 1) combining their endpoint intersection memberships, 2) combining their geometries, and 3) updating the intersection membership information.

Geometric Modifications The road geometry needs to be modified: GIS data does not account for low level details such as intersection geometry. Therefore, roads that meet at intersections need to be pulled back. This can be done by scaling the final segment of each road connected to an intersection by a constant amount. Doing so

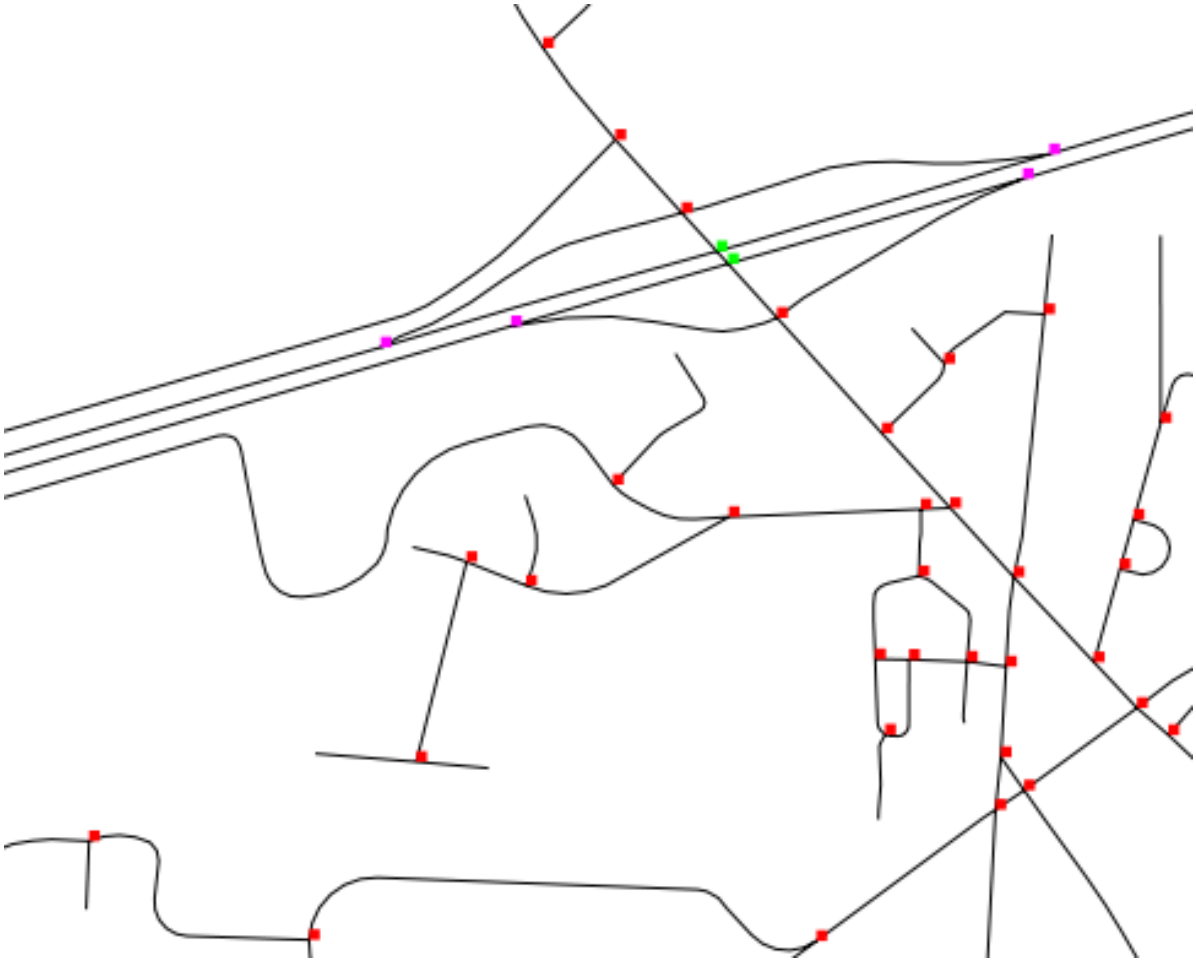


Figure C.1: The intersections are classified as ramps, in purple, highways, in green, or street lights, in red.

preserves the road orientation. However, this approach is complicated by the fact that roads do not approach intersections from the same angle. A constant scaling can be conservative for roads that approach each other at a small angle. A solution to this is to base the scaling factor on the reciprocal of the approach angles.

Ramps change road geometry further. It is desirable that ramps will follow alongside the highway for a period of time to allow realistic merging on and off. To account for this, the highway segment can be added prior to the ramp in the ramp's geometry. Finally, the road geometry must be modified to account for overpasses. Overpasses are present in the *highway* class of intersections. However, there is not a one-to-one relationship between overpasses and these intersections. For example, a divided highway crossed by a street would have two highway type intersections but one overpass. To account for this, all highway intersections within some distance can be joined and considered one. This will not account for the most complicated overpass structures — ones with multiple highways and levels. The geometry of an overpass can be thought of as like a plateau. The incoming roads need to ramp up to the level of the overpass, cross it, and then ramp down to street level. This geometric modification can be defined using two circles centered at the origin at the overpass. The first indicates when the road starts ascending, the second when it crests. Due to the sparseness of GIS data, it is usually necessary to add points at the intersections of the road and the two circles.

Lanes and Adjacencies It is also necessary to define the structure of lanes within the roads and their adjacencies. For the most part, this is trivial: each road has a number of lanes based on its class, these lanes are all adjacent to each other, and they run the length of the road. This is complicated for the ramp case, however. The ramp's lane is always a member of the ramp, but it is adjacent to the highway along a specific interval. This interval must be defined both in terms of its position on the ramp and on the highway.

These intervals are described as follows: for ramps, the adjacency interval appears either at the end or the beginning of the ramp, depending on whether it is an offramp or an onramp. The length of the interval is the length of the segment from the highway that was added to the ramp geometry. For the highway, its interval starts at the point of the ramp class intersection and lasts the length of the segment of the highway added to the ramp.

Intersections The intersections that were not classed as highways or ramps will be defined as street light controlled intersections. These lights have a number of states describing which input lanes can flow to which output lanes. Based on the road intersection membership and lane data that was previously calculated, the states can be defined as every legal pairing of lanes from each member road.

Appendix D

Parallel Traffic Simulation

One advantage of this approach is the ease with which it may made to take advantage of parallel hardware. In theory, the computation of each flux through the solution of the Riemann problem (i.e. Section 4.3.3.4) is independent. Likewise, the given these fluxes, the time integration of the unknowns (see Equation (4.4)) is independent at each cell, and so too is the advection of each carticle as described in Section 4.3.4.

In practice, this is too fine a granularity to be useful, so I choose the handle the advancement of the solution along each lane as a task. I partition all lanes among the available processors and the various kernels mentioned above — the computation of fluxes, integration of the continuum equations, and the advection of the carticles — are all applied in parallel.

The amount of work performed in each lane is directly proportional to the number of cells in that lane, and thus the length (Equation (4.5) ensures that the number of cells in each lane is proportional to its length). To ensure that the workload assigned to each processor is roughly equal, I perform the partitioning of lanes in such a way as to have the number of cells assigned to each processor be as even as possible. Since the number and lengths of lanes can be safely be assumed to be constant throughout the simulation, this can be computed as a static partition.

The problem of jobs of varying length among multiple processors so as to minimize the total processing time is known as the *makespan*. This is an NP-complete problem that is closely related to the more familiar *bin-packing* problem. While it has been proven that there are no fully polynomial approximation algorithms for these techniques, there are $(1+\epsilon)$ polynomial approximation algorithms that are suitable for the static problem

(c.f. [Hochbaum and Shmoys, 1987].)

The parallelization scheme outlined above works well for computing flow along lane. The amount of communication between lanes is a constant amount of data — the computation of fluxes at boundaries only requires the last grid cell of the incoming lane and the first grid cell of the outgoing one. Very little memory is shared and the effect of the computer's memory hierarchy is limited, and this is the scheme that I currently use in my implementation.

In the presence of lane changes, where lanes may be adjacent for a period proportional to their length, lanes may have greater communication needs. This makespan-based partitioning scheme fails to account for adjacencies and could suffer from latency problems due to increased bandwidth and memory hierarchy issues, and the task of producing a partitioning that considers adjacency information in addition to lane length to achieve maximum throughput is a promising problem I hope to explore in future work.

Appendix E

Arc Roads

E.1 Preliminaries

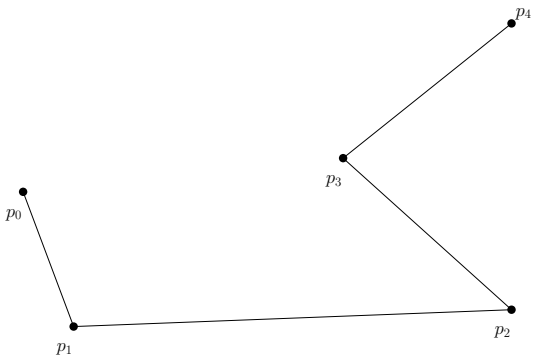
We have an ordered sequence P of n points:

$$P := (p_0, p_1, \ldots, p_{n-2}, p_{n-1}) \tag{E.1}$$

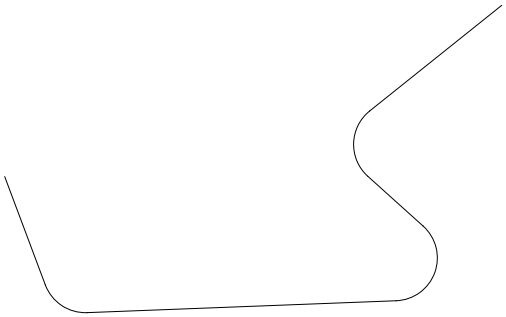
These points define a (not necessary planar) polyline with $n-1$ segments such as that in Figure E.1a. Assume that there are no two points adjacent in the sequence that are equal, and that there are no three adjacent points that are colinear; clearly these adjacent repeated points or the interior points in a colinear sequence can be eliminated without modifying the line's shape.

We wish to 'smooth' this polyline to something like what is shown in Figure E.1b, which I shall refer to as P_S. We construct P_S by replacing the region around each interior point $p_i, i \in \mathbb{Z}\,[1, n-2]$ of P with a circular arc and retaining the exterior points p_0 and p_{n-1}.

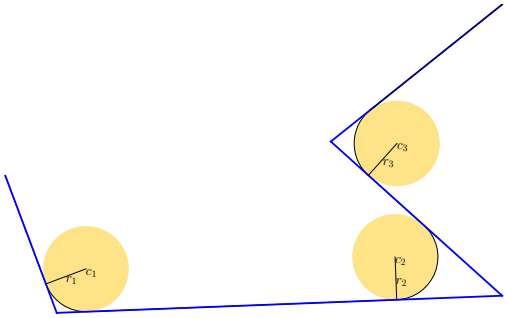
Each of these circular arcs can be characterized by a center c_i, radius r_i, orientation $\mathbf{o}_i$, start radius direction $\mathbf{s}_i$, and angle ϕ_i; see Figure E.2.



(a) A polyline P



(b) P_S: A 'smoothed' version of the polyline P



(c) The polyline P and the circles defining P_S

Figure E.1: Polylines

E.2 Construction of arc roads from polylines

E.2.1 Arc formulation

As mentioned above, each arc is defined by a center c_i, a radius r_i, orientation $\mathbf{o}_i$, start radius vector $\mathbf{s}_i$, and angle ϕ_i. Each arc i corresponds to an interior point p_i, and I require it to be tangent to $\overline{p_{i-1}p_i}$ and $\overline{p_ip_{i+1}}$. See Figures E.1c and E.2; the full circles corresponding to each arc are shown.

To help describe each arc i, I introduce the following quantities derived from the

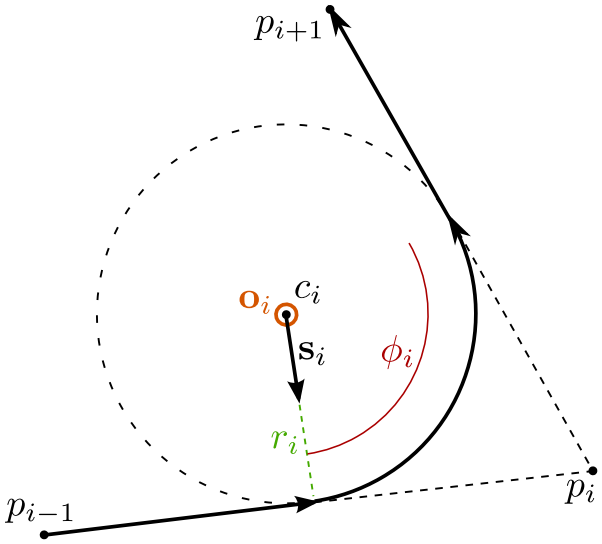


Figure E.2: Quantities defining an arc i corresponding to interior point p_i; the orientation vector $\mathbf{o}_i$ is coming out of the page.

polyline P:

$$\mathbf{v}_i = p_{i+1} - p_i \tag{E.2}$$

their lengths:

$$L_i = |\mathbf{v}_i| \tag{E.3}$$

and the associated unit vectors:

$$\mathbf{n}_i = \frac{\mathbf{v}_i}{L_i} = \frac{\mathbf{v}_i}{|\mathbf{v}_i|} \tag{E.4}$$

We shall frequently refer to $-\mathbf{n}_{i-1} = \frac{p_{i-1}-p_i}{|p_{i-1}-p_i|}$. We also refer to the normal of the plane containing the circle:

$$\mathbf{o}_i = -\mathbf{n}_{i-1} \times \mathbf{n}_i \tag{E.5}$$

At certain times, it is useful to construct a matrix $\mathbf{F}_i$ that is the frame defined by $\mathbf{n}_i$,

$\mathbf{s}_i$, and $\mathbf{o}_i$:

$$\mathbf{F}_i = \begin{bmatrix} \mathbf{n}_i & \mathbf{s}_i & \mathbf{o}_i \end{bmatrix} \tag{E.6}$$

The matrix $\mathbf{H}_i$ representing the homogeneous transform of translation to the arc center c_i along with the frame is defined as follows:

$$\mathbf{H}_i = \begin{bmatrix} \mathbf{n}_i & \mathbf{s}_i & \mathbf{o}_i & c_i \\ & 0 & & 1 \end{bmatrix} \tag{E.7}$$

E.2.1.1 Tangent points

The projections of $c_i - p_i$ onto $-\mathbf{n}_{i-1}$ and onto $\mathbf{n}_i$ have equal length α_i; then the tangent points of the circle on $-\mathbf{v}_{i-1}$ and $\mathbf{v}_i$ are:

$$(c_i - p_i) \text{ proj } -\mathbf{n}_{i-1} + c_i = -\alpha_i \mathbf{n}_{i-1} + c_i \tag{E.8}$$

$$(c_i - p_i) \text{ proj } \mathbf{n}_i + c_i = \alpha_i \mathbf{n}_i + c_i \tag{E.9}$$

E.2.1.2 Radius vectors

We are also interested in the (negative) radius vectors from these points to the center c_i:

$$\mathbf{r}_i^- = -r_i \mathbf{s}_i = c_i - (-\alpha \mathbf{n}_{i-1} + p_i) = c_i + (\alpha \mathbf{n}_{i-1} - p_i) \qquad = r_i \mathbf{n}_{i-1} \times \mathbf{o}_i \tag{E.10}$$

$$\mathbf{r}_i^+ = c_i - (\alpha \mathbf{n}_i + p_i) \qquad = r_i \mathbf{n}_i \times \mathbf{o}_i \tag{E.11}$$

Obviously, since $\mathbf{n}_{i-1}$ and $\mathbf{n}_i$ are perpendicular to $\mathbf{o}_i$ (see Equation (E.5)), $\left|\mathbf{r}_i^-\right| = \left|\mathbf{r}_i^+\right| = r_i$.

E.2.1.3 The center

The center c_i can be determined by combining Equations (E.10), (E.11) and (E.8), (E.9):

$$c_i = p_i - \alpha_i \mathbf{n}_{i-1} + \mathbf{r}_i^- \tag{E.12}$$

$$= p_i + \alpha_i \mathbf{n}_i + \mathbf{r}_i^+ \tag{E.13}$$

We can also write c_i in terms of the unit bisector $\mathbf{b}_i$:

$$c_i = p_i + \sqrt{r_i^2 + \alpha_i^2}\mathbf{b}_i \tag{E.14}$$

Where $\mathbf{b}_i$ is of course given by:

$$\mathbf{b}_i = \frac{\mathbf{n}_i + \mathbf{n}_{i-1}}{|\mathbf{n}_i - \mathbf{n}_{i-1}|} \tag{E.15}$$

See Fig. E.3 for a visual depiction of these quantities.

E.2.1.4 Angles

From Figure E.3, we know:

$$\cos\phi_i = \mathbf{r}_i^- \cdot \mathbf{r}_i^+ \tag{E.16}$$

$$\cos 2\theta_i = -\mathbf{n}_{i-1} \cdot \mathbf{n}_i \tag{E.17}$$

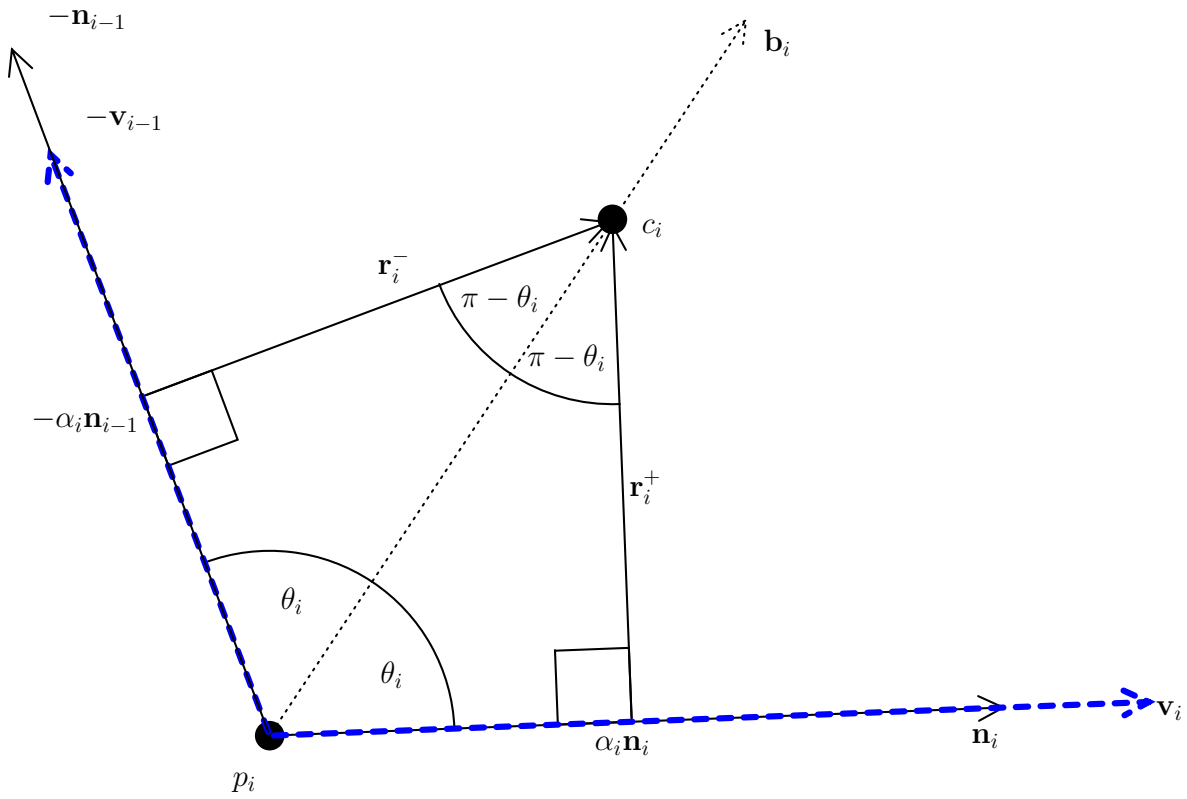


Figure E.3: The interior point p_i with backward vector $-\mathbf{v}_{i-1}$ and forward vector $\mathbf{v}_i$. $\mathbf{b}_i$ is the unit bisector of these vectors

Combining Equations (E.16) and (E.17), we get the following equation for the angle ϕ_i of each arc:

$$\begin{aligned}
\phi_i &= 2\,(\pi - \theta_i) \\
&= 2\pi - \arccos -\mathbf{n}_{i-1} \cdot \mathbf{n}_i \\
&= 2\pi - (\arccos \mathbf{n}_{i-1} \cdot \mathbf{n}_i - \pi) \\
&= \pi - \arccos \mathbf{n}_{i-1} \cdot \mathbf{n}_i
\end{aligned} \tag{E.18}$$

E.2.1.5 Relating r_i and α_i

Say we are given a triple of points p_{i-1}, p_i, p_{i+1} to which we wish to fit an arc. Of the quantities that characterize an arc listed in Section E.1, the orientation $\mathbf{o}_i$, angle ϕ_i depend solely on the normals $\mathbf{n}_{i-1}$ and $\mathbf{n}_i$; see Equations (E.5) and (E.18). The center c_i and start radius vector $\mathbf{s}_i$ depend on these normals and the radius r_i.

To fit an arc to an interior point i, we must choose an appropriate radius to complete the definition. The obvious lower bound condition is $r_i > 0$; as upper bound, each radius depends on the geometry of the triple of points about p_i. To remain tangent to both $\overline{p_{i-1}p_i}$ and $\overline{p_ip_{i+1}}$, we must be certain that the points of tangency $-\alpha_i\mathbf{n}_{i-1}$ and $\alpha_i\mathbf{n}_i$ are on those segments. This translates to the following:

$$\alpha_i \leq \min\{L_{i-1}, L_i\} \tag{E.19}$$

To put this limit in terms of r_i, we need a formula relating r_i, and α_i. From inspection of Fig. E.3, we can see that:

$$r_i = \alpha_i \tan\theta_i \tag{E.20}$$

We know from trigonometry that

$$\tan\theta = \sqrt{\frac{1-\cos 2\theta}{1+\cos 2\theta}} \tag{E.21}$$

We can combine Equation (E.20) with Equation (E.21) to obtain

$$r_i = \alpha_i\sqrt{\frac{1-\cos 2\theta_i}{1+\cos 2\theta_i}} \tag{E.22}$$

Finally, we can substitute Equation (E.17) into Equation (E.22), and obtain

$$r_i = \alpha_i \sqrt{\frac{1 - \mathbf{n}_i \cdot -\mathbf{n}_{i-1}}{1 + \mathbf{n}_i \cdot -\mathbf{n}_{i-1}}} = \alpha_i \sqrt{\frac{1 + \mathbf{n}_i \cdot \mathbf{n}_{i-1}}{1 - \mathbf{n}_i \cdot \mathbf{n}_{i-1}}} \tag{E.23}$$

Then the limits on r_i subject to the following:

$$r_i \in \left(0, \min\left\{L_{i-1}, L_i\right\} \sqrt{\frac{1 + \mathbf{n}_i \cdot \mathbf{n}_{i-1}}{1 - \mathbf{n}_i \cdot \mathbf{n}_{i-1}}}\right]$$

E.2.2 Fitting the r_i

In Section E.2.1, I developed the tools to compute an arc for each interior point of a polyline P given a radius r_i for each.

Given an arbitrary polyline P, it is desirable to automatically select the r_i to complete the definition of a smoothed polyline P_S. A reasonable goal is to pick the r_i such that the quantity

$$\min_{i \in [1, n-2]} r_i \tag{E.24}$$

is *maximal* over all valid configurations of r_i; this helps minimize the 'sharpness' of each corner.

We must consider what values of r_i are valid configurations. The bounds on α_i given in Equation (E.19) are valid for a given triple of points, but when we are concerned with the α_i for all of the interior points of P, we must consider that the L_i between interior points p_{i-1} and p_i are in contention, i.e.

$$\alpha_i + \alpha_{i+1} \leq L_i \tag{E.25}$$

where again we set $\alpha_0 = \alpha_{n-1} = 0$.

E.2.2.1 Fitting algorithm

I have developed a recursive algorithm for selecting the r_i for each arc given a polyline P that satisfies Equation (E.24). Briefly, we iterate over all of the segments $\overline{p_i p_{i+1}}$, $i \in [0, n-2]$ and consider how *large* a radius it is possible to assign to the arcs i, $i+1$ at either end of the segment i; we take the *smallest* such segment (and associated radius) and assign this radius to the associated arcs. This process is repeated until each interior point has been assigned a radius value.

A key component of the algorithm is how one considers how large a radius can be assigned to the arcs at either end of a segment i; we wish to 'balance' the radii of the arcs at either end of the segments such that $r_i = r_{i+1}$. This leads to:

$$r_i = r_{i+1} = \alpha_i f_i = \alpha_{i+1} f_{i+1} \tag{E.26}$$

Where I have used Equation (E.23) and we introduce the convenience

$$f_i = \sqrt{\frac{1 + \mathbf{n}_i \cdot \mathbf{n}_{i-1}}{1 - \mathbf{n}_i \cdot \mathbf{n}_{i-1}}} \tag{E.27}$$

Combining Equations (E.26) and (E.25), we compute:

$$\begin{aligned}
L_i - \alpha_i \geq \alpha_{i+1} &= \frac{\alpha_i f_i}{f_{i+1}} \\
L_i - \alpha_i &\geq \frac{\alpha_i f_i}{f_{i+1}} \\
L_i &\geq \frac{\alpha_i f_i}{f_{i+1}} + \alpha_i \\
L_i &\geq \alpha_i \frac{f_i}{f_{i+1}} + 1 \\
L_i &\geq \alpha_i \frac{f_i + f_{i+1}}{f_{i+1}} \\
\frac{L_i f_{i+1}}{f_i + f_{i+1}} &\geq \alpha_i
\end{aligned} \tag{E.28}$$

This gives (invoking Equation (E.25) again):

$$\alpha_i = \min\left\{L_{i-1} - \alpha_{i-1}, \frac{L_i f_{i+1}}{f_i + f_{i+1}}\right\} \tag{E.29}$$

$$\alpha_{i+1} = \min\left\{L_{i+1} - \alpha_{i+2}, L_i - \alpha_i\right\} \tag{E.30}$$

Since it is natural to define $\alpha_0 = \alpha_n = 0$, Equation (E.29) is not considered when $i = 0$; nor is Equation (E.30) when $i = n - 1$. Algorithm E.1 gives pseudocode for this radius-balancing procedure. The rest of the algorithm is given in detail in Algorithm E.2; the

Algorithm E.1 RADIUS-BALANCE

RADIUS-BALANCE(i)
1 $\alpha_a = \min\left\{L_{i-1}, \frac{L_i f_{i+1}}{f_i + f_{i+1}}\right\}$
2 $\alpha_b = \min\left\{L_{i+1}, L_i - \alpha_a\right\}$
3 **return** α_a, α_b, $\max\{f_i\alpha_a, f_{i+1}\alpha_b\}$

The RADIUS-BALANCE returns the radius that balances the radii on either end segment i.

procedure ALPHA-ASSIGN is invoked with $s = 0$ and $e = n - 1$ (with A the array of α_i, with $A[0] = A[n-1] = 0$); the r_i are easily computed after ALPHA-ASSIGN completes using Equation (E.23). Both RADIUS-BALANCE and ALPHA-ASSIGN implicitly make use of (but do not modify) quantities associated with the polyline P: L_i from Equation (E.3) and f_i from Equation (E.27).

E.2.2.2 Optimality of radii-selection algorithm

The algorithm described above in Section E.2.2.1 aims to maximize Equation (E.24). Here I informally demonstrate that the resulting assignment of r_i makes the value of Equation (E.24) as large as possible given the shape of the input polyline P.

Consider that ALPHA-ASSIGN has been run on a polyline P. Now consider the set

Algorithm E.2 ALPHA-ASSIGN

ALPHA-ASSIGN(A, s, e)

```
    // A — array of n α values, s — start segment index, e — end segment index + 1
 1  r_min, i_min = ∞, e     // Initialize min. radius, index of min. index
 2  if s + 1 ≥ e            // Return if interval is length zero
 3       return
 4  α_b = min{L_s − A[s], L_{s+1}}
 5  r_current = max{f_s A[s], f_{s+1} α_b}          // Radius at initial segment
 6  if r_current < r_min
 7       r_min, i_min = r_current, s
 8       α_low, α_high = A[s], α_b
 9  for i = s + 1 to e − 2                          // Radii for internal segments
10       α_a, α_b, r_current = RADIUS-BALANCE(i)
11       if r_current < r_min
12            r_min, i_min = r_current, i
13            α_low, α_high = α_a, α_b
14  α_a = min{L_{e−2}, L_{e−1} − A[e]}
15  r_current = max{f_{e−1} α_a, f_e A[e]}          // Radius at final segment
16  if r_current < r_min
17       r_min, i_min = r_current, e − 1
18       α_low, α_high = α_a, A[e]
19  A[i_min] = α_low                    // Assign alphas at ends of selected segment
20  A[i_min + 1] = α_high
21  ALPHA-ASSIGN(A, s, i_min)           // Recur on lower segments
22  ALPHA-ASSIGN(A, i_min + 1, e)       // Recur on higher segments
```

The ALPHA-ASSIGN procedure assigning radii based on a polyline P

of radii $R_{\min}$ that have the smallest value of radius $r_{\min}$, namely:

$$r_i = r_{\min},\ \forall i \in R_{\min} \tag{E.31}$$

$$r_i > r_{\min},\ \forall i \in [1, n-2]/R_{\min} \tag{E.32}$$

To increase the value of Equation (E.24), one must increase the value of $r_{\min}$ — i.e increase r_i, $\forall i \in R_{\min}$.

Now recall how Alpha-Assign works; each call examines the unassigned A in its range and finds the largest radius that could be assigned to each. Then the arc that has the smallest such 'largest' radius is assigned to. Thus the radii assigned (indirectly, through the $A[i]$) in a call must be equal to or greater than that assigned in its caller, and so on up to the top-level call. Then the value of the radius computed in the top-level call to Alpha-Assign is $r_{\min}$; to show that the computed $r_{\min}$ cannot be improved upon, it is sufficient to show that the two arcs assigned to (one if $i_{\min} = s$ or $e-1$) in the top-level call to Alpha-Assign cannot have their radii increased.

Boundary case First consider the case where $i_{\min} = s = 0$ (since this the top-level invocation, $s = 0$). Then in Line 4 of Alpha-Assign, know we must have chosen

$$\alpha_b = L_0 - A[0] = L_0 \tag{E.33}$$

(recall that $A[0] = 0$). Otherwise, that would mean

$$L_0 - A[0] = L_0 > L_1 \tag{E.34}$$

This would lead to r_{current} having been assigned $f_1 L_1$ at Line 5. But then when $i = 1$ and invoke RADIUS-BALANCE(1) on Line 10,

$$\alpha_a = \min\left\{L_0, \frac{L_1 f_2}{f_1 + f_2}\right\} = \frac{L_1 f_2}{f_1 + f_2} \tag{E.35}$$

in Line 1 of RADIUS-BALANCE because $f_2/(f_1 + f_2) < 1$ and $L_0 > L_1$ (from Equation (E.34)). Then we would have replaced $r_{\text{current}} = f_1 L_1$ from Line 5 with $r_{\text{current}} = f_1 L_1 f_2/(f_1 + f_2)$ at Line 11, and we would not have chosen $i_{\min} = 0$. Thus, by contradiction, if $i_{\min} = 0$ in the first call to ALPHA-ASSIGN, we know that $A[1] = L_0$ and $r_1 = f_1 L_0 = r_{\min}$.

So $A[1] = \alpha_1 = L_0$ and $r_{\min} = r_1 = f_1 L_0$; then the condition in Equation (E.25) dictates that we cannot increase α_1. Neither, in that case, can we increase $r_{\min}$. A similar argument can be made in the case where $i_{\min} = e - 1 = n - 2$.

General case I shall move on to demonstrating that when $0 < i_{\min} < n - 2$, neither of the assigned $A[i_{\min}]$ and $A[i_{\min} + 1]$ (nor their associated radii) may be increased.

I shall do this by demonstrating that $r_{i_{\min}} = r_{i_{\min}+1} = r_{\min}$ and that $A[i_{\min}] + A[i_{\min} + 1] = L_{i_{\max}}$; then there is no way to increase either of $r_{i_{\min}}$ or $r_{i_{\min}+1}$ without decreasing the other, and thus $r_{\min}$ must hold.

Consider $i_{\max} \in [1, n-3]$ from the top-level invocation of ALPHA-ASSIGN; the associated α_a and α_b must have been computed via RADIUS-BALANCE(i) on Line 10. I contend that Line 1 in RADIUS-BALANCE(i) must have computed $\alpha_a = \min\left\{L_{i-1}, \frac{L_i f_{i+1}}{f_i + f_{i+1}}\right\} = \frac{L_i f_{i+1}}{f_i + f_{i+1}}$.

If it had not, then it would have computed $\alpha_a = L_{i-1}$ and we would know that

$$L_{i-1} < \frac{L_i f_{i+1}}{f_i + f_{i+1}} \tag{E.36}$$

Furthermore, we know that RADIUS-BALANCE would have returned some Q such that

$$r_{\text{current}} = Q \geq f_i L_{i-1} \tag{E.37}$$

as computed on its Line 3. Now consider what the computation for $i-1$ must have looked like; if $i > 1$, then we have

$$\alpha_a = \min\left\{L_{i-2}, \frac{L_{i-1} f_i}{f_{i-1} + f_i}\right\} \tag{E.38}$$

$$\alpha_b = \min\left\{L_i, L_{i-1} - \alpha_a\right\} \tag{E.39}$$

These equations are from Lines 1 and 2 of a call to RADIUS-BALANCE $(i-1)$. By combining Equations (E.36) and (E.39), it can be deduced that $\alpha_b = L_{i-1} - \alpha_a$, regardless of the value of α_a. Then, in Line 3, the radius for $i-1$ is

$$r_{\text{current}} = \max\left\{f_{i-1}\alpha_a, f_i\left(L_{i-1} - \alpha_a\right)\right\} \tag{E.40}$$

In fact, we know

$$f_{i-1}\alpha_a \leq f_i\left(L_{i-1} - \alpha_a\right) \tag{E.41}$$

because α_a is given by Equation (E.38) — if $\alpha_a = \frac{L_{i-1} f_i}{f_{i-1}+f_i}$, then we know by Equations (E.26) (E.29), and (E.30) that $f_{i_1}\alpha_a = f_i\left(L_{i-1} - \alpha_a\right)$. On the other hand, if $\alpha_a = L_{i-2}$, then by Equation (E.38), $L_{i-2} \leq \frac{L_{i-1} f_i}{f_{i-1}+f_i}$; thus Equation (E.41) must be true.

So we know that the radius computed in Equation (E.40) for $i-1$ is $f_i\left(L_{i-1} - \alpha_a\right)$. Regardless of which value we select for α_a in Equation (E.38), this radius is obviously less than the Q from Equation (E.37) that we computed for the segment i that is supposed to be $i_{\min}$. Thus choosing anything but $\alpha_a = \frac{L_i f_{i+1}}{f_i + f_{i+1}}$ in Line 1 in RADIUS-BALANCE(i) results in a contradiction.

In fact, this last discussion particularly demonstrated that contradiction under the

assumption that $i > 1$. Having $i = 1$ ($i = 0$ was already covered in the discussion of boundary cases above) results in slightly different Equations (E.38) and (E.39), but the contradiction develops on similar grounds.

On the way to show that $r_{i_{\min}} = r_{i_{\min}+1} = r_{\min}$ and $A[i_{\min}] + A[i_{\min}+1] = L_{i_{\max}}$ (as must be done to demonstrate that we cannot increase $r_{\min}$), we have just shown that Line 1 in RADIUS-BALANCE(i) computed $\alpha_a = \frac{L_i f_{i+1}}{f_i + f_{i+1}}$. When we compute α_b in Line 2 of RADIUS-BALANCE(i), we have

$$\alpha_b = \min\left\{L_{i+1}, L_i - \frac{L_i f_{i+1}}{f_i + f_{i+1}}\right\} \tag{E.42}$$

It is straightforward to show that $L_{i+1} \geq L_i - \frac{L_i f_{i+1}}{f_i + f_{i+1}}$ using a process similar to the one just used to show $\alpha_a = \frac{L_i f_{i+1}}{f_i + f_{i+1}}$; with that condition, clearly $\alpha_{i_{\min}} + \alpha_{i_{\min}+1} = L_i$ and $r_{i_{\min}} = f_{i_{\min}}\alpha_{i_{\min}} = r_{i_{\min}+1} = f_{i_{\min}+1}\alpha_{i_{\min}+1} = r_{\min}$. Since neither $\alpha_{i_{\min}}$ nor $\alpha_{i_{\min}+1}$ can be increased without decreasing the other, and because the radii are equal in size, the minimum radius computed in the top-level call to ALPHA-ASSIGN cannot be made larger; it is the maximum value possible for Equation (E.24).

E.2.3 Length of a smoothed polyline

It is useful to consider the length of a smoothed polyline P_S. This will be the sum of the circumference of each arc a_i plus the length of the straight line segments connected consecutive arcs and the segments connecting p_0 to arc 1 and arc $n-2$ to p_{n-1}.

The length w_i of an arc i given by the standard formula:

$$w_i = 2\pi r_i \frac{\phi_i}{2\pi} = r_i \phi_i \tag{E.43}$$

The length s_i of a segment connecting two consecutive arcs i and $i+1$ is simply the

length of the original line from p_i to p_{i+1} (L_i) minus the α_i and α_{i+1} of arcs i and $i+1$:

$$s_i = |p_{i+1} - p_i| - (\alpha_{i+1} + \alpha_i) = L_i - (\alpha_{i+1} + \alpha_i) \tag{E.44}$$

Defining $\alpha_0 = \alpha_{n-1} = 0$ as usual, we can use Equation (E.44) for the beginning/end segments as well.

For nondegenerate P_S (i.e. with $n > 2$), we have:

$$\begin{aligned}
L(P_S) &= \sum_{i=1}^{n-2} r_i \phi_i + \sum_{i=0}^{n-2} s_i \\
&= \sum_{i=1}^{n-2} r_i \phi_i + \sum_{i=0}^{n-2} L_i - (\alpha_{i+1} - \alpha_i) \\
&= \sum_{i=1}^{n-2} r_i \phi_i + L(P) - \sum_{i=0}^{n-2} \alpha_{i+1} - \sum_{i=0}^{n-2} \alpha_i \\
&= L(P) + \sum_{i=1}^{n-2} r_i \phi_i - \sum_{i=1}^{n-1} \alpha_i - \sum_{i=0}^{n-2} \alpha_i \\
&= L(P) + \sum_{i=1}^{n-2} r_i \phi_i - 2\sum_{i=1}^{n-2} \alpha_i - \alpha_0 - \alpha_{n-1} \\
&= L(P) + \sum_{i=1}^{n-2} r_i \phi_i - 2\alpha_i
\end{aligned} \tag{E.45}$$

E.2.4 Offset polylines

So far, given a planar polyline P and an r_i for each interior point p_i, we can compute the smoothed polyline P_S by computing the associated arcs for each interior point of P using the equations in Section E.2.1; we can even compute the r_i automatically in such a way as to minimize curvature using the algorithm presented in Section E.2.2.1.

Suppose that we wish to compute a new smoothed polyline P'_S that has the property that at every point, the nearest point on P_S is exactly distance d away. That is, P'_S is 'offset' from P_S to one side by a signed distance d; see Fig. E.4. I have used the convention that $d > 0$ refers to a 'right' offset (the lower blue line in Fig. E.4) and $d < 0$

to a 'left' offset (the upper blue line in the same figure). The new arcs i corresponding

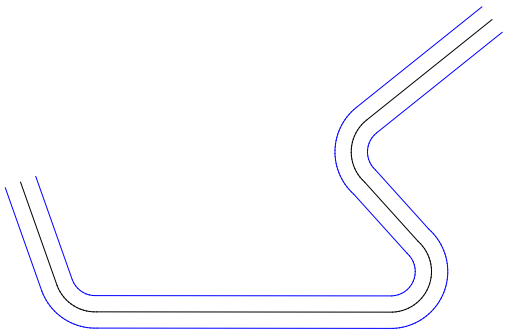


Figure E.4: A 'fattened' smoothed polyline; the original smoothed polyline P_S as computed above is drawn in black. The blue lines represent the same shape offset to either side by an equal distance.

to P'_S (with signed offset d) can be derived from P_S by replacing each r_i with $r_i + d$.

New endpoints p'_0 and p'_{n-1} must be established for this line; a reasonable definition is use the plane of the first and last arcs to choose a perpendicular suitable for placing these offset endpoints.

$$p'_0 = p_0 + d(\mathbf{n}_0 \times \mathbf{o}_1) \tag{E.46}$$

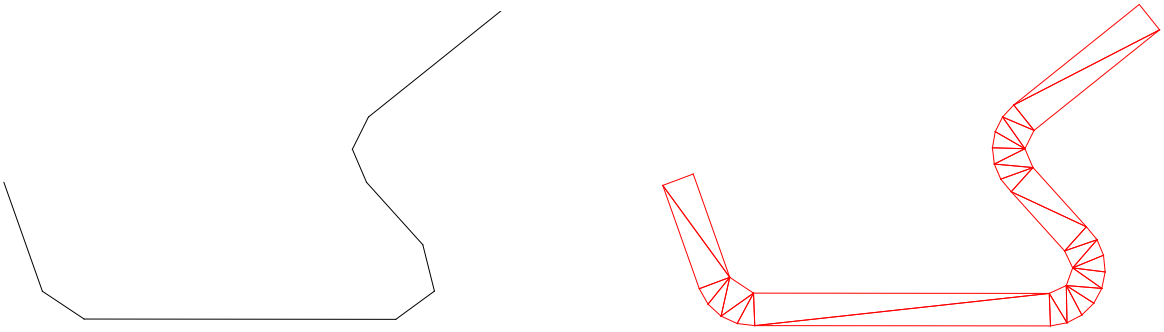
$$p'_{n-1} = p_{n-1} + d(\mathbf{n}_{n-2} \times \mathbf{o}_{n-1} \tag{E.47}$$

E.2.5 Discrete approximations of smooth polylines

To visually depict a smoothed polyline P_S, we may wish to compute a discrete representation.

E.2.5.1 Polylines

One obvious way to do this is by approximating the shape by a series of lines — that is to say, a new polyline P^*. See Figure E.5a. Each arc i must simply be approximated with an sequence Γ_i of $q_i \in \mathbb{Z}_{>1}$ points, and these points connected. Then the sequence



(a) A polyline approximation of a smoothed polyline P_S

(b) A triangle mesh approximation of a 'fattened' smoothed polyline P_S

Figure E.5: Discrete approximations of smoothed polylines

of $m = 2 + \sum_{i=1}^{n-2} q_i$ points in P^* is simply:

$$P_S^* = \left(p_0, \Gamma_1^0, \ldots, \Gamma_1^{q_1-1}, \ldots, \Gamma_{n-2}^0, \ldots, \Gamma_{n-2}^{q_{n-2}-1}, p_{n-1}\right) \tag{E.48}$$

Each Γ_i is generated by rotating and scaling the frame $\mathbf{F}_i$ (from Equation (E.6)) of each arc incrementally and translating by the center c_i:

$$\Gamma_i^j = c_i + r_i \mathbf{F}_i \left[\cos t^j, \sin t^j, 0\right]^{\mathrm{T}},\ j \in \mathbb{Z}[0, q_i - 1] \tag{E.49}$$

Here the t^j are the elements of a sequence $[0, \phi_i/(q_i - 1), 2\phi_i/(q_i - 1), \ldots, \phi_i]$ of length q_i.

E.2.5.2 Triangle meshes

A surface representation of a smoothed polygon can be easily computed from a pair of offset polygons (computed as in Sec. E.2.4). f Given a smoothed polygon P_S and two smooth polygons offset from P_S, order the polygons by offset so that we have a 'left' smoothed polygon P_S^l with a lesser offset than the 'right' smoothed polygon P_S^r.

Now we can use any constrained triangulation technique to compute a planar triangle

mesh with P_S^l and P_S^r as the boundaries; see Figure E.5b.

Bibliography

[Abgrall, 2006] Abgrall, R. (2006). Residual distribution schemes: current status and future trends. *Computers and Fluids*, 35(7):641–669. 20, 24, 38

[Adams et al., 2007] Adams, B., Pauly, M., Keiser, R., and Guibas, L. J. (2007). Adaptively sampled particle fluids. In *ACM SIGGRAPH '07*, page 48, New York, NY, USA. ACM. 19

[Alexiadis et al., 2007] Alexiadis, V., Colyar, J., and Halkias, J. (2007). A model endeavor. Technical Report Publication Number: FHWA-HRT-2007-002, U.S. Department of Transportation Federal Highway Administration. http://www.fhwa.dot.gov/publications/publicroads/07jan/01.cfm. 154

[Algers et al., 1997] Algers, S., Bernauer, E., Boero, M., Breheret, L., Taranto, C. D., Dougherty, M., Fox, K., and Gabard, J. F. (1997). SMARTEST project: Review of micro-simulation models. *EU project No: RO-97-SC*, 1059. 87

[Arikan, 2010] Arikan, O. (2010). Pixie. 35

[Aw and Rascle, 2000] Aw, A. and Rascle, M. (2000). Resurrection of "second order" models of traffic flow. *SIAM journal on applied mathematics*, 60:916–938. 87, 88, 91, 94, 95, 100

[Baraff and Witkin, 1998] Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA. ACM Press. 31

[Batty et al., 2007] Batty, C., Bertails, F., and Bridson, R. (2007). A fast variational framework for accurate solid-fluid coupling. In *ACM SIGGRAPH '07*. 19, 57, 58

[Blender Foundation, 2007] Blender Foundation (2007). Blender 2.45. http://www.blender.org/. 35, 76

[Bridson, 2008] Bridson, R. (2008). *Fluid Simulation for Computer Graphics*. AK Peters Ltd. 18

[Burggraf, 1966] Burggraf, O. (1966). Analytical and numerical studies of the structure of steady separated flows. *Journal of Fluid Mechanics*, 24(01):113–151. 157

[Carlson et al., 2004] Carlson, M., Mucha, P. J., and Turk, G. (2004). Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph.*, 23(3):377–384. 19, 57, 58

[Chaos Group, 2010] Chaos Group (2010). V-Ray. http://www.chaosgroup.com/en/2/vray.html. 76

[Chen et al., 2008] Chen, G., Esch, G., Wonka, P., Mueller, P., and Zhang, E. (2008). Interactive procedural street modeling. In *SIGGRAPH 2008*, New York, NY, USA. ACM. 115

[Chen et al., 2005] Chen, L., Özsu, M., and Oria, V. (2005). Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM. 161, 169

[Chentanez et al., 2007] Chentanez, N., Feldman, B. E., Labelle, F., O'Brien, J. F., and Shewchuk, J. R. (2007). Liquid simulation on lattice-based tetrahedral meshes. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 219–228, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association. 19

[Chentanez et al., 2006] Chentanez, N., Goktekin, T. G., Feldman, B. E., and O'Brien, J. F. (2006). Simultaneous coupling of fluids and deformable bodies. In *SCA '06: Proeedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. 19, 31, 58

[Courant et al., 1928] Courant, R., Friedrichs, K., and Lewy, H. (1928). Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100(1):32–74. 56, 93

[Daganzo, 1995] Daganzo, C. (1995). Requiem for second-order fluid approximations of traffic flow. *Transportation Research Part B*, 29(4):277–286. 87

[Devroye, 1986] Devroye, L. (1986). *Non-Uniform Random Variate Generatiom*. Springer-Verlag. 131, 132

[Elcott et al., 2007] Elcott, S., Tong, Y., Kanso, E., Schröder, P., and Desbrun, M. (2007). Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.*, 26(1):4. 19

[Fedkiw et al., 2003] Fedkiw, R., Sapiro, G., and Shu, C. (2003). Shock capturing, level sets, and PDE based methods in computer vision and image processing: a review of Oshers contributions. *Journal of Computational Physics*, 185(2):309–341. 44

[Feldman et al., 2003] Feldman, B. E., O'Brien, J. F., and Arikan, O. (2003). Animating suspended particle explosions. In *ACM SIGGRAPH '03*, pages 708–715, New York, NY, USA. ACM. 44

[Feldman et al., 2005] Feldman, B. E., O'Brien, J. F., and Klingner, B. M. (2005). Animating gases with hybrid meshes. In *ACM SIGGRAPH '05*, pages 904–909, New York, NY, USA. ACM Press. 19

[Foster and Fedkiw, 2001] Foster, N. and Fedkiw, R. (2001). Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer*

graphics and interactive techniques, pages 23–30, New York, NY, USA. ACM Press. 18

[Foster and Metaxas, 1996] Foster, N. and Metaxas, D. (1996). Realistic animation of liquids. *Graph. Models Image Process.*, 58(5):471–483. 18

[Frigo and Johnson, 2005] Frigo, M. and Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231. Special Issue on "Program Generation, Optimization, and Platform Adaptation". 66

[Galin et al., 2010] Galin, E., Peytavie, A., Maréchal, N., and Guérin, E. (2010). Procedural generation of roads. In *Eurographics 2010*, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association, Eurographics Association. 115

[Game Physics Simulation, 2010] Game Physics Simulation (2010). Bullet Physics Library. http://www.bulletphysics.com/. 58

[Genevaux et al., 2003] Genevaux, O., Habibi, A., and Dischler, J.-M. (2003). Simulating fluid-solid interaction. In *Proc. Graphics Interface '03*. 18

[Gerlough, 1955] Gerlough, D. L. (1955). *Simulation of freeway traffic on a general-purpose discrete variable computer*. PhD thesis, UCLA. 87

[Gibson and Mirtich, 1997] Gibson, S. F. F. and Mirtich, B. (1997). A survey of deformable modeling in computer graphics. Technical Report TR1997-019, Mitsubishi Electronic Research Labratories. 20

[Go et al., 2005] Go, J., Vu, T., and Kuffner, J. (2005). Autonomous behaviors for interactive vehicle animations. In *International Journal of Graphical Models*. 86

[Godunov, 1959] Godunov, S. K. (1959). A difference scheme for numerical solution of discontinuous solution of hydrodynamic equations. *Math. Sbornik*, (47):271–306. 43

[Guenelman et al., 2005] Guenelman, E., Selle, A., Losasso, F., and Fedkiw, R. (2005). Coupling water and smoke to thin deformable and rigid shells. In *ACM SIGGRAPH '05*, pages 973–981, New York, NY, USA. ACM Press. 19

[Guy et al., 2010] Guy, S., Chhugani, J., Curtis, S., Dubey, P., Lin, M. C., and Manocha, D. (2010). PLEdestrians: A Least-Effort Approach to Crowd Simulation. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*. 157

[Helbing, 2001] Helbing, D. (2001). Traffic and related self-driven many-particle systems. *Reviews of Modern Physics*, 73(4):1067–1141. 87

[Hirschfelder et al., 1964] Hirschfelder, J. O., Curtiss, C. F., and Bird, R. B. (1964). *The Molecular Theory of Gases and Liquids*. Wiley-Interscience, revised edition edition. 148

[Hochbaum and Shmoys, 1987] Hochbaum, D. S. and Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34(1):144–162. 208

[Hughes, 2000] Hughes, T. J. R. (2000). *The Finite Element Method—Linear Static and Dynamic Finite Element Analysis*. Dover Publishers, New York, NY. 20, 30

[Kajiya, 1986] Kajiya, J. T. (1986). The rendering equation. In *SIGGRAPH 1986*, pages 143–150. 174

[Kessenich et al., 2009] Kessenich, J., Baldwin, D., and Rost, R. (2009). *The OpenGL© Shading Language*. The Kronos Group, 1.1 edition. http://www.opengl.org/registry/doc/GLSLangSpec.1.50.09.pdf. 62

[Kesting et al., 2007] Kesting, A., Treiber, M., and Helbing, D. (2007). General lane-changing model MOBIL for car-following models. *Transportation Research Record: Journal of the Transportation Research Board*, 1999(-1):86–94. 127

[Kim et al., 2009] Kim, K., Oh, S., Lee, J., and Essa, I. (2009). Augmenting aerial earth maps with dynamic information. In *IEEE International Symposium on Mixed and Augmented Reality*. 151

[Klingner et al., 2006] Klingner, B. M., Feldman, B. E., Chentanez, N., and O'Brien, J. F. (2006). Fluid animation with dynamic meshes. In *ACM SIGGRAPH '06*, pages 820–825, New York, NY, USA. ACM Press. 19, 34

[Knuth, 1997] Knuth, D. (1997). *The Art of Computer Programming*, volume 2, chapter 3, pages 142–148. Addison-Wesley, 3rd edition. 137

[Lax and Wendroff, 1960] Lax, P. and Wendroff, B. (1960). Systems of conservation laws. *Comm. Pure Appl. Math*, 13(2):217–237. 47

[Lebacque et al., 2007] Lebacque, J., Mammar, S., and Haj-Salem, H. (2007). The Aw–Rascle and Zhangs model: Vacuum problems, existence and regularity of the solutions of the Riemann problem. *Transportation Research Part B*, 41(7):710–721. 87, 88

[Lebacque, 1996] Lebacque, J.-P. (1996). The godunov scheme and what it means for first order traffic flow. In Lesort, J., editor, *Transportation and Traffic Theory*, pages 647–677. ISTTT, Elsevier. 100

[Lebacque et al., 2005] Lebacque, J.-P., Haj-Salem, H., and Mammar, S. (2005). Second order traffic flow modeling: supply-demand analysis of the inhomogeneous Riemann problem and of boundary conditions. In *10th EURO Working Group Transportation Meeting*, Poznan, Poland. EURO Working Group on Transportation. 100

[Leveque, 2002] Leveque, R. J. (2002). *Finite Volume Methods for Hyperbolic Problems*. Cambgridge University Press, New York. 8, 9, 43, 45, 54, 92, 194

[Lewis and Shedler, 1979] Lewis, P. A. W. and Shedler, G. S. (1979). Simulation of nonhomogeneous poisson processes by thinning. *Naval Research Logistics Quaterly*, 26:403–413. 134

[Lighthill and Whitham, 1955] Lighthill, M. J. and Whitham, G. B. (1955). On kinematic waves. ii. a theory of traffic flow on long crowded roads. *Proceedings of the Royal Society of London*, A229(1178):317–345. 87

[Mark et al., 2003] Mark, W. R., Glanville, R. S., Akeley, K., and Kilgard, M. J. (2003). Cg: a system for programming graphics hardware in a c-like language. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 896–907, New York, NY, USA. ACM. 62

[Mazarak et al., 1999] Mazarak, O., Martins, C., and Amanatides, J. (1999). Animating exploding objects. In *Proc. Graphics Interface '99*, pages 211–218, Wellesley, MA, USA. AK Peters. 44

[Morse and Patel, 2007] Morse, M. and Patel, J. (2007). An efficient and accurate method for evaluating time series similarity. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 569–580. ACM. 161, 169

[Müller et al., 2003] Müller, M., Charypar, D., and Gross, M. (2003). Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association. 19

[Nagel and Schreckenberg, 1992] Nagel, K. and Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. *Journal de Physique I*, 2(12):2221–2229. 87

[Nealen et al., 2006] Nealen, A., Muller, M., Keiser, R., Boxerman, E., and Carlson, M. (2006). Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836. 20, 30

[Neff and Fiume, 1999] Neff, M. and Fiume, F. (1999). A visual model for blast waves and fracture. In *Proc. Graphics Interface '99*, pages 193–202, Wellesley, MA, USA. AK Peters. 44

[Nelson et al., 1997] Nelson, P., Bui, D., and Sopasakis, A. (1997). A novel traffic stream model deriving from a bimodal kinetic equilibrium. In *Proceedings of the 1997 IFAC meeting, Chania, Greece*, pages 799–804. 88

[Newell, 1961] Newell, G. (1961). Nonlinear effects in the dynamics of car following. *Operations Research*, 9(2):209–229. 87

[Nieuwenhuisen et al., 2004] Nieuwenhuisen, D., Kamphuis, A., Mooijekind, M., and Overmars, M. (2004). Automatic construction of roadmaps for path planning in games. In *Proceedings of the International Conferefence on Computer Games, Artificial Intelligence, Design and Education*, pages 285–292. 116

[O'Brien et al., 2001] O'Brien, D., Fisher, S., and Lin, M. C. (2001). Automatic simplification of particle system dynamics. In *Computer Animation*, pages 210–218. 115

[OpenMP ARB, 2005] OpenMP ARB (2005). OpenMP Version 2.5 Specification. http://www.openmp.org/drupal/mp-documents/spec25.pdf. 35

[OpenStreetMap community, 2010] OpenStreetMap community (2010). OpenStreetMap. http://www.openstreetmap.org/. xvi, 119

[Payne, 1971] Payne, H. J. (1971). Models of freeway traffic and control. *Mathematical Models of Public Systems*, 1:51–60. Part of the Simulation Councils Proceeding Series. 87

[Peeper and Mitchell, 2003] Peeper, C. and Mitchell, J. L. (2003). *Introduction to the DirectX© 9 High Level Shading Language*. Microsoft Corporation and ATI Research. http://ati.amd.com/developer/ShaderX2_IntroductionToHLSL.pdf. 62

[Pettré et al., 2008] Pettré, J., Kallmann, M., and Lin, M. C. (2008). Motion planning and autonomy for virtual humans. In *ACM SIGGRAPH 2008 classes*, pages 1–31. 86

[Prigogine and Andrews, 1960] Prigogine, I. and Andrews, F. C. (1960). A Boltzmann-like approach for traffic flow. *Operations Research*, 8(789). 87

[Redon et al., 2005] Redon, S., Galoppo, N., and Lin, M. C. (2005). Adaptive dynamics of articulated bodies. In *ACM SIGGRAPH 2005 Papers*, page 945. 115

[Reggio, 1982] Reggio, G. (1982). Koyaanisqatsi. Film. MGM Studios. 149

[Reynolds, 1987] Reynolds, C. (1987). Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34. ACM New York, NY, USA. 86

[Richards, 1956] Richards, P. I. (1956). Shock waves on the highway. *Operations Research*, 4(1):42–51. 87

[Roe, 1981] Roe, P. (1981). Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *Journal of Computational Physics*, 43:357. 43, 52, 54

[Roe, 1982] Roe, P. L. (1982). Fluctuation and signals: A framework for numerical evolution problems. In *Proceedings of IMA Conference on Numerical Methods for Fluid Dynamics*, pages 219–257, London. Academic Press. 11, 20

[Roe, 1987] Roe, P. L. (1987). Linear advection schemes on triangular meshes. Technical Report 8720, Cranfield Institute of Technology. 17, 20, 22, 26, 29

[Selle et al., 2005] Selle, A., Rasmussen, N., and Fedkiw, R. (2005). A vortex particle method for smoke, water and explosions. In *ACM SIGGRAPH '05*, pages 910–914. 44

[Settles, 2001] Settles, G. S. (2001). *Schlieren and shadowgraph techniques: Visualizing phenomena in transparent media*. Springer-Verlag. 76

[Sewall et al., 2008] Sewall, J., Galoppo, N., Tsankov, G., and Lin, M. C. (2008). Visual simulation of shockwaves. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2008*, pages 19–28, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association. 12

[Sewall et al., 2009] Sewall, J., Galoppo, N., Tsankov, G., and Lin, M. C. (2009). Visual simulation of shockwaves. *Graphical Models*, 71(4):126–138. 12

[Sewall et al., 2007] Sewall, J., Mecklenburg, P., Mitran, S., and Lin, M. (2007). Fast fluid simulation using residual distribution schemes. In *Eurographics Workshop on Natural Phenomena 2007*, pages 47–54, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association. 11

[Sewall et al., 2010a] Sewall, J., van den Berg, J., Lin, M., and Manocha, D. (2010a). Virtualized traffic: Reconstructing traffic flows from discrete spatio-temporal data. *To appear in IEEE Transactions on Visualization and Computer Graphics*. 86

[Sewall et al., 2010b] Sewall, J., Wilkie, D., Merrell, P., and Lin, M. C. (2010b). Continuum traffic simulation. In *Eurographics 2010*. 13, 86

[Shewchuk, 1994] Shewchuk, J. (1994). An introduction to the conjugate gradient method without the agonizing pain. http://www.cs.cmu.edu/~jrs/jrspapers.html#cg. 31

[Shi and Yu, 2005] Shi, L. and Yu, Y. (2005). Controllable smoke animation with guiding objects. *ACM Trans. Graph.*, 24(1):140–164. 18

[Shvetsov and Helbing, 1999] Shvetsov, V. and Helbing, D. (1999). Macroscopic dynamics of multilane traffic. *Physical Review E*, 59(6):6328–6339. 88

[Si, 2004] Si, H. (2004). Tetgen, a quality tetrahedral mesh generator and three-dimensional delaunay triangulator, v1.3 user's manual. Technical Report 9, Weierstrauss Institute for Applied Analysis and Stochastics. 35

[Song et al., 2005] Song, O.-Y., Shin, H., and Ko, H.-S. (2005). Stable but nondissipative water. *ACM Trans. Graph.*, 24(1):81–97. 18

[Stam, 1999] Stam, J. (1999). Stable fluids. In Rockwood, A., editor, *SIGGRAPH 1999, Computer Graphics Proceedings*, pages 121–128, Los Angeles. Addison Wesley Longman. 18

[Struijs et al., 1991] Struijs, R., Deconinck, H., and Roe, P. L. (1991). Flucuations splitting schemes for the 2d euler equations. *VKI LS*, 1991-01. 20

[Treiber et al., 2000] Treiber, M., Hennecke, A., and Helbing, D. (2000). Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805–1824. 123, 126, 158

[CUDA, 2009] CUDA (2009). *NVIDIA CUDA Reference Manual*. Nvidia Corporation, 2.3 edition. http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/CUDA_Reference_Manual_2.3.pdf. 62

[SUMO, 2009] SUMO (2009). SUMO — Simulation of Urban MObility. http://sumo.sourceforge.net/index.shtml. 108

[U.S. Census Bureau, 2010] U.S. Census Bureau (2010). TIGER/Line®. http://www.census.gov/geo/www/tiger/. xvi, 119

[van den Berg and Overmars, 2007] van den Berg, J. and Overmars, M. (2007). Kinodynamic motion planning on roadmaps in dynamic environments. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4253–4258, San Diego, CA. 116

[van der Weide et al., 1999] van der Weide, E., Deconnick, H., Issman, E., and Degrez, G. (1999). A parallel, implicit, multi-dimensional upwind, residual distribution method for the navier-stokes equations on unstructured grids. *Computational Mechanics*, 23:199–208. 20

[van Leer, 1977] van Leer, B. (1977). Towards the ultimate conservative difference scheme IV: A new approach to numerical convection. *Journal of Computational Physics*, 23:276. 43, 47

[Wendt et al., 2007] Wendt, J., Baxter, W., Oguz, I., and Lin, M. (2007). Finite-volume flow simulations in arbitrary domains. *Graphical Models*, 69(1):19–32. 19

[Whaley et al., 2001] Whaley, R. C., Petitet, A., and Dongarra, J. J. (2001). Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 www.netlib.org/lapack/lawns/lawn147.ps. 66

[Whitham, 1974] Whitham, G. B. (1974). *Linear and nonlinear waves*. John Wiley and Sons, New York, New York. 87

[Yngve et al., 2000] Yngve, G. D., O'Brien, J. F., and Hodgins, J. K. (2000). Animating explosions. In *ACM SIGGRAPH '00*, pages 29–36, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. 44, 77, 78

[Zhang, 2002] Zhang, H. (2002). A non-equilibrium traffic model devoid of gas-like behavior. *Transportation Research Part B*, 36(3):275–290. 87, 88, 91, 94

[Zhu and Bridson, 2005] Zhu, Y. and Bridson, R. (2005). Animating sand as a fluid. In *ACM SIGGRAPH '05*, pages 965–972, New York, NY, USA. ACM Press. 19