DIGITAL ANALOG SIMULATION OF UNIFORM MOTION IN REPRESENTATIONS OF PHYSICAL N-SPACE BY LATTICE-WORK MIMD COMPUTER ARCHITECTURES

A DISSERTATION SUBMITTED TO THE
FACULTY OF THE GRADUATE SCHOOL OF
STATE UNIVERSITY OF NEW YORK
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Anil M. Shende
April 1991

Abstract

This doctoral dissertation is part of an ongoing research project with John Case, Dayanand S. Rajan and myself. We are investigating the possibility of solving problems in scientific computing involving the motion of objects in a bounded region of physical n-space by (a) representing points in the region of space by processors in a lattice-work mesh of processors with local connections for inter-processor communication, and (b) *literally, analogically* simulating the motion of objects by representing the particles of these objects by algorithms which move themselves about in the lattice-work of processors, much as the motion in real space of the particles making up real objects, in effect, constitutes the motion of those objects.

The main contributions of this dissertation are (i) two provably correct algorithms to generate *virtually perfectly shaped* spherical wavefronts emanating from a point source at virtually constant radial speed, (ii) a provably correct algorithm template for simulating the uniform-speed traversal of certain curves by a single particle, and (iii) for the two algorithms of (i) and the template of (ii), the specification of classes of meshes which are excellent discrete approximations to bounded regions of euclidean n-space, and on which these algorithms can be executed. Algorithms for the simulation of uniform-speed translation and uniform angular speed revolution of single particles are presented as specific instances of the algorithm template. A *crucial* aspect of *all* the algorithms in this dissertation is that, in spite of the discreteness of the representation of physical n-space, the *speed* of the simulations, using these algorithms, is approximately *linearly proportional* to the speed of the phenomenon being simulated. Furthermore, all decisions made by processors running these algorithms are based only on *local* information, such as messages from neighboring processors.

Acknowledgements

Several people have contributed to the successful completion of this dissertation, and I am grateful to all of them. I would like to take this opportunity to express my gratitude for some of them in particular.

The original idea that led to this project is due to my thesis advisor, Dr. John Case. I am very grateful to him for his patience with numerous drafts of this thesis, and his guidance and help with "getting truths said right." These sessions with him have been quite an educational experience for me, which, I am sure, will be reflected in my future work. And, using Dr. Case's metaphor, the "important digressions" during these sessions actually harmonised well with the shop-talk, and were instructive in their own right. Of course, had he seriously taken my insistence on using emacs, we would probably have had a lot more time for these important digressions!

I was also extremely fortunate in having a co-advisor, in all respects, in Dr. Dayanand S. Rajan. I am very thankful to him for guiding me, both as a mentor and as a good friend, through the labyrinths of possibilities this project lends itself to so easily. Without his constructive criticism, advice, and wisdom, I would probably still be floundering. Besides, it was so much fun working with him!

On the whole, it is only fair to say that this thesis owes its existence to Dr. Case's conception and Dr. Rajan's direction.

I am grateful to Drs. Sreejit Chakravarty, Mark Fulk, and Russ Miller for reading my thesis at a rather short notice, and for their helpful comments, and to Dr. Sanjay Jain for useful discussions regarding some parts of this project.

I would like to thank Eloise Benzel, Sally Elder and Gloria Koontz in Buffalo, and Vicki Roark in Delaware for all their help and support. I thank Helene Kershner for all her encouragement and advice in my role as a teacher at Buffalo. I am grateful to the Departments of Computer Science at SUNY Buffalo, University of Delaware, and Haverford College for the use of their facilities.

I would like to thank the graduate students at SUNY Buffalo and University of Delaware for providing a wonderful environment to work in; it was quite enjoyable. I would especially like to thank Linda Z. Suri for proof-reading several of my thesis drafts, and giving me helpful suggestions, besides being delightful company during the final stages of completing my dissertation. I would also like to thank Dr. Subhash Suri for some enlightening discussions about Computational Geometry, and for providing me with an excellent survey article regarding Voronoi cells.

As seems to be the tradition with this project, my sessions with Dr. Case to go over the thesis drafts, typically, took the form of endless nights. I am most grateful to our respective wives, Kumza Case and Susheela for being extremely supportive and understanding of our nocturnal existence. Without Susheela's help on the domestic front, it would have been impossible for me to carry out this endeavour to completion.

Finally, I would like to thank my grandparents, parents and my brother for all their help, support and encouragement throughout my life.

Contents

Chapter 1

Introduction

Stepleman [SC$^+$83, Introduction, page v] defines scientific computing as:

> The coordination of mathematics and computing for the modeling of physical reality.

In this dissertation, we assume that physical reality can be modeled as taking place in euclidean n-space, for various n.

A physical phenomenon is, then, a happening in a region of a euclidean space over a period of time. There is a set of objects (e.g., particles, airplanes, waves, geometric forms, etc.) participating in the phenomenon. In a model of the phenomenon, each participating object has a set of associated attributes (such as color, speed, etc.). At each instant in time, in a given time period, the set of participating objects together with their attribute *values* at that time completely describes a snapshot in the development of (the model of) the phenomenon over time. Usually, one of the attributes of each participating object is its location in the associated euclidean space. In most physical phenomena, the location of participating objects changes over time, i.e., most phenomena involve the *motion* of participating objects. We will refer to the development of a physical phenomenon in a euclidean space over time as the *unfolding* of that phenomenon.

Most problems in scientific computing are about phenomena, whose unfolding involves the motion of participating objects in a euclidean space over time. Solutions to these problems usually involve determining (predicting) the attribute values of participating objects over time. When solving such problems using a digital computer, one is restricted to a discrete set of time values at which times, during the unfolding of a phenomenon, the prediction of the attribute values of the participating objects can be done. In some cases (of physical phenomena), closed form solutions, as functions of time, are known for computing the attribute values of participating objects at any given instant in time. On the other hand, there are phenomena, where the only apparent method for predicting the attribute values of the participating objects at any instant in time, is to *simulate* the unfolding of the phenomenon up through that instant in time [Gre82].

We are interested in using parallel computing for the prediction of physical phenomena to solve problems in scientific computing. This dissertation comes out of a research project[1] involving the development of a *literal/analogical* approach for representing and simulating physical phenomena in a representation of the corresponding euclidean space by a lattice-work of processors. The dissertation itself mostly describes and gives proofs of correctness for the algorithms we have designed so far that we expect to be fundamental in simulations of physical phenomena. It also describes a wide class of parallel computers suitable for these algorithms.

We first review, in Section 1.1, some of the current approaches, other than ours, for solving problems in scientific computing using parallelism. There we point out some of the disadvantages we see with these approaches. Then, in Section 1.2, we give an overview of this dissertation.

1.1 Previous Approaches

[AG89, Qui87] indicate that parallel processing is of relevance to at least these scientific computing problems: fluid flow, computing a terrain map based on reflected signals from radar, weather prediction, molecular dynamics, tracing planetary motion, and the N-body problem. Existing applications use analytical models of such phenomena and compute the unfolding. These models involve sets of equations. (See

[1]The current group working on this project consists of Dr. John Case, Dr. Dayanand S. Rajan, and Anil M. Shende.

[AG89].) The attributes of the objects involved (location in space, mass, velocity, acceleration, etc.) are the parameter values for the set of equations in an analytic model for the phenomenon and the methods for solving this set of equations are the procedures.

It is useful, at this time, to give an overview of the different kinds of parallel architectures that are in use currently.

Since the evolution of parallel computing, a large number of parallel architectures have been proposed (e.g., [Ung58, Uhr72, Pea77, Tof77a, BWW81, Dye, Dye81, SS$^+$81, AS84, Hil84, Fun85, PF$^+$85, Sei85]). Of these, some have been simulated in software (e.g., [Fun85]), and some have been built (e.g., [Hil84, Tof84, PF$^+$85, AC$^+$86]; also see [DL81, PU82, PD84, Lev85, Man] for extensive lists of existing machines). Some of these machines dynamically achieve their interconnection patterns with the aid of interconnection (switching) networks (e.g., [Hil84]; see [Sie85] for a discussion of interconnection networks). With respect to current technology, three categories are usually used to describe the size of these architectures [Mil89, Chapter 1]: coarse-grained, medium-grained and fine-grained. Further, they can be divided into three categories with respect to their mode of operation [Mil89, Chapter 1]: Single Instruction Single Data Stream (SISD), Single Instruction Multiple Data stream (SIMD) and Multiple Instruction Multiple Data stream (MIMD); usually, coarse-grained machines operate in MIMD mode and fine-grained machines operate in SIMD mode. Medium-grained machines opt for one of the two modes and some allow the user a choice (for an example of the latter see [SS$^+$81]). All single processor systems typically operate in SISD mode.

Applications running on coarse-grained machines, typically, have a small number of parallel processes (one per processor), where each process is responsible for a suitable portion of the data. A host machine communicates with all the processes, collecting the partial results and combining them into a final result. (See [Mil89, Chapter 1].)

Applications running on fine-grained machines first distribute data among the processors (typically a small portion of data per processor), and then a host machine broadcasts instructions, one at a time, to all the processors simultaneously. At any instant in time, all processors execute the same instruction on their respective data items. The final configuration of data/results among the processors *is* the final result. (See [Mil89, Chapter 1].)

In this dissertation, we are mainly concerned with architectures having a very large number of processors. So, we will not discuss approaches to scientific computing using coarse-grained architectures.

Several architectures have been proposed for medium-grained and fine-grained parallel machines, e.g., [Uhr72, Pea77, BWW81, Dye, Dye81, SS$^+$81, AS84, Hil84, Tof84, PF$^+$85, Sei85]). One of the key motivation for all parallel architectures is to reduce the computing time to solve problems. Some of these architectures were proposed with specific problems in mind (e.g., the pyramid for image processing [Uhr72, Dye, AS84]), but most of them were proposed for general-purpose problem solving. Parallel algorithm design is currently based on the architectural model being used [Qui87]. Data for a problem needs to be distributed appropriately so that algorithms may run *efficiently* on the corresponding target architecture. There is a "semantic gap" between the problem at hand and the architecture being used, and algorithms bridge this gap. Thus, algorithms mold the problem into a framework suitable for the target architecture. We see several disadvantages in such transformations of problems, especially when dealing with a spatial domain (as in scientific computing).

In current solutions to problems in a spatial domain, the representation of real objects is not "direct". As a consequence, the representation lacks some spatial attributes of these objects. For example, the distance, in the architecture being used, between two processors representing two different objects may be completely unrelated to the euclidean distance between the two represented objects in euclidean space (e.g., [AG89, N-body problem on the Cosmic Cube, page 367]). The representation of an object, within a processor, *includes*, as data, the co-ordinates of the object with respect to some fixed frame of reference attached to the region of euclidean space being represented. Algorithms use this information, extracted out of the representation, to compute spatial attributes such as euclidean distance. Therefore, a change in the location of an object in euclidean space gets represented by a change in the *representation* of the object; *not* in the *spatial location* of the representation of the object.

The same processor, over time, represents objects that are at different locations in euclidean space.[2]

One of the fundamental operations in parallel computing, in general as well as for scientific applications, is that of sorting (e.g., sorting points in euclidean space by their x-coordinates is a step in computing the convex hull of a set of points in euclidean space [MS89]—convex hulls are used in topological feature extraction, normalizing patterns in image processing, etc. [MS88]). For sorting, the maximum of the path lengths between any two pieces of data in the architecture becomes of paramount importance because this length places a worst-case lower bound on the amount of time required to move a data item to its destination in the sorted sequence. Keeping the maximum such path length, i.e., the *diameter* of the architecture, as low as possible is clearly, *then*, an important criterion in designing the architectures used in *current* approaches (other than ours). *As a consequence*, these architectures take on topologies that do not satisfy certain properties of euclidean space, which properties *could* be useful in solutions to problems in a spatial domain. For example, in the N-body problem, the force exerted by one body on another is a function of the euclidean distance between the two bodies. In the implementation of the solution on the Cosmic Cube [AG89], these euclidean distances have to be *computed* based on the co-ordinates stored as a part of the representation of each body. This computation is necessary because the path length, in the network, between two processors representing two bodies need not be proportional to the euclidean distance between the two bodies being represented. Thus, the interconnection scheme predominantly serves the purpose of interprocessor communication, but no *spatial* information is being represented by this scheme. For problems in a *spatial domain*, this contributes to the "semantic gap" mentioned earlier.

From the previous discussion, it is clear that, currently, one of the important goals in the design of parallel computers is a low diameter. Algorithms are developed suited to architectures satisfying this goal. This creates the "semantic gap" for spatial domains discussed earlier in this section. As a consequence, for scientific computing,

[2]In Chapter 2 we note the possible advantage of *our* approach for producing simulations (approximately) linear in real–time *and* indicate evidence of potential complexity savings.

which is almost always *in* a spatial domain, algorithms rarely represent a "direct" solution to the problem; they represent a solution to an analytic model of the problem. These models are in some sentential form (system of equations, or perhaps, a list of symbolic rules). Intermediate steps in the solution correspond to intermediate states of the phenomenon (in euclidean space), i.e., intermediate steps in the unfolding of the phenomenon; *but* the time spent in computing an intermediate step of the solution is not generally proportionate to the actual time that the phenomenon takes to reach that corresponding state. This is because the phenomenon is represented as an "artificial" data structure (e.g., a matrix) whose manipulation *analytically* represents the unfolding of the phenomenon.

To maximize the benefits of the underlying architecture, algorithms represent the phenomenon in suitable data structures (matrices on meshes, linked list structures on shared-memory architectures, etc.). Further, these data structures are suitably interpreted depending on the problem. This shifts the focus of the algorithms from representing the *actual unfolding* to representing *data manipulation* which in turn models the unfolding.

Since data regarding the phenomenon is abstracted into some "artificial" data structure, *spatial* relationships between objects participating in the phenomenon are not directly represented in the data structure; they have to be *computed* from the data structure. Thus, for example, inferring whether there is a collision during the unfolding may require an exhaustive search on the data structure (e.g., [AG89, N-body problem]). This needn't be required in our approach.

Problems that have been considered for solutions using current approaches are from a wide variety of domains (graph-theoretic, numerical analysis, computational geometry, scientific computing, etc. [Qui87, AG89, Akl89]). Further, most architectures that have been developed are not domain-specific.[3] Hence, software implementations of algorithms to solve these problems use some general-purpose language

[3] Image processing seems to be the only area for which there are domain-specific parallel architectures, e.g., [Uhr72, Dye, AS84, PF$^+$85]; also see [DL81, PU82, PD84, Lev85] for extensive lists of machines.

available on the machine. Algorithms are not developed with any particular programming language in mind. This is desirable from the point of view of algorithm design, but implementing these algorithms using a particular programming language is not always a trivial task. A large part of the implementation of an algorithm is devoted to implementing the data structures (of the algorithm) in the particular programming language used. This creates *another* virtual level between the problem at hand and an *implementation* of an algorithm for a solution to the problem.

1.2 Overview of Dissertation

We propose a *literal/analogical* approach for solving problems in scientific computing.

Our approach involves

1. representing a finite region of a euclidean space by a suitable mesh-connected computer,[4]

2. representing points in the space by the processors,

3. representing paths in the space by a succession of local communication links between processors, and

4. representing the motion of objects by the movement from processor to processor of the algorithms and data representing the "particles" of the objects.[5]

This dissertation describes a class of mesh computers[6], and algorithms to be run on these mesh computers, with their proofs of correctness, for simulating some simple physical phenomena. Because of the basic nature of these physical phenomena, we

[4]For us a *mesh-connected computer* consists of a finite number of processors locally interconnected for communication. Our use of the term 'mesh-connected computer' is more general than its usage in the literature (see [Mil89]). In particular we do not restrict ourselves to meshes where the interconnection scheme is a square grid with processors at grid points. Sometimes we will refer to a mesh-connected computer as a *mesh computer* or, simply, as a *mesh*.

[5]The algorithms representing such particles would mostly be concerned with getting themselves moved, *with appropriate timing*, to adjacent processors so as to simulate motion of the particles they represent.

[6]See also [CRS90b].

consider these algorithms to be basic to our approach, and expect to use these (or similar) algorithms for simulating the motion of physical objects. The mesh algorithms presented here provide *virtually perfect, virtually constant speed*[7] simulation of n-dimensional

1. spherical wave front progression from a point source,

2. translation of particles, and

3. revolution of particles about points.[8]

In Chapter 2 we elaborate on our approach, discuss motivations for it, and indicate other research contributions in the literature relevant to our approach.

Chapter 3 provides background for the rest of this dissertation.

In Section 3.1 we present notation, terminology, concepts, and definitions from the theory of vector spaces and other mathematical background areas.

If the interconnection scheme of a mesh computer is suitably patterned, we can define the *idealized* version of that mesh computer as its infinite continuation in every direction. In Section 3.2 we consider such idealized mesh computers based on *lattices* [GL87, CS88] and discuss the ill-defined problem of finding "n-dimensional" mesh computer interconnection schemes which provide "optimal" representations of corresponding regions of euclidean n-space.

In Section 3.2.1 lattices are defined, and we discuss the problem of optimally representing euclidean spaces by lattices. Then in Section 3.2.2 we define exactly how we base idealized mesh computers and real mesh computers on lattices and prove a theorem motivating an important restriction we place on lattices associated with real and idealized mesh computers.

---

[7]For the present, it suffices to say that by 'virtually perfect, virtually constant speed' we mean approximately perfect, approximately constant speed where the error in the approximation can be made arbitrarily small. We will formalise this notion in Section 5.1.

[8]Our algorithm, REVOLVE for revolution of particles about points, provides virtually constant *angular* speed. The two algorithms, WAVE and WAVE$'$ for spherical wave front progression from a point source, and the algorithm, TRANSLATE for straight-line translation of particles each achieve mathematically best possible senses of virtually perfect, virtually constant speed (Section 5.1 and Chapter 8); the other, REVOLVE achieves virtually perfect, virtually constant angular speed in an excellent practical sense (Section 5.3).

It is useful at this juncture in the overview of this dissertation to expand on some of the topics in Section 3.2.

Let $\mathbf{R}^n$ denote the set of all n-tuples of real numbers. We consider any vector space isomorphic[9] to $\mathbf{R}^n$ to be a *perfect* representation of euclidean n-space. We choose certain *discrete* sets of points, $\mathcal{D}$, from some perfect representation of euclidean n-space as those to be *explicitly* represented as processors in *idealized* meshes: we choose such $\mathcal{D}$ to be n-dimensional lattices [GL87, CS88].[10] One of the important properties of any lattice (Property 3.2.2) is that there is a *minimal* distance between any two (distinct) points in that lattice. In the idealized mesh associated with a lattice we make an immediate connection between processors iff the euclidean distance between the corresponding lattice points is minimal for the lattice. *Actual* meshes that could conceivably be constructed in the real world can be taken to be *finite*, connected regions of idealized meshes. Our motives behind the choice of n-dimensional *lattices* for $\mathcal{D}$ are two-fold:

1. we would like such $\mathcal{D}$ to represent euclidean n-space *well*, and
2. we would like the *structure* of $\mathcal{D}$ to be suitable for the development of elegant and efficient algorithms (for simulating phenomena in finite submeshes of the corresponding idealized mesh).

We can further explain the choice of lattices as follows. First suppose we decide to solve the problem of discretely representing all of infinite n-dimensional euclidean space; then we represent bounded regions of n-dimensional space by cutting out corresponding, bounded regions of the representation of the whole infinite space. It is plausible to require the discrete representation of all of euclidean n-space be a set of points, $\mathcal{D}$, from a copy of euclidean n-space which has the following three properties.

[9] Vector spaces are discussed formally in Chapter 3 below. All of the isomorphisms we talk about are to be understood as also preserving relative angles geometrically. For example, $\mathbf{R}^2$ is isomorphic to the plane $\{(x, y, z) \in \mathbf{R}^3 \mid x + y + z = 0\}$.

[10] *Roughly*, an n-dimensional *lattice* is a set of points, from some perfect representation of euclidean n-space, providing a discrete variant of an n-dimensional vector space in euclidean n-space. Any point P in a lattice can also be thought of as a vector from the origin to P. As usual, geometrically, we identify any vector with all its translates. In the vector spaces we consider, the scalar multipliers are real numbers which can be thought of as agents for magnifying and shrinking vectors. In lattices the scalar multipliers are *integers*.

1. There is a positive constant μ so that, for each point P of the copy of euclidean n-space, there is a point in the representation $\mathcal{D}$ within a distance μ of P.

2. The points in the representation $\mathcal{D}$ don't cluster arbitrarily close together.

3. Moving from one point in the representation $\mathcal{D}$ to another without changing orientation does not change the structure of the neighborhood of the point in the representation $\mathcal{D}$.

(These informal properties are expressed somewhat more formally as Properties 3.2.1, 3.2.2, and 3.2.3 below.) The connection between these three properties and lattices is that $\mathcal{D}$ satisfies the three properties iff $\mathcal{D}$ is an n-dimensional lattice. (See Theorem 3.2.2.) We discuss this in [CRS90a] and in Chapter 3 below.

In [CRS90a] we also discuss, in detail, a class of meshes associated with lattices having some *additional* especially nice properties. One of these additional properties, motivated, as was the lattice property, by both representing euclidean space well and computational cleanness, is a regularity property. This is a strong symmetry property (Property 3.2.4) which, in effect, says that there is a basis (see Footnote 11) for $\mathcal{D}$ of minimal length vectors with constant angle between any two distinct basis vectors. For n-dimensional lattices, $\mathcal{D}$, Property 3.2.4 is provably equivalent to the Property 1.2.1 immediately below.

Property 1.2.1 *$\mathcal{D}$ has an n element subset $\mathcal{B}$ generating*[11] *$\mathcal{D}$ such that*

1. *each vector in $\mathcal{B}$ is minimal length for vectors in $\mathcal{D}$ and*

2. any *two $k \leq n$ element subsets of $\mathcal{B}$ generate k-dimensional sublattices of $\mathcal{D}$ that are mere (generalized) rotational and/or reflectional variants of one another.*[12]

[11] The *generation* process here involves forming all the vectors obtained by adding together *integer* multiples of vectors from $\mathcal{B}$. A $\mathcal{B} \subseteq \mathcal{D}$ with exactly n elements and which generates the n-dimensional lattice $\mathcal{D}$ is called a *basis* for $\mathcal{D}$. See Chapter 3.

[12] Equivalently, $\mathcal{D}$ has an n element subset $\mathcal{B}$ generating $\mathcal{D}$ such that *any* two $k \leq n$ element subsets of $\mathcal{B}$ generate k-dimensional sublattices of $\mathcal{D}$ that are isometric. (An *isometry* between lattices is an isomorphism which preserves actual *euclidean* distances between corresponding lattice points.)

We will call n-dimensional lattices with Property 1.2.1 (or, equivalently, Property 3.2.4) *regular*. $\mathbf{Z}^n$ and A_n (see [CS88][13]) are examples of n-dimensional regular lattices. Idealized extensions of the meshes from the literature (square meshes [Mil89]) are usually associated with the lattice $\mathbf{Z}^2$. As will be seen below, we are particularly interested in the n-dimensional lattices from the class, $\mathbf{A}_n$, of lattices isomorphic to A_n. It can be shown that regularity of an n-dimensional lattice $\mathcal{D}$ implies the following property.

Property 1.2.2 *$\mathcal{D}$ is an n-dimensional lattice generated by its minimal length vectors, and, if we suppose $\mathcal{B}$ is any set of n vectors from $\mathcal{D}$ such that*

1. *each vector in $\mathcal{B}$ is minimal length for vectors in $\mathcal{D}$ and*

2. *no vector in $\mathcal{B}$ can be generated from the other vectors in $\mathcal{B}$,*

then $\mathcal{B}$ generates all of $\mathcal{D}$.

It is perhaps surprising that there are n-dimensional lattices generated by their minimal length vectors which *fail* Property 1.2.2; the 4-dimensional lattice D_4 from [CS88] is an example. Hence, there is a set $\mathcal{B}$ of 4 vectors in D_4 satisfying both numbered clauses in Property 1.2.2 just above, so that, if one used these vectors as basic steps, forward and backward, for walking around within D_4, one would never get to some points in D_4; the 4 vectors generate a *proper* 4-dimensional sublattice of D_4.

In [CRS90a] it is essentially shown that A_n is the unique (up to isomorphism)[14] n-dimensional regular lattice for which the number of immediate neighbors of any processor in the corresponding idealized mesh is greater than or equal to the number of nearest neighbors of any processor in the idealized mesh based on *any* n-dimensional *regular* lattice. This result is a crucial basis for our special interest in $\mathbf{A}_n$.

A_n also satisfies the following pleasing Property 1.2.3 which fails for some families of regular lattices.

[13] By definition $\mathbf{Z}^n$ is the set of n-tuples of integers, and A_n is the set of $(n+1)$-tuples of integers that add up to 0.

[14] In [CRS90a] we do not make a distinction between A_n and $\mathbf{A}_n$—we treat all lattices in $\mathbf{A}_n$ equivalently.

Property 1.2.3 *There is a basis $\mathcal{B}$ for A_n such that every $k \leq n$ dimensional sublattice of A_n, generated by k vectors from $\mathcal{B}$, is isometric to A_k.*

In Section 3.3 we present results about the prevalence of irrational distances in lattices. These results point to the necessity, in the descriptions of our mesh algorithms above, of the 'virtually' in 'virtually perfect, virtually constant speed' (see Chapter 5).

In Section 3.4 we present the basic definitions and properties for Voronoi cells in lattices. The *Voronoi cell*[15] of any point P in a lattice $\mathcal{D}$ is roughly characterized[16] as the set of points, in the corresponding euclidean space, closer to P than to any other point in $\mathcal{D}$. Intuitively, the Voronoi cell of a lattice point is its *region of sole influence*. As one might expect, Voronoi cells are important to many of our algorithms.

We note here that Voronoi cells (and diagrams) of sets of points in *two* dimensional euclidean space have been studied extensively by researchers in Computational Geometry. Similarly, some line and curve drawing algorithms in graphics turn out to light up a picture element (pixel) on a two dimensional square grid of pixels iff the line or curve passes through the Voronoi cell around the center of the pixel. [Aur90] is an excellent survey of all these applications of Voronoi cells. Almost all these applications are in two dimensions, and not specific to Voronoi cells around points in a lattice. Problems in Computational Geometry usually deal with *creating* Voronoi cells (or diagrams) of a set of points, not with *using* properties of Voronoi cells. We are interested in using the properties of Voronoi cells specifically around points in n-dimensional lattices. Furthermore, we are interested in being able, if possible, to decide membership in the Voronoi cell around a point using *only* integer arithmetic—a concern seemingly irrelevant to the field of Computational Geometry.

It turns out that the algorithms described in this dissertation apply to a rather wide and interesting class of mesh computers. These are described in Chapter 4.

Section 4.1 describes the lattices associated with the idealized versions of the interconnection schemes for the classes of mesh computers sufficient for (most of) our algorithms. These lattices are called *well-represented*. At the beginning of the

[15] The formal definition is Definition 3.4.2. See also [CS88].
[16] See Proposition 3.4.2.

next paragraph we sketch informally the definition of well-represented; the formal definition is Definition 4.1.4. As will be seen in Note 4.1.1, A_n, the lattice we are especially interested in, turns out to be well-represented. We can show that some regular lattices fail to be well-represented (see Note 4.1.1 below); furthermore, many well-represented lattices fail to be regular (e.g., D_4).

Henceforth we shall refer to the minimal length vectors in any lattice generated by them as its *basic directions*.

A lattice $\mathcal{D}'$ is *well-represented* iff[17] $\mathcal{D}'$ is isomorphic to a lattice $\mathcal{D}$ such that

1. $\mathcal{D}$ is some set of tuples of integers,
2. $\mathcal{D}$ is generated by its minimal length vectors (or basic directions), and
3. for each face of the Voronoi cell of any point in $\mathcal{D}$, there is a basic direction in $\mathcal{D}$ perpendicular to that face.

The point of having $\mathcal{D}$ be some set of tuples of integers in the characterization of well-represented just above is that integers are easy to compute with and without loss of information. The point of having $\mathcal{D}$ be generated by its minimal length vectors (or basic directions) is that idealized meshes associated with lattices, the way we associate idealized meshes and lattices, but *without* this property are *disconnected* (Theorem 3.2.5).

In Section 4.2 we present some basic results about Voronoi cells in well-represented lattices.

As will be seen in Chapters 7 and 8, the strictly minimal set of assumptions, on the actual meshes, required for our algorithms depends on the algorithm. In Section 4.3 we specify two classes of assumptions on the hardware of actual mesh computers necessary (but in some cases not sufficient) for our algorithms to work:

1. convexity assumptions on the actual finite mesh computers and simple lattice

[17]See Propositions 3.4.7 and 3.4.9.

restrictions[18] on the idealized meshes in which they are embeddable[19] and

2. general assumptions about the hardware of the processors in a mesh-connected computer.[20]

Below we refer to actual meshes satisfying both classes of assumptions, enumerated just above, as *suitable*.

In Chapter 5 we present some calculations to help us with the analysis of

1. the meaning of virtually in Section 5.1 and

2. resolution of distances (Section 7) and angles (Section 5.3) in certain lattices and meshes based on them.

In Chapter 6 we describe the structure of all our algorithms, including (Section 6.1) their *crucial* general iterative form *with constant execution time for each iteration* and, more generally, the whole algorithm template for our algorithms. This general template provides the framework for what follows.

In Chapter 7 we discuss two algorithms, WAVE and WAVE′, for uniform[21] spherical wave front progression from a point source. Both these algorithms are instances of the general algorithm template presented in Chapter 6. WAVE′ has superior features from the standpoint of simplicity and apparent relevance to actual physical phenomena, but it is presented herein to work only in suitable meshes with associated lattices in $\mathbf{A}_n$ although it can clearly be modified to work in meshes associated with a few

[18]Essentially these restrictions amount to the lattice being isomorphic to a lattice satisfying clauses 1 and 2 in the informal definition above of well-represented.

[19]The algorithm template CURVE (Chapter 8), the basis for the algorithms in Chapter 9, requires more assumptions on the lattices associated with the idealized meshes, namely full well-representedness.

[20]These assumptions are satisfied by our hardware design for individual processors in [CRS90b].

[21]By *uniform* we mean virtually constant speed. Deviations from constant speed are small, *bounded*, and occur only at the processors associated with those lattice points at irrational (relative to the minimal distance) euclidean distance from the source of the wave. Furthermore, if a lattice point P is at such an irrational distance, then the deviation from constant speed in reaching the processor at P will be opposite in sign from that for the processor at lattice point $2P$. See Section 5.1.

other lattices[22] (and their isomorphs). WAVE, on the other hand, works in all suitable meshes.

In Chapter 8 we present an algorithm template for uniform particle traversal of certain curves. It is a refinement of the more general algorithm template of Chapter 6.

In Chapter 9 we discuss two algorithms which are instances of the template of Chapter 8 for the particle traversal of certain curves. One of these algorithms (Section 9.2) is for uniform translation, the other (Section 9.3) for uniform (*angular* speed) revolution about a point. These two algorithms work in suitable meshes *associated with the well-represented lattices*.

Finally in Chapter 10 we present, in a Pascal-like specification language, a detailed listing of the algorithms presented in Chapters 7 and 9.

[22] For example, $\mathbf{Z}^n$.

Chapter 2

Our Methodology

2.1 Our Literal Approach

Our approach to massively parallel computing is meant specifically for scientific computing. As pointed out in the previous chapter, most problems in scientific computing deal with the unfolding of physical phenomena in euclidean space[1]. Such unfolding usually involves the motion of objects in space. Solutions to such scientific problems can be obtained by simulating the phenomenon under consideration. The methodology we employ involves using a mesh-like network of computer processors with local connections, representing points in space by the processors, representing paths in space by a succession of communication links between processors, and representing the motion of objects by the movement from processor to processor of the data and/or algorithms representing the "particles" of the objects. Our meshes *literally/analogically*, but discretely, represent the bounded region of space in which the phenomenon under consideration unfolds over time. Our representation essentially preserves spatial properties, such as euclidean distance, relative angles in space, etc., of the objects participating in the phenomenon. Thus, the representation *directly* provides the simulation with *such* spatial properties of objects; these spatial properties do not have to be *intricately derived* from the representation.

We begin the description of our approach with an example class of meshes. These

[1] **Convention:** Henceforth by "space", we will mean euclidean space.

are *not* the only meshes we are interested in, *but* these meshes are easier to understand for anyone raised on cartesian coordinate systems.

Imagine, for the moment, a large, finite, connected region of 2–dimensional space with an attached cartesian coordinate system. We shall refer to the points with *integer* coordinates lying in the region as *grid points*. Assume one has a large number of identical, spherical, computer processors each of physical diameter less than the unit distance for the coordinate system. Imagine now that there is one of these computer processors sitting, centered, on each grid point and that each such processor is *directly* physically linked (for communication) to all and only its (up to) four nearest neighbors. (At the boundaries of the mesh there will be fewer than four nearest neighbors.) This configuration of connected processors is referred to as a *2–dimensional square mesh computer*. This mesh computer *discretely represents*, then, the finite region of 2–dimensional space whence it comes. Here is how to have this computer *discretely, literally/analogically* simulate the *non-diagonal* motion of, say, one (2–dimensional) "physical" object in (2–dimensional) space. Select a pattern of the grid points, which pattern is (from a distance) approximately shaped like the object. Insert, into all and only the processors attached to those grid points which represent "particles" of the object, an algorithm for behaving like a particle of the object; in this case, where the proper behavior is simple motion of the object, the algorithm for each particle of the object should tell a processor containing it to empty its essential contents into its nearest neighbor in the desired direction of motion. Turn on the mesh computer.

With proper attention to I/O, processor synchrony and bumping into boundaries, one can achieve, in this fashion, a discrete, literal/analogical simulation of the *non-diagonal* motion of our object. Clearly this scenario can be generalized to $n \geq 2$ dimensional space, and the corresponding computer is called an *n–dimensional square mesh computer*.

So far, we have been careful to avoid discussing the case of diagonal motion[2]. Actually (and necessarily), our algorithms (which also work in square meshes) provide

[2]In the case of a square mesh, if an immediate connection is traversed in one time unit, diagonals take two units instead of $\sqrt{2}$ units.

a beautiful solution to the apparent problem of lack of uniform-motion-in-"any"-direction. The problem is solved in part by carefully building in delays in the timing for an algorithm to move to an adjacent processor.

We hope that in general we can obtain simulations which proceed in time virtually linear in the real–time required for the process being simulated to unfold. Of course, this would be blocked if, in the course of a simulation, we had to rescale for better resolution, but, then, perhaps, simulations could still be carried out piecewise virtually linear in real–time. Happily, at least the algorithms we report on herein successfully simulate certain constant speed motions virtually linear in real–time.

Our general approach is related to, but not the same as, the computational models of physics based on cellular automata [Sny47, Hil55, Ung58, Zus69, Tof77a, Tof77b, Fey82, FT82, Min82, Wol83, Mar84, Tof84, Vic84, Mac86, Svo86, SW86, TM87]. Lattice Gas models [Has87] are often studied using cellular automata to simulate complex physical phenomena. In a cellular automaton implementation of a Lattice Gas model, emphasis is placed on the simplicity of the local transition rule for the cellular automaton [FHP86]. Results are essentially computed by the statistical averaging of the states over cells in pre-determined "neighborhoods" in the cellular automaton [Has87]. Our methodology differs from Lattice Gas models in both these regards. We do not restrict ourselves to overly simple cellular automaton-like rules. Unlike individual cells in a cellular automaton, the individual processors in our meshes are computationally more powerful than finite state automata.[3] We are not restricted to averaging over the states of neighbors.

Clearly, the accuracy of our approach depends on (i) the choice of a discrete representation of physical space, and (ii) the timing of each of the algorithms representing the local behavior of the "objects" in the phenomenon being simulated. Of course, since the algorithms must be "spatial" in nature, they will depend on the particular choice of a discrete representation of physical space.

[3] Of course, any physical instance of one of our individual processors must have finite local memory and hence could be modeled as a (rather complex) finite state automaton.

2.2 Motivations For Our Approach

The initial motivation came from some results in Experimental Psychology. The experiments of interest involve human subjects performing tasks ostensibly requiring dynamic *mental* imagery (e.g., *mentally* checking congruence of pairs of complex 3–dimensional shapes [SM71], the *mental* solving of box folding puzzles [SF72], etc.). The human reaction times produced are invariably linear in the real–time to physically carry out actual real–world rotation of shapes for matching, the folding of boxes, etc. [SC82, Kos80, Kos83]. This suggests that if one were to model, *at a high level*, human, non-verbal, spatial cognition, simulations of such spatial operations, as mentioned above, should be linear in real–time. The results of such spatial operations *can* be computed by suitable *matrix* operations to achieve the required geometric transformation of the objects. In the case of rotation through an angle, the timing of standard algorithms for the matrix operations wouldn't be linearly sensitive to the *value* of the angle. Simulations of such tasks using our literal transformations would be (approximately) linear in real–time, matching the human case.

B. Selman, in his survey on reasoning with analogs in Artificial Intelligence [Sel88], classifies spatial knowledge representation by analogs as *vivid* and *pictorial*. Regarding *spatial* (and other) problems [Slo71, For81, For83, Hin79, Bro81, MD84, LS87, Sel88] emphasize the computational complexity savings in certain examples resulting from vivid (and non–vivid) analog processing and/or point out the *in*efficiency of propositional or relational representations. Their arguments apply even more to our literal representations.

Heisenberg [Hei58, pages 164–165] argues that there must exist a universal constant in nature—the smallest length. This might mean that the universe, including space, is discrete. Some of the workers in discrete physics [Sny47, Zus69, Tof77a, Fey82, Min82, Hil55, Svo86, TM87] take this idea seriously. Of course, if the universe is really discrete, it is not clear whether the interconnection scheme of real space is nicely patterned, and if so with what pattern, whether it is unpatterned, whether it changes with time, … [Min82]. Furthermore, real space may be discrete, but admit of infinite regress, i.e., have no minimal distance between points. Herein, we model space simply by computer meshes based on (certain) *lattices* (Section 1.2).

The theory of Cellular Automata [Neu66, Bur70, Cod68] is a clean, theoretical model for representing discrete space, especially from the point of view of massively parallel processing. Several cellular automata based architectures have been proposed and built for problems in discrete physics [TM87] and medical image processing [PD84, Chaps. 10 & 11]. Most of these architectures are 2-dimensional square meshes.

Greenspan and LaBudde [Gre72, Gre82, LG74] have positive results in discretizing newtonian physics analytically. They formulate *discrete mechanics* as a system of difference equations corresponding to the differential equations of newtonian mechanics. This method of discrete mechanics could likely be adapted to be used in our approach. The difference equations of this method could very likely be modified into algorithms for computing "next" position/attributes of particles at each, discretely represented, point in the representation of the actual, euclidean trajectory of motion. We note that the 2-dimensional, *literal*, fluid flow simulations on the **Connection Machine** [Mac86, SW86] were quite successful. This strengthens our belief in the use of literal representations in simulating physical phenomena.

Chapter 3

Preliminaries

In this chapter we define concepts and present useful facts about these concepts that we will use in this document.

In Section 3.1 we set up notation, and define standard terms.

Section 3.2.1 is essentially a summary of [CRS90a]. In that section, we discuss regular lattices and in Section 3.2.2 we discuss idealized meshes based on lattices.

In Section 3.3 we present some technical observations regarding the presence of irrational euclidean distances in lattices. These observations are, necessarily, taken into account in Section 5.1.

In Section 3.4 we formally define, and prove some useful properties of, Voronoi cells (introduced in Section 1.2) in a lattice.

Except where noted otherwise, for proofs of theorems, lemmas, etc. stated in Sections 3.1, 3.2 and 3.4, see [Sim63], [GL87] and [CS88], respectively; these books are general references for concepts defined and discussed in these sections.

3.1 Background Definitions and Notation

We will use the following standard notation throughout the rest of this document.

$\mathbf{N}$ denotes the set $\{0, 1, 2, \ldots\}$.

$\mathbf{Z}$ denotes the set of integers.

$\mathbf{R}$ denotes the set of reals.

$|x|$ for x, a real number, denotes the absolute value of x.

$\mathbf{R}^n$ denotes the set of all n-tuples, $n \in \mathbf{N}$, of real numbers.

$\mathbf{Z}^n$ denotes the set of all n-tuples, $n \in \mathbf{N}$, of integers.

A_n denotes the set of all $(n+1)$-tuples, $(a_0, \ldots, a_n)$, $n \in \mathbf{N}$, of integers satisfying $a_0 + \cdots + a_n = 0$.

$\forall$ denotes the universal quantifier.

$\exists$ denotes the existential quantifier.

$\exists!$ denotes unique existence.

$\in$ denotes the relation of membership in a set.

$\subset$ denotes the relation of proper containment on sets.

$\subseteq$ denotes the relation of containment on sets.

$\cap$ denotes the operation of set intersection.

$\cup$ denotes the operation of set union.

$\cap_P$ denotes the intersection of sets satisfying the predicate P.

$\cup_P$ denotes the union of sets satisfying the predicate P.

$|A|$ for a set A, denotes the cardinality of A, i.e., the number of elements in A.

A^c denotes the complement of the set A in a given universe U.

$A \setminus B$ denotes the set difference of the sets A and B. $A \setminus B = \{x | x \in A, x \notin B\} =$ the complement of B in A.

Definition 3.1.1 *A* vector space, *V, over* $\mathbf{R}$*, is a set (whose elements are called* vectors*), so that*

1. *there is a binary function* $+ : (V \times V) \to V$ *satisfying:*

   (a) $\forall x, y, z \in V, x + (y + z) = (x + y) + z$,

   (b) $\exists!\ 0 \in V, \forall x \in V, x + 0 = 0 + x = x$,

   (c) $\forall x \in V, \exists!\ (-x) \in V, x + (-x) = (-x) + x = 0$,

   (d) $\forall x, y \in V, (x + y) = (y + x)$,

2. *there is a function* $\cdot : (\mathbf{R} \times V) \to V$, *satisfying*

   (a) $\forall \alpha, \beta \in \mathbf{R}, \forall x \in V, (\alpha\beta) \cdot x = \alpha \cdot (\beta \cdot x)$,

   (b) $1 \cdot x = x$,

3. $\forall \alpha \in \mathbf{R}, \forall x, y \in V, \alpha \cdot (x + y) = \alpha \cdot x + \alpha \cdot y$, *and*

4. $\forall \alpha, \beta \in \mathbf{R}, \forall x \in V, (\alpha + \beta) \cdot x = \alpha \cdot x + \beta \cdot x$.

Notation 3.1.1 *For x and y points in a vector space V, we will write*

$$x - y$$

for the point $(x + (-y))$ in V.

Terminology 3.1.1 *In a vector space V, the 0 from 1(b) in Definition 3.1.1 is referred to as the* origin *of V.*

Definition 3.1.2 *Let V be a vector space over $\mathbf{R}$.*

1. *A* linear combination *of vectors $x_1, \ldots, x_k \in V, k \geq 0$ is a vector of the form $\alpha_1 \cdot x_1 + \cdots + \alpha_k \cdot x_k$, where $\forall i, 1 \leq i \leq k, \alpha_i \in \mathbf{R}$.*

2. *$S \subseteq V$, is* linearly independent *iff $\forall n \in \mathbf{N}, \forall x_1, x_2, \ldots, x_n \in S$, $\forall i, 1 \leq i \leq n, \alpha_i \in \mathbf{R}$,*

$$\alpha_1 \cdot x_1 + \alpha_2 \cdot x_2 + \cdots + \alpha_n \cdot x_n = 0 \Longleftrightarrow \forall i, 1 \leq i \leq n, \alpha_i = 0.$$

3. *A maximal, linearly independent subset B of V is called a* basis for V. *If B is a basis for V, then V is said to have* dimension $|B|$. *(The dimension, as defined here, of a vector space is well-defined by Theorem 3.1.2 below.)*

Theorem 3.1.1 *Every vector space has a basis.* □

Theorem 3.1.2 *Every basis for a vector space, V, has the same cardinality.* □

Theorem 3.1.3 *Suppose V is a vector space with basis B. Then, every x in V, $x \neq 0$, is uniquely representable as a linear combination of vectors in B.* □

Example 3.1.1 $\mathbf{R}^n$ *is a vector space of dimension n. For each $i, 1 \leq i \leq n$, let e_i denote the n-tuple, $(e_{i1}, e_{i2}, \ldots, e_{in})$, of integers where $\forall j \neq i, 1 \leq j \leq n, e_{ij} = 0$, and $e_{ii} = 1$. Then, the set of vectors $\{e_1, e_2, \ldots, e_n\}$ is a basis for $\mathbf{R}^n$, and is called the* standard basis *for $\mathbf{R}^n$.*

Definition 3.1.3

1. *For any two elements a and b, $a \leq b$, in $\mathbf{R}$,*

   (a) $[a, b) = \{x \in \mathbf{R} | a \leq x < b\}$,

   (b) $(a, b] = \{x \in \mathbf{R} | a < x \leq b\}$,

   (c) $(a, b) = \{x \in \mathbf{R} | a < x < b\}$, *and*

   (d) $[a, b] = \{x \in \mathbf{R} | a \leq x \leq b\}$.

2. *We define notation for line segments (with/without end-points) in $\mathbf{R}^n$. We explain the geometric meaning of this notation in one case below. For any two points a and b, in $\mathbf{R}^n$*

   (a) $[a, b) = \{x \in \mathbf{R}^n | \exists t \in [0, 1), x = a + t \cdot (b - a)\}$, *i.e., $[a, b)$ is the line segment, in $\mathbf{R}^n$ from a to b including the point a and not including the point b.*

(b) $(a, b] = \{x \in \mathbf{R}^n | \exists t \in (0, 1], x = a + t \cdot (b - a)\}$,

(c) $(a, b) = \{x \in \mathbf{R}^n | \exists t \in (0, 1), x = a + t \cdot (b - a)\}$, *and*

(d) $[a, b] = \{x \in \mathbf{R}^n | \exists t \in [0, 1], x = a + t \cdot (b - a)\}$.

3. *A* metric, *or* distance function *on* $X (\neq \emptyset)$ *is a function* $d : (X \times X) \rightarrow \mathbf{R}$, *satisfying,* $\forall x, y, z \in X$,

   (a) $d(x, y) \geq 0$, *and* $d(x, y) = 0 \iff x = y$,

   (b) $d(x, y) = d(y, x)$, *and*

   (c) $d(x, y) \leq d(x, z) + d(z, y)$ *(the triangle inequality).*

4. *Suppose* V *is a vector space. Then, an* inner product *in* V, *is a function* $(\, , \,) : (V \times V) \rightarrow \mathbf{R}$, *satisfying*

   (a) $\forall \alpha, \beta \in \mathbf{R}, \forall x, y, z \in V, (\alpha \cdot x + \beta \cdot y, z) = \alpha(x, z) + \beta(y, z)$, *and*

   (b) $\forall x, y \in V, (x, y) = (y, x)$.

5. *Suppose* x *and* $y \in \mathbf{R}^n$, *where* $x = x_1 \cdot e_1 + \cdots + x_n \cdot e_n$, *and* $y = y_1 \cdot e_1 + \cdots + y_n \cdot e_n$. *Then, the* standard inner product *in* $\mathbf{R}^n$ *is defined as*

$$(x, y) = \sum_{i=1}^{n} x_i \cdot y_i.$$

   Note 3.1.1 *Henceforth, by* inner product in $\mathbf{R}^n$, *we will mean the standard inner product in* $\mathbf{R}^n$, *with respect to the standard basis for* $\mathbf{R}^n$.

6. *The standard* euclidean length *in* $\mathbf{R}^n$, *denoted* $\| \ \|$, *is defined in terms of the standard inner product in* $\mathbf{R}^n$ *by* $\forall x \in \mathbf{R}^n$, $\|x\| = \sqrt{(x, x)}$, *where* $\sqrt{(x, x)}$ *denotes the positive square root of* (x, x). *This induces a metric,* d, *called* euclidean distance, *on* $\mathbf{R}^n$, *defined by* $\forall x, y \in \mathbf{R}^n, d(x, y) = \|(x - y)\|$.
   $\|(x - y)\|$ *is called the* euclidean distance between x and y.

7. *Two distinct vectors,* x *and* y, *in* $\mathbf{R}^n$ *are* perpendicular *iff* $(x, y) = 0$.

8. S, *a basis for* $\mathbf{R}^n$, *is an* orthonormal basis *iff*

(a) $\forall x \in S, \|x\| = 1,$ *and*

(b) $\forall x_1, x_2 \in S, x_1 \neq x_2, (x_1, x_2) = 0.$

Terminology 3.1.2 *Henceforth, distance will mean euclidean distance.*

Theorem 3.1.4 *The standard basis for* $\mathbf{R}^n$ *is an orthonormal basis.* $\square$

It is useful at this juncture to clarify the connection between the vector space $\mathbf{R}^n$ (from Example 3.1.1) and its standard pictorial, geometric interpretation. When we speak of the objects in $\mathbf{R}^n$ as *points*, we think of these points as being in a standard one-to-one correspondence with the *geometric* points in euclidean n-space. When we speak of the objects in $\mathbf{R}^n$ as *vectors*, we usually think of each of these vectors v *geometrically* as directed (straight) line segments (or arrows), with tail at the geometric point in euclidean n-space corresponding to the n-tuple in $\mathbf{R}^n$ toand with head at the geometric point in euclidean n-space corresponding to the *point* v.

Definition 3.1.4 *Suppose x and y are in $\mathbf{R}^n$. The vector* from x to y *(in $\mathbf{R}^n$) is defined to be the vector $y - x$. We sometimes speak of the vector from x to y as having* base point x.

Geometrically, we can and do *also* think of the vector from x to y as corresponding to a directed (straight) line segment (or arrow), with tail at the geometric point in euclidean n-space corresponding to the *point* x, and with head at the geometric point in euclidean n-space corresponding to the *point* y. Geometrically, this latter arrow is a *translate* of the first arrow we associated with the *point* $y - x$ in $\mathbf{R}^n$, i.e., the arrow with tail at the geometric point in euclidean n-space corresponding to the origin of $\mathbf{R}^n$, and with head at the geometric point in euclidean n-space corresponding to the *point* $y - x$. In contexts in which we speak of the *base point x of a vector v*, we are thinking of v geometrically in the second way, i.e., as an arrow from the geometric point in euclidean n-space corresponding to x to the geometric point in euclidean n-space corresponding to a point y so that $v = y - x$. Of course there is unique such y, and we can clearly take $y = x + v$.

Theorem 3.1.5 *For all $x, y \in \mathbf{R}^n, (x, y) = \|x\| \cdot \|y\| \cdot \cos(\theta)$ where, $(\ , \ )$ is the standard inner product in $\mathbf{R}^n$, and θ is any one of the angles, between the two half-lines, each having an end-point at the origin, one defined by the point x and the origin, and the other by y and the origin.*[1] $\square$

Definition 3.1.5

1. *An orthonormal basis, $S = \{x_1, x_2, \ldots, x_n\}$, for $\mathbf{R}^n$ defines an* orthonormal co-ordinate system, *where 0 is the origin, and the lines $\{\alpha \cdot x_i | \alpha \in \mathbf{R}\}, 1 \leq i \leq n$ are the co-ordinate axes. If $x = \alpha_1 \cdot x_1 + \alpha_2 \cdot x_2 + \cdots + \alpha_n \cdot x_n$, then the* co-ordinates *of x, with respect to S, are $(\alpha_1, \ldots, \alpha_n)$.*

2. *Given an orthonormal co-ordinate system for $\mathbf{R}^n$, the set of points, in $\mathbf{R}^n$, with integer co-ordinates is a* representation for $\mathbf{Z}^n$ with respect to this co-ordinate system*.*

3. *A subset G of $\mathbf{R}^n$ is* convex *iff $\forall a, b \in G$, $[a, b] \subseteq G$.*

4. *Let G be a subset of $\mathbf{R}^n$. Then, the* convex hull of G *is the intersection of all convex supersets, in $\mathbf{R}^n$, of G.*

5. *For all x in $\mathbf{R}^n$, the* open sphere, *$S(x, r)$ (of radius $r > 0$ and centered at x), is given by*
$$S(x, r) = \{y \in \mathbf{R}^n | \|(y - x)\| < r\},$$
and the closed sphere, *$\overline{S(x, r)}$ (of radius $r > 0$ and centered at x), is given by*
$$\overline{S(x, r)} = \{y \in \mathbf{R}^n | \|(y - x)\| \leq r\}.$$

6. *A subset G of $\mathbf{R}^n$ is an* open set *iff $\forall x \in G$, $\exists r \in \mathbf{R}, r > 0$, so that $S(x, r) \subseteq G$. A subset G of $\mathbf{R}^n$ is a* closed set *iff the complement of G in $\mathbf{R}^n$ is an open set. The* closure *of an open set G in $\mathbf{R}^n$ is the intersection of all closed supersets, in $\mathbf{R}^n$, of G.*

[1] In the case where x or y is the origin and, then, one or both half-lines is a single point, the theorem holds for *any* choice of θ.

7. *A countable collection,* $S = \{S_1, S_2, \ldots\}$*, of open subsets of* $\mathbf{R}^n$ *is said to* tessellate $\mathbf{R}^n$ *iff*

   (a) $\mathbf{R}^n = \overline{\cup_{S_i \in S} S_i}$*, and*

   (b) $\forall S_i, S_j \in S, S_i \neq S_j, S_i \cap S_j = \emptyset$.

8. *Let* $n \leq n'$*. Then a function* $E : \mathbf{R}^n \to \mathbf{R}^{n'}$ *is an* isomorphism from $\mathbf{R}^n$ *(onto its image,* $E(\mathbf{R}^n) = \{E(x) \in \mathbf{R}^{n'} \mid x \in \mathbf{R}^n\}$*) iff for any* x, y *in* $\mathbf{R}^n$*, and any real numbers* α, β*,*

   (a) $E(\alpha x + \beta y) = \alpha E(x) + \beta E(y)$*,* *(i.e., E is a linear transformation) and,*

   (b) *there is a real number* $\sigma > 0$ *so that* $\|E(x)\| = \sigma \|x\|$ *(which implies that E preserves angles).*

   The image of an isomorphism from $\mathbf{R}^n$ *is called a* perfect representation of $\mathbf{R}^n$*. If, in 8b above,* $\sigma = 1$*, then we say that E is an* isometry*. (It can be easily shown that isomorphisms, as defined, are one-to-one.)*

Example 3.1.2 *Each open sphere in* $\mathbf{R}^n$ *is a convex set, and so is each closed sphere.*

Theorem 3.1.6 *The union of any collection of open sets in* $\mathbf{R}^n$ *is an open set in* $\mathbf{R}^n$*, and the intersection of a finite collection of open sets in* $\mathbf{R}^n$ *is an open set in* $\mathbf{R}^n$*.* $\square$

We note here that the above definition of "tessellate" is not the same as the usual definition [Cox73], although the basic concept is identical. Our definition is convenient for our purposes.

3.2 Representing Euclidean Space by Meshes

In Section 3.2.1 we motivate (as in [CRS90a]) and formally define the class of regular lattices. In that section we note that amongst n-dimensional regular lattices, lattices in $\mathbf{A}_n$ have the largest number of basic directions.

In Section 3.2.2 we discuss idealized meshes based on lattices. Theorem 3.2.5 motivates our need for lattices that can be generated by their basic directions.

3.2.1 Optimally Representing Euclidean Space

Convention 3.2.1 *From now until Convention 3.2.2 let $\mathcal{D}$ be an arbitrary subset of a perfect representation, $\mathbf{E}^n$, of euclidean n-space.*

We require that $\mathcal{D}$ have the following three reasonable properties so as to be a *good, discrete* representation of euclidean n-space.

Property 3.2.1 THERE IS A REAL NUMBER $\mu = \mu(\mathcal{D}) \geq 0$ SO THAT EVERY OPEN SPHERE[2], IN $\mathbf{E}^n$, WITH RADIUS GREATER THAN μ CONTAINS AT LEAST ONE POINT OF $\mathcal{D}$. *$\mu(\mathcal{D})$ is then called the* measure of proximity of $\mathcal{D}$*.*

Property 3.2.2 POINTS IN $\mathcal{D}$ DO NOT CLUSTER TOO CLOSELY. *That is, there is a real number, $\delta > 0$, so that the distance between any two points in $\mathcal{D}$ is at least δ. If there are two points in $\mathcal{D}$ so that the distance between these two points is δ, then we say $\mathcal{D}$ has minimal distance δ (or that the* minimal distance of $\mathcal{D}$ is δ*).*

Property 3.2.3 *Informally, an observer moved, without change of orientation, from any point in $\mathcal{D}$ to any other point in $\mathcal{D}$ will see the same picture.* $\mathcal{D}$ IS INVARIANT UNDER TRANSLATIONS.

Theorem 3.2.2 is essentially a combination of Theorems 1 and 2 proved in [GL87, Pages 18–19], and characterizes the sets $\mathcal{D}$ that satisfy these three properties. Towards this, we first give the usual definition of lattices, and state some theorems about them.

Definition 3.2.1

$\mathcal{D}'$ is a lattice *iff there is an $n' \in \mathbf{N}$, and a finite, linearly independent subset $S = \{s_1, \ldots, s_k\}$ of the* vector space $\mathbf{R}^{n'}$ *such that*

$$\mathcal{D}' = \{x \in \mathbf{R}^{n'} | \exists \alpha_1, \ldots, \alpha_k \in \mathbf{Z}, x = \alpha_1 \cdot s_1 + \cdots + \alpha_k \cdot s_k\}.$$

[2]Of course, open sphere in $\mathbf{E}^n$ means the isomorphic image of an open sphere in $\mathbf{R}^n$ under an isomorphism from $\mathbf{R}^n$ to $\mathbf{E}^n$.

In this case we say that S is a basis *for the lattice $\mathcal{D}'$ and that the* dimension *of $\mathcal{D}'$ is $|S|$. (The dimension, as defined here, of a lattice is well-defined by Theorem 3.2.1 below.)*
The origin *of $\mathcal{D}'$, in this case, is the origin of $\mathbf{R}^{n'}$.*

Theorem 3.2.1 *Every basis for a lattice has the same cardinality.* □

Theorem 3.2.2 *$\mathcal{D}$ ($\subseteq$ $\mathbf{E}^n$) is an n-dimensional lattice iff $\mathcal{D}$ has properties 3.2.1, 3.2.2 and 3.2.3.* □

Example 3.2.1

1. *For each $n \geq 1$, $\mathbf{Z}^n$ is an n-dimensional lattice. The standard basis for $\mathbf{R}^n$ (see Example 3.1.1 above) is a basis for $\mathbf{Z}^n$. $\mathbf{R}^n$ is the perfect representation of $\mathbf{R}^n$ containing $\mathbf{Z}^n$.*

2. *For each $n \geq 1$, A_n is an n-dimensional lattice. $\forall i \neq j, 0 \leq i, j \leq n$, let $\lambda^n_{ij} = (a_0, \ldots, a_n)$ denote an $(n+1)$-tuple of integers so that (i) $a_i = 1$, (ii) $a_j = -1$, and (iii) for each l, $0 \leq l \leq n, l \notin \{i, j\}, a_l = 0$. Then, $\{\lambda^n_{in} \mid 0 \leq i < n\}$ is a basis for A_n. $\{(x_0, \ldots, x_n) \in \mathbf{R}^{n+1} \mid x_0 + \cdots + x_n = 0\}$ is the perfect representation of $\mathbf{R}^n$ containing A_n.*

3. *Let $x \in \mathbf{R}$ so that $0 < x < 1/2$. Suppose b_1 and b_2 are two vectors of unit length in $\mathbf{R}^2$ so that $(b_1, b_2) = x$. Then, the set of points*

$$\Gamma(b_1, b_2) = \{\alpha_1 \cdot b_1 + \alpha_2 \cdot b_2 \mid \alpha_1, \alpha_2 \in \mathbf{Z}\}$$

is a 2-dimensional lattice with basis $\{b_1, b_2\}$. $\mathbf{R}^2$ is the perfect representation of $\mathbf{R}^2$ containing this lattice.

Notation 3.2.1 *Suppose $\mathcal{D}$ is a lattice. Then, for each $\alpha \in \mathbf{R}$, $\alpha\mathcal{D}$ denotes the set of points $\{\alpha \cdot x | x \in \mathcal{D}\}$.*

Theorem 3.2.3 *Suppose $\mathcal{D}$ is a lattice. Then, for each $\alpha \in \mathbf{R}$, $\alpha\mathcal{D}$ is also a lattice. Furthermore, if $\alpha \neq 0$, $\mathcal{D}$ and $\alpha\mathcal{D}$ have the same dimension.* □

The distance between points, P and Q, in a lattice is the euclidean length of the vector $(Q - P)$. So, the minimal length of vectors in a lattice is exactly the minimal distance of that lattice. We will not make any distinction between the points in a lattice and the corresponding vectors.

Convention 3.2.2 *Henceforth, $\mathcal{D}$ denotes an arbitrary n-dimensional lattice contained in $\mathbf{E}^n$.*

Definition 3.2.2

1. $\mathrm{MinLgth}(\mathcal{D})$ *denotes the minimal distance of* $\mathcal{D}$.
2. *For $\mathcal{D}$, a lattice generated by its minimal length vectors, a vector in $\mathcal{D}$ of length* $\mathrm{MinLgth}(\mathcal{D})$, *is called a* basic direction of $\mathcal{D}$; *the* set of basic directions *in $\mathcal{D}$ is denoted* $\mathrm{Dirs}(\mathcal{D})$.

In [CRS90a], we impose the additional constraint that $\mathcal{D}$ be *regular* in the sense of the following property.

Property 3.2.4 THERE IS A BASIS, $B = \{b_1, \ldots, b_n\}$, FOR $\mathcal{D}$, AND A CONSTANT, $K(\mathcal{D}, B)$, SO THAT THE INNER PRODUCT

$$(b_i, b_j) = \begin{cases} K(\mathcal{D}, B), & \text{if } i \neq j, \\ \mathrm{MinLgth}(\mathcal{D})^2, & \text{if, } i = j. \end{cases}$$

We then say that B is a regular basis *for $\mathcal{D}$, and that $\mathcal{D}$ is* regular.

As noted in Section 1.2, Property 3.2.4 says that there is a basis for $\mathcal{D}$ of minimal length vectors with constant angle between any two distinct basis vectors. As noted in Section 1.2, regularity is important for having a lattice that *nicely* represents the corresponding euclidean space of the same dimension and for computational cleanness. Vector inner product is an *extremely* basic operation in our algorithms; having its calculation simplified is guaranteed by the existence of a regular basis. Furthermore, this makes computations more efficient, since the inner product of any pair of distinct basis vectors is a constant. We also proved, in [CRS90a], essentially the following theorem whose direct consequence is that the n-dimensional lattice A_n (see Example 3.2.1 above) is the unique (up to isomorphism) lattice that, among *regular* lattices, has the *maximum* number of basic directions. Our special interest in A_n derives, in large part, from this result.

Theorem 3.2.4 *Suppose $n > 1$ and suppose $\mathcal{D}$ is a regular lattice with* regular *basis B. Let $k = K(\mathcal{D}, B)$. Then, $|k| \leq \frac{1}{2}\mathrm{MinLgth}(\mathcal{D})^2$ and*

$$|(\mathrm{Dirs}(\mathcal{D}))| = \begin{cases} 2n, & \text{if } -(\frac{1}{n})\mathrm{MinLgth}(\mathcal{D})^2 < k < \frac{1}{2}\mathrm{MinLgth}(\mathcal{D})^2; \\ n(n+1), & \text{if } k = \frac{1}{2}\mathrm{MinLgth}(\mathcal{D})^2; \\ 2(n+1), & \text{if } k = -(\frac{1}{n})\mathrm{MinLgth}(\mathcal{D})^2. \end{cases}$$

□

Note 3.2.1

1. *$\{\lambda^n_{in} \mid 0 \leq i < n\}$ (see Example 3.2.1, part 2. above) is a* regular *basis for A_n. This basis is a subset of $\Lambda^n \stackrel{\mathrm{def}}{=} \{\lambda^n_{ij} \mid 0 \leq i \neq j \leq n\}$—the set of minimal length vectors in A_n; thus, A_n is generated by its minimal length vectors. So $\mathrm{Dirs}(A_n) = \Lambda^n$ and $|(\mathrm{Dirs}(A_n))| = n(n+1)$. The minimal length of vectors in A_n is $\sqrt{2}$.*

2. *For all $b_1, b_2 \in \mathbf{R}^2$ satisfying the condition in part 3 of Example 3.2.1 above, $\Gamma(b_1, b_2)$ is a 2-dimensional regular lattice.*

3.2.2 Idealized Meshes Based on Lattices

A *nearest neighbor* of a point, P, in $\mathcal{D}$, is (by definition) a point, Q, in $\mathcal{D}$, so that Q is at distance $\mathrm{MinLgth}(\mathcal{D})$ from P. Suppose, for the rest of this paragraph, that we have a countably infinite set of processors. For ease of exposition, let each processor in this set be an n-dimensional sphere of (fixed) diameter *less than* $\mathrm{MinLgth}(\mathcal{D})$. *The* *idealized mesh based on $\mathcal{D}$* is constructed as follows. Position processors, from this set, in $\mathbf{E}^n$, so that every point in $\mathcal{D}$ coincides with the center of exactly one processor; then, for every two-element set, $\{P, Q\}$, of points in $\mathcal{D}$, connect the processor with center at P with the processor with center at Q if and only if P and Q are nearest neighbors of each other in $\mathcal{D}$. The *idealized mesh based on $\mathcal{D}$*, denoted $\mathcal{M}(\mathcal{D})$, is, then, the resultant infinite network of locally connected processors. In $\mathcal{M}(\mathcal{D})$, the connection *from* any processor, at a point P in $\mathcal{D}$ *to* one of its nearest (directly connected) neighbors, at a point Q in $\mathcal{D}$, corresponds to the minimal length vector from P to Q.

Given any subset X of $\mathcal{D}$, $\mathcal{M}(X)$ will denote the sub-mesh of $\mathcal{M}(\mathcal{D})$ consisting of all those processors of $\mathcal{M}(\mathcal{D})$ corresponding to points in X, together with all interconnections, which exist in $\mathcal{M}(\mathcal{D})$, between these processors. X is then called the *underlying set* of $\mathcal{M}(X)$.

Naturally, for a physical embedding, in real 3-dimensional space, of an n-dimensional mesh based on a *finite* set $X \subseteq \mathcal{D}$, for $n > 3$, some physical connections will have to be bent. This does not cause any problem since the mesh still functions with bent connections as if it were part of a lattice. Having immediate connections be constant length can help ensure processor synchrony. We show in [CRS90b] that an embedding of $\mathcal{M}(A_n)$ into physical 3-space can be achieved with constant length direct communication links.

We note an important fact about idealized meshes based on lattices, which we will use later in Section 4.3.

Theorem 3.2.5 *The idealized mesh, $\mathcal{M}(\mathcal{D})$, is connected iff $\mathcal{D}$ can be generated by its minimal length vectors.*

PROOF. By the construction, described above, of an idealized mesh based on $\mathcal{D}$, there is a path between any two processors, $\mathcal{M}(P)$ and $\mathcal{M}(Q)$, in the idealized mesh iff there is a finite sequence of points $P = P_1, P_2, \ldots, P_t = Q$, so that each of the vectors $P_{i+1} - P_i, 1 \leq i < t$ is a minimal length vector. This is equivalent to saying that each point in $\mathcal{D}$ can be represented as a sum of integer multiples of the minimal length vectors in $\mathcal{D}$. □

3.3 Euclidean Distances Between Points in Lattices

Lemma 3.3.1 *Suppose $a, b,$ and c are integers, $a, c \neq 0$. Then, $(ax+b)^2 + c$ is a perfect square only for a finite number of integer values of x.*

PROOF. Since $a, b,$ and c are fixed integers, $a, c \neq 0$, there exists $x_0 > 0$, such that $\forall |x| > x_0$,

$$\begin{aligned} |2(ax+b) - 1| &> |c|, \text{ and} && (1) \\ |2(ax+b) + 1| &> |c|. && (2) \end{aligned}$$

Therefore, using (1) and (2), $\forall |x| > x_0$,

$$\begin{aligned} |(ax+b)^2 - ((ax+b)-1)^2| &= |2(ax+b) - 1| > |c|, \text{ and} && (3) \\ |((ax+b)+1)^2 - (ax+b)^2| &= |2(ax+b)+1| > |c|. && (4) \end{aligned}$$

The largest perfect square lesser than and the smallest perfect square greater than $(ax+b)^2$ are $((ax+b)-1)^2$ and $((ax+b)+1)^2$, respectively. Furthermore, from (3) and (4), the absolute difference between each of these two perfect squares and $(ax+b)^2$ is more than $|c|$. Therefore, since $c \neq 0$, $\forall |x| > x_0$, $(ax+b)^2 + c$ is not a perfect square. □

Lemma 3.3.2 *Suppose $\alpha, \beta,$ and γ are rational numbers, $\alpha, \gamma \neq 0$. Let f be a function from integers into rationals, given by*

$$f(x) = (\alpha x + \beta)^2 + \gamma.$$

Then, $f(x)$ is the ratio of two perfect squares only for a finite number of values of x.

PROOF. Suppose

$$\alpha = a_1/a_2, \ \beta = b_1/b_2, \text{ and } \gamma = c_1/c_2, \tag{5}$$

where each of a_1, b_1, and c_1 is an integer, $a_1, c_1 \neq 0$, and each of a_2, b_2, and c_2 is a non-zero integer. Then, substituting, from (5), for each of α, β, and γ in the expression for $f(x)$ and simplifying, we have

$$a_2^4 b_2^4 c_2^2 f(x) = \left(a_1 a_2 b_2^2 c_2 x + b_1 b_2 a_2^2 c_2\right)^2 + c_1 c_2 a_2^4 b_2^4.$$

Now, the right hand side of the above equation is an integer. So, the left hand side must be too. Notice that the form of the right hand side of the equation is $(ax+b)^2+c$, where a, b, and c are integers, $a, c \neq 0$. So, by the previous lemma, there are only a finite number of values of x so that the right hand side of the equation is a perfect square. $\square$

Lemma 3.3.3 *Suppose α, β, and γ are three rational numbers, $\alpha \neq 0$. Let g be a function from integers into rationals given by*

$$g(x) = \alpha^2 x^2 + \beta x + \gamma^2.$$

Then, $(\forall x)[g(x)$ is the ratio of two perfect squares $]$ iff $\beta = \pm 2\alpha\gamma$.

PROOF.

$$\begin{aligned} g(x) &= \alpha^2 x^2 + \beta x + \gamma^2 \\ &= \alpha^2 x^2 + \beta x + \left(\frac{\beta}{2\alpha}\right)^2 + \left(\gamma^2 - \left(\frac{\beta}{2\alpha}\right)^2\right) \\ &= \left(\alpha x + \frac{\beta}{2\alpha}\right)^2 + \left(\gamma^2 - \left(\frac{\beta}{2\alpha}\right)^2\right). \end{aligned}$$

Then, as implied by the previous lemma,

$$(\forall x)[g(x) \text{ is the ratio of two perfect squares] iff } \gamma^2 - \left(\frac{\beta}{2\alpha}\right)^2 = 0.$$

$\square$

Theorem 3.3.1 *Every lattice with dimension greater than 1 contains points at irrational euclidean distances from the origin.*

PROOF. Since every lattice with dimension greater than 1 contains a 2-dimensional sublattice, it suffices to show that the statement of the theorem is correct for 2-dimensional lattices. Suppose $\mathcal{D}'$ is a 2-dimensional lattice with basis $\{b_1, b_2\}$. Let, without loss of generality, $\|b_1\| = 1$. Suppose $\|b_2\|$ is rational; otherwise, we are done. Let $\|b_2\| = p/q$, for some natural numbers p and q, and let θ be the angle between b_1 and b_2. Since b_1 and b_2 form a basis for $\mathcal{D}'$, they cannot be collinear, and hence $|\cos(\theta)| < 1$. Since b_2 is a basis vector, it has non-zero length, and so, $p > 0$. Then, for each natural number x, the euclidean length of the lattice point $b_1 + xq \cdot b_2$ is the square root of $h(k)$, where

$$h(x) = p^2x^2 + 2q(b_1, b_2)x + 1 \tag{6}$$

$$= (px + \cos(\theta))^2 + \sin^2(\theta). \tag{7}$$

By an application of the previous lemma to (6), $h(x)$ is a perfect square for each x iff $\cos(\theta) = \pm 1$. Since $|\cos(\theta)| < 1$, we have the theorem. □

Corollary 3.3.1 *Every lattice with dimension greater than 1 contains an infinite number of points that are at irrational euclidean distances from the origin.*

PROOF. Consider (7) in the proof of the theorem just above. Since $p \neq 0$ and $\cos(\theta) \neq 1$, $\sin^2(\theta) \neq 0$. Therefore, (7) satisfies the conditions of Lemma 3.3.2. So, by Lemma 3.3.2, there are only a finite number of values of x so that $h(x)$, in the above proof, is a perfect square. □

3.4 Voronoi Cells Around Points in Lattices

Since the countable collection of points in $\mathcal{D}$ occur rather sparsely among the continuum of points in $\mathbf{E}^n$, it is convenient to be able to think of each single point of $\mathcal{D}$ as "representing" a whole region around it in $\mathbf{E}^n$. We think of such a region for a point P in $\mathcal{D}$ as a *region of influence of* P (and correspondingly a *region of influence of the processor* $\mathcal{M}(P)$). A region of influence of P might be divided into two disjoint parts: the set of points that P and no other point "represents" (P's *region of sole influence*) and the remaining points (P's *region of shared influence*). It seems natural to let the region of sole influence of a point, P, in $\mathcal{D}$ be the set of those points in $\mathbf{E}^n$ which are *closer*, with respect to euclidean distance, to P than to any other point in $\mathcal{D}$. As will be seen in Proposition 3.4.2 below, the points closer to P than to any other point in $\mathcal{D}$ form what we call the *Voronoi cell* around P in $\mathcal{D}$ (see Definition 3.4.2 below). The boundary of the Voronoi cell around P (see Definition 3.4.2 below) forms the region of shared influence of P. In this section, we define Voronoi cells around points in $\mathcal{D}$, and prove a few technical lemmas about Voronoi cells.

Convention 3.4.1 *We let $\vec{0}$ be the origin of $\mathbf{E}^n$. Hence, $\vec{0}$ is also the origin of $\mathcal{D}$.*

Convention 3.4.2 *$\mathcal{D}^\circ$ denotes $\mathcal{D} \setminus \{\vec{0}\}$.*

Definition 3.4.1 *For all P in $\mathcal{D}^\circ$, the* half-space determined by P, *denoted $H(P, \mathcal{D})$, is given by*

$$H(P,\mathcal{D}) = \{x \in \mathbf{E}^n \mid \|x\| < \|P - x\|\},$$

the closed half-space determined by P, *denoted $\overline{H(P,\mathcal{D})}$, is given by*

$$\overline{H(P,\mathcal{D})} = \{x \in \mathbf{E}^n \mid \|x\| \leq \|P - x\|\}, \text{ and}$$

the boundary of the half-space determined by P, *denoted* Boundary$(H(P,\mathcal{D}))$, *is given by*

$$\text{Boundary}(H(P,\mathcal{D})) = \{x \in \mathbf{E}^n \mid \|x\| = \|P - x\|\}.$$

Lemma 3.4.1 *For each P in $\mathcal{D}^\circ$, the half-space determined by P is a convex, open set in $\mathbf{E}^n$.*

PROOF. This is clear geometrically; we omit the straightforward formal proof. $\square$

Definition 3.4.2

1. *The* Voronoi cell around $\vec{0}$, *denoted* $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$, *is given by*

$$\mathrm{Voronoi}(\vec{0}, \mathcal{D}) = \cap_{P \in \mathcal{D}^\circ} H(P, \mathcal{D});$$

the closure of the Voronoi cell around $\vec{0}$, *denoted* $\overline{\mathrm{Voronoi}(\vec{0}, \mathcal{D})}$, *is given by*

$$\overline{\mathrm{Voronoi}(\vec{0}, \mathcal{D})} = \cap_{P \in \mathcal{D}^\circ} \overline{H(P, \mathcal{D})}, \text{ and}$$

the boundary of the Voronoi cell around $\vec{0}$, *denoted* $\mathrm{Boundary}(\mathrm{Voronoi}(\vec{0}, \mathcal{D}))$, *is given by*

$$\mathrm{Boundary}(\mathrm{Voronoi}(\vec{0}, \mathcal{D})) = \overline{\mathrm{Voronoi}(\vec{0}, \mathcal{D})} \setminus \mathrm{Voronoi}(\vec{0}, \mathcal{D}).$$

2. *For each point P in $\mathcal{D}^\circ$, the* Voronoi cell, closure of the Voronoi cell, *and* boundary of the Voronoi cell *around P, denoted* $\mathrm{Voronoi}(P, \mathcal{D}), \overline{\mathrm{Voronoi}(P, \mathcal{D})}$, *and* $\mathrm{Boundary}(\mathrm{Voronoi}(P, \mathcal{D}))$, *respectively, are defined by*

$$\begin{aligned} \mathrm{Voronoi}(P, \mathcal{D}) &= P + \mathrm{Voronoi}(\vec{0}, \mathcal{D}), \\ \overline{\mathrm{Voronoi}(P, \mathcal{D})} &= P + \overline{\mathrm{Voronoi}(\vec{0}, \mathcal{D})}, \text{ and} \\ \mathrm{Boundary}(\mathrm{Voronoi}(P, \mathcal{D})) &= P + \mathrm{Boundary}(\mathrm{Voronoi}(\vec{0}, \mathcal{D})). \end{aligned}$$

where, for $S \subseteq \mathbf{E}^n$, $P + S = \{P + x \mid x \in S\}$.

In [CS88, GL87], what is meant by "the Voronoi cell around a point P" is what we mean by "the closure of the Voronoi cell around P". We define Voronoi cells as above only for our convenience.

The following three propositions are now immediate consequences from the definition of Voronoi cells around lattice points.

Proposition 3.4.1 *Each Voronoi cell around a lattice point is a translate of the Voronoi cell around every other lattice point.* □

So, it suffices to define concepts, and prove properties of Voronoi cells, in terms of $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$.

Proposition 3.4.2 *For each point, P, in $\mathcal{D}$, the Voronoi cell around P is the set of points in $\mathbf{E}^n$ that are closer to P than any other point in $\mathcal{D}$.* □

Proposition 3.4.3 $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$ *is convex.* □

Lemma 3.4.2 $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$ *is bounded.*

PROOF. Suppose $x \in \mathbf{E}^n$ is in $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$. Then, x is closer to $\vec{0}$ than to any other point in $\mathcal{D}$. Since $\mathcal{D}$ is a lattice, x is at most μ units from some point in $\mathcal{D}$, where μ is the measure of proximity of $\mathcal{D}$. So, x is at most μ units away from $\vec{0}$. Therefore $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$ is contained in an open sphere, in $\mathbf{E}^n$, centered at $\vec{0}$, and of radius μ. Therefore, $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$ is bounded. □

Definition 3.4.3

1. *The* set of relevant vectors in $\mathcal{D}$, *denoted* $\mathrm{Rel}(\mathcal{D})$, *is the smallest set of vectors in $\mathcal{D}$ so that*
$$\cap_{R \in \mathrm{Rel}(\mathcal{D})} H(R, \mathcal{D}) = \mathrm{Voronoi}(\vec{0}, \mathcal{D}).$$
2. *For each vector $R \in \mathrm{Rel}(\mathcal{D})$, the* face, corresponding to R, of $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$, *denoted* $\mathrm{Face}_R(\vec{0}, \mathcal{D})$, *is given by*
$$\mathrm{Face}_R(\vec{0}, \mathcal{D}) = \{x \in \overline{\mathrm{Voronoi}(\vec{0}, \mathcal{D})} \mid (x, R) = (R, R)/2\}.$$

Note 3.4.1 *It can be shown [CS88, Cox73] that* $\mathrm{Face}_R(\vec{0}, \mathcal{D})$, *for any $R \in \mathrm{Rel}(\mathcal{D})$, is a bounded, $(n-1)$-dimensional planar surface perpendicular to R.*

We now prove a few facts about relevant vectors in an n-dimensional lattice.

Theorem 3.4.1 $\mathrm{Rel}(\mathcal{D})$ *is finite.*

PROOF. The closure of the Voronoi cell around the origin is contained within an closed sphere, in $\mathbf{E}^n$, centered at the origin, of radius equal to the measure of proximity, μ of $\mathcal{D}$. Therefore, every face of the Voronoi cell around the origin is at most μ units away from the origin. Therefore, every relevant vector is of length at most 2μ, and hence contained in an closed sphere, in $\mathbf{E}^n$, centered at the origin, of radius 2μ. Then, since $\mathcal{D}$ is a lattice, there are only a finite number of points of $\mathcal{D}$ within any closed sphere, in $\mathbf{E}^n$, centered at the origin. □

Proposition 3.4.4

$$x \in \overline{\text{Voronoi}(\vec{0}, \mathcal{D})} \Longleftrightarrow (\forall R \in \text{Rel}(\mathcal{D})), (x, R) \leq (R, R)/2.$$

PROOF. The proof follows from Definitions 3.4.3 and 3.4.1.

$$\begin{aligned}
x \in \overline{\text{Voronoi}(\vec{0}, \mathcal{D})} &\Longleftrightarrow x \in \cap_{R \in \text{Rel}(\mathcal{D})} \overline{H(R, \mathcal{D})} \\
&\Longleftrightarrow (\forall R \in \text{Rel}(\mathcal{D}))\|x\| \leq \|R - x\| \\
&\Longleftrightarrow (\forall R \in \text{Rel}(\mathcal{D}))(x, x) \leq ((R - x), (R - x)) \\
&\Longleftrightarrow (\forall R \in \text{Rel}(\mathcal{D}))(x, x) \leq (R, R) + (x, x) - 2(R, x) \\
&\Longleftrightarrow (\forall R \in \text{Rel}(\mathcal{D}))(R, x) \leq (R, R)/2.
\end{aligned}$$

□

Proposition 3.4.5 *For each point, P, in $\mathcal{D}$,* Voronoi$(P, \mathcal{D})$ *is an open set in* $\mathbf{E}^n$*.*

PROOF. For each P in $\mathcal{D}$, Voronoi$(P, \mathcal{D})$ is, by Definition 3.4.3, the intersection of the half-spaces with respect to the relevant vectors in $\mathcal{D}$. By Lemma 3.4.1, each of these half-spaces is an open set in $\mathbf{E}^n$. Then, by Lemma 3.4.1, Voronoi$(P, \mathcal{D})$ is a finite intersection of open sets in $\mathbf{E}^n$, and so by Theorem 3.1.6, is an open set in $\mathbf{E}^n$. □

Theorem 3.4.2 *Voronoi cells around points in $\mathcal{D}$ tessellate*[3] $\mathbf{E}^n$*.*

[3] The concept of tessellation of $\mathbf{R}^n$ from Definition 3.1.5 is easily extended to tessellation of $\mathbf{E}^n$.

PROOF. By Proposition 3.4.2, the Voronoi cell around a point, P, in $\mathcal{D}$, is the set of points in $\mathbf{E}^n$ closer to P than any other point in $\mathcal{D}$. Therefore, Voronoi cells around any two distinct points in $\mathcal{D}$ have an empty intersection. Every point in $\mathbf{E}^n$ is closest to some subset, S, of points in $\mathcal{D}$, and is hence, in the closure of the Voronoi cell around each of the points in S. By Proposition 3.4.5, the Voronoi cell around any point in $\mathcal{D}$ is an open set in $\mathbf{E}^n$. $\square$

Proposition 3.4.6 *For each minimal length vector, V, in $\mathcal{D}$,*

$$\frac{1}{2} \cdot V \in \text{Boundary}(\text{Voronoi}(\vec{0}, \mathcal{D})).$$

PROOF. Let V be a minimal length vector in $\mathcal{D}$. It suffices to show the following claims.

1. $\frac{1}{2} \cdot V \in \overline{\text{Voronoi}(\vec{0}, \mathcal{D})}$, and
2. $\exists P \in \mathcal{D}, P \neq \vec{0}$, so that $\frac{1}{2} \cdot V \in \overline{\text{Voronoi}(P, \mathcal{D})}$.

Proof Claim 1 Suppose, by way of contradiction,

$$\frac{1}{2} \cdot V \notin \overline{\text{Voronoi}(\vec{0}, \mathcal{D})}. \tag{8}$$

$\vec{0}$ and V are equidistant from $\frac{1}{2} \cdot V$, each at a euclidean distance of $\frac{1}{2} \cdot \|V\|$. Therefore, by (8) and Proposition 3.4.2,

$$\frac{1}{2} \cdot V \notin \overline{\text{Voronoi}(V, \mathcal{D})}. \tag{9}$$

Therefore, by Theorem 3.4.2, there is a point P, in $\mathcal{D}$, distinct from $\vec{0}$ and V, so that

$$\frac{1}{2} \cdot V \in \overline{\text{Voronoi}(P, \mathcal{D})}. \tag{10}$$

Then, by (8) and Proposition 3.4.2, $d(P, \frac{1}{2} \cdot V) < \frac{1}{2}\|V\|$. Furthermore, since V is a minimal length vector in $\mathcal{D}$, $\|P\| \geq \|V\|$.
Hence, the points $\vec{0}$, $\frac{1}{2} \cdot V$, and P form a triangle and

$$\|P\| \geq \|V\| > \|\frac{1}{2} \cdot V\| + d(P, \frac{1}{2} \cdot V)$$

violating the triangle inequality in the definition of the distance metric and thus giving us a contradiction.

Proof Claim 2 As a consequence of Claim 1 and Proposition 3.4.2, it is easy to see that

$$\frac{1}{2} \cdot V \in \overline{\mathrm{Voronoi}(V, \mathcal{D})}.$$

□

Now, from the above proposition and Proposition 3.4.2, it follows that

Corollary 3.4.1 *For each minimal length vector, V, in $\mathcal{D}$,*

$$\forall \alpha \in [0, \frac{1}{2}],\ \ \alpha \cdot V \in \overline{\mathrm{Voronoi}(\vec{0}, \mathcal{D})}.$$

□

Proposition 3.4.7 characterizes the set of relevant vectors in a lattice, Proposition 3.4.9 states that the set of minimal length vectors in a lattice is a subset of the relevant vectors in that lattice, and Proposition 3.4.10 shows that faces of Voronoi cells corresponding to relevant vectors are non-trivial.

Proposition 3.4.7

$$R \in \mathrm{Rel}(\mathcal{D}) \Longleftrightarrow \mathrm{Voronoi}(\vec{0}, \mathcal{D}) \subset \cap_{R \neq R' \in \mathcal{D}^\circ} H(R', \mathcal{D}).$$

PROOF. Expressing the statement in set notation, we need to show that

$$\begin{aligned} R \in \mathrm{Rel}(\mathcal{D}) \Longleftrightarrow \exists x \in \mathbf{E}^n, \forall R' \in \mathcal{D}^\circ, R \neq R', \\ [(|(x, R')| \leq (R', R')/2) \text{ and } ((x, R) > (R, R)/2)]. \end{aligned} \tag{11}$$

($\Longrightarrow$) Suppose $R \in \mathrm{Rel}(\mathcal{D})$. Suppose, by way of contradiction,

$$\begin{aligned} \forall x \in \mathbf{E}^n, \exists R' \in \mathcal{D}^\circ, R \neq R', \\ [(|(x, R')| \leq (R', R')/2) \Longrightarrow ((x, R) \leq (R, R)/2)]. \end{aligned}$$

Then,

$$\cap_{R' \in \mathcal{D}^\circ, R' \neq R} H(R', \mathcal{D}) \subseteq \mathrm{Voronoi}(\vec{0}, \mathcal{D}).$$

Then, by the minimality condition on the set of relevant vectors, $R \notin \mathrm{Rel}(\mathcal{D})$, giving us a contradiction.

($\Longleftarrow$) Suppose the right hand side of (11). Therefore, for some $x \in \mathbf{E}^n$,

$$\forall R' \in \mathcal{D}^\circ, R \neq R', [|(x, R')| \leq (R', R')/2], \tag{12}$$

and

$$[(x, R) > (R, R)/2]. \tag{13}$$

Suppose, by way of contradiction, $R \notin \mathrm{Rel}(\mathcal{D})$. Then, by (13), x is closer to R than to $\vec{0}$; hence, by Proposition 3.4.2,

$$x \notin \mathrm{Voronoi}(\vec{0}, \mathcal{D}).$$

But, since $R \notin \mathrm{Rel}(\mathcal{D})$, by (12) and the definition of the set of relevant vectors in $\mathcal{D}$ (Definition 3.4.3),

$$x \in \mathrm{Voronoi}(\vec{0}, \mathcal{D})$$

giving us a contradiction.

□

The following Proposition is immediate from the definition of the faces of a Voronoi cell.

Proposition 3.4.8 *If* $R \in \mathrm{Rel}(\mathcal{D})$ *then,* $\frac{1}{2} \cdot R \in \mathrm{Face}_R(\vec{0}, \mathcal{D})$. □

Proposition 3.4.9 *Every minimal length vector in* $\mathcal{D}$ *is in* $\mathrm{Rel}(\mathcal{D})$.

Proof. Suppose V is a minimal length vector in $\mathcal{D}$. Then, by Proposition 3.4.6,

$$\frac{1}{2} \cdot V \in \mathrm{Boundary}(\mathrm{Voronoi}(\vec{0}, \mathcal{D})). \tag{14}$$

Suppose, by way of contradiction, $V \notin \mathrm{Rel}(\mathcal{D})$. Then, since (14), there exists $R \in \mathrm{Rel}(\mathcal{D})$ such that

$$\frac{1}{2} \cdot V \in \mathrm{Face}_R(\vec{0}, \mathcal{D}). \tag{15}$$

Also, by Proposition 3.4.8,

$$\frac{1}{2} \cdot R \in \mathrm{Face}_R(\vec{0}, \mathcal{D}). \tag{16}$$

Then, using (15), (16) and part 2 of Definition 3.4.3 above, we have that the points $\vec{0}$, $\frac{1}{2} \cdot R$ and $\frac{1}{2} \cdot V$ form a right triangle with $[\vec{0}, \frac{1}{2} \cdot V]$ as its hypotenuse.
Since V is a minimal length vector in $\mathcal{D}$,

$$\|\frac{1}{2} \cdot R\|^2 \geq \|\frac{1}{2} \cdot V\|^2.$$

Also, since V and R are distinct points in $\mathcal{D}$,

$$d(\frac{1}{2} \cdot V, \frac{1}{2} \cdot R) > 0.$$

Then,

$$\|\frac{1}{2} \cdot R\|^2 + (d(\frac{1}{2} \cdot V, \frac{1}{2} \cdot R))^2 > \|\frac{1}{2} \cdot V\|^2,$$

giving us a contradiction to Pythagoras' theorem. □

Now, from the proof of the above Proposition, we have the following Corollary.

Corollary 3.4.2 *Suppose V is a minimal length vector in $\mathcal{D}$. Then,*

$$(\forall R \in \mathrm{Rel}(\mathcal{D}) \textit{ such that } R \neq V), \frac{1}{2} \cdot V \notin Face_R(\vec{0}, \mathcal{D}).$$

□

Lemma 3.4.3 *Suppose $R \in \mathrm{Rel}(\mathcal{D})$, and $x \in \mathrm{Face}_R(\vec{0}, \mathcal{D})$. Then, for all $\alpha \in \mathbf{R}, \alpha > 1, (\alpha \cdot x, R) > (R, R)/2$.*

PROOF. Immediate from the definition of $\mathrm{Face}_R(\vec{0}, \mathcal{D})$. □

Proposition 3.4.10 *Suppose R is a relevant vector in $\mathcal{D}$. Then there is a point, x, on the face corresponding to R, of $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$, so that x is not in any other face of $\mathrm{Voronoi}(\vec{0}, \mathcal{D})$.*

PROOF. Suppose R is a relevant vector in $\mathcal{D}$. Then, as implied by Proposition 3.4.7,

$$\exists x \in \mathbf{E}^n, \forall R' \in \mathrm{Rel}(\mathcal{D}), R \neq R',$$
$$[(|(x, R')| \leq (R', R')/2) \text{ and } ((x, R) > (R, R)/2)].$$

Therefore, $\exists \alpha \in \mathbf{R}, \alpha < 1$ so that $(\alpha \cdot x, R) = (R, R)/2$.
Suppose, by way of contradiction, $\exists R' \in \text{Rel}(\mathcal{D}), R' \neq R$ so that $\alpha \cdot x \in \text{Face}_{R'}(\vec{0}, \mathcal{D})$, i.e., $(\alpha \cdot x, R') = (R', R')/2$. Then, by Lemma 3.4.3, $\forall \beta \in \mathbf{R}, \beta > \alpha$,

$$(\beta/\alpha)(\alpha \cdot x, R') = (\beta \cdot x, R') > (R', R')/2.$$

In particular, for $\beta = 1, (x, R') > (R', R')/2$, giving us a contradiction. $\square$

Proposition 3.4.11 *Suppose x and y are two points in $\mathbf{E}^n$. Then, there exists a finite subset, S, of points in $\mathcal{D}$ such that $[x, y]$, and therefore each of (i) (x, y), (ii) $[x, y)$, and (iii) $(x, y]$, is contained in $\overline{\cup_{P \in S} \text{Voronoi}(P, \mathcal{D})}$.*

PROOF. $[x, y]$ is bounded and closed. Hence, there is a finite region of $\mathbf{E}^n$ that contains $[x, y]$. $\square$

Chapter 4

Well-Represented Lattices and Meshes

In Chapter 1.2 we informally introduced a class of lattices called well-represented. In Section 4.1 below, we formally define three classes of lattices: *minimally generated* lattices (Definition 4.1.2), *integrally-representable, minimally generated* lattices (Definition 4.1.3) and *well-represented* lattices (Definition 4.1.4). Each of the latter two classes of lattices is a subset of its predecessor in the sequence. We also set up notation for some concepts associated with lattices in these classes.

As will be seen from the definition of well-represented lattices, Voronoi cells play an important role in characterizing well-represented lattices. In Section 4.2 we prove some useful properties of Voronoi cells in these lattices.

As noted in Chapter 1.2, actual meshes could be taken to be finite, connected regions of idealized meshes. The assumptions on actual meshes specified in Section 4.3 are necessary for each of our algorithms. Recall from Section 1.2 that we call actual meshes satisfying these assumptions *suitable* meshes. Section 4.3.1 specifies our assumptions on the *structure*, and Section 4.3.2 specifies the assumptions, in general, on the *hardware* of suitable meshes.

Algorithm WAVE works in suitable meshes. As will be seen in Section 4.3.1, the idealized meshes corresponding to suitable meshes are associated with integrally-representable, minimally generated lattices. Algorithms that are refinements of Algorithm CURVE (see Chapter 8), work in a class of actual meshes more restrictive than the class of actual meshes suitable for Algorithm WAVE. As pointed out in Section 1.2, the restriction is in the structure of the corresponding idealized mesh—for these algorithms the corresponding idealized mesh must be associated with a well-represented lattice.

4.1 Well-Represented Lattices

Recall that by Convention 3.2.2, $\mathcal{D}$ is an arbitrary n-dimensional lattice contained in $\mathbf{E}^n$, a perfect representation of $\mathbf{R}^n$.

Definition 4.1.1 *Suppose $\mathcal{D}'$ and $\mathcal{D}''$ are lattices. $\mathcal{D}'$* is isomorphic to *$\mathcal{D}''$ iff there is a bijective mapping $G : \mathcal{D}' \to \mathcal{D}''$ such that, for any X, Y in $\mathcal{D}'$ and any* integers *α, β,*

1. *$G(\alpha X + \beta Y) = \alpha G(X) + \beta G(Y)$, and*
2. *there is a* real *number σ so that $\|G(X)\| = \sigma\|X\|$ (which implies that G preserves angles).*

If, in 2 above, $\sigma = 1$, then we say that G is an isometry.

Every lattice isomorphism uniquely extends to an isomorphism between the smallest corresponding euclidean spaces containing the lattices.

Definition 4.1.2 *$\mathcal{D}$ is* minimally generated *iff $\mathcal{D}$ is generated by its set of minimal length vectors.*

Definition 4.1.3 *$\mathcal{D}$ is* integrally-representable, minimally generated *iff there is an $n' \geq n$, and an isomorphic (n-dimensional) lattice $\mathcal{D}'$ so that*

1. *$\mathcal{D}'$ is minimally generated, and*

2. *$\mathcal{D}'$ is a sublattice of $\mathbf{Z}^{n'}$.*

We then say that $\mathcal{D}'$ integrally-represents *$\mathcal{D}$* in *$\mathbf{Z}^{n'}$.*

Definition 4.1.4 *$\mathcal{D}$ is* well-represented *iff*

1. *$\mathcal{D}$ is integrally-representable, minimally generated (as witnessed by $\mathcal{D}'$, a subset of $\mathbf{Z}^{n'}$), and*

2. *the set of relevant vectors in $\mathcal{D}$ is exactly the set of basic directions of $\mathcal{D}$.*

We then say that $\mathcal{D}'$ well-represents *$\mathcal{D}$* in *$\mathbf{Z}^{n'}$.*

Note 4.1.1

1. *Clearly, for each $n' \in \mathbf{N}$, $\mathbf{Z}^{n'}$ is a well-represented lattice.*

2. *The n-dimensional sublattice, A_n, of $\mathbf{Z}^{n+1}$, consisting of all those points of $\mathbf{Z}^{n+1}$ whose individual co-ordinates add up to zero, is a well-represented lattice.*

3. *We can show that, for all $b_1, b_2 \in \mathbf{R}^2$ satisfying the condition in part 3 of Example 3.2.1 above, the 2-dimensional regular lattice $\Gamma(b_1, b_2)$ (see part 2 of Note 3.2.1) fails Requirement 2 in Definition 4.1.4 above, and hence fails to be a well-represented lattice.*

Convention 4.1.1 *Henceforth, $\mathcal{L}$ will denote an arbitrary n-dimensional lattice, $n > 1$, which well-represents* itself *in $\mathbf{Z}^{n'}$, for $n' \geq n$. Each point (vector) in $\mathcal{L}$ can thus be represented as a unique n'-tuple of integers. We suppose $\mathbf{E}^n$ is the perfect representation of $\mathbf{R}^n$ that contains $\mathcal{L}$.*

It is also convenient to have the following

Convention 4.1.2 *Henceforth, $\mathcal{K}$ will denote an arbitrary n-dimensional lattice, $n > 1$, which integrally-represents* itself *in $\mathbf{Z}^{n''}$, for $n'' \geq n$. Each point (vector) in $\mathcal{K}$ can thus be represented as a unique n''-tuple of integers.*

We note the following important properties of $\mathcal{K}$.

Property 4.1.1 EACH VECTOR IN $\mathcal{K}$ CAN BE REPRESENTED USING INTEGERS ONLY.

Property 4.1.2 THE USUAL INNER PRODUCT OF ANY TWO VECTORS, IN $\mathcal{K}$, IS AN INTEGER, AND HENCE THE SQUARE OF THE EUCLIDEAN DISTANCE BETWEEN ANY TWO POINTS (VECTORS), IN $\mathcal{K}$, IS AN INTEGER. *This is important for us because we wish to avoid computations that directly involve irrational quantities. Since inner-products will be ubiquitous in our calculations, having these appear as ("easily calculable") integers is very useful.*

Property 4.1.3 THE COSINE AND THE SQUARES OF THE COSINES AND SINES OF THE ANGLE BETWEEN ANY TWO BASIC DIRECTIONS IN $\mathcal{K}$ ARE RATIONAL NUMBERS. *This is useful for us, as will be seen in Chapter 8.*

Of course, $\mathcal{L}$ shares these properties with $\mathcal{K}$. It is easy to show from Theorem 3.4.2, Proposition 3.4.6, Corollary 3.4.1 and Corollary 3.4.2 that $\mathcal{L}$, by virtue of it well-representing itself, has the following additional property.

Property 4.1.4 TRAVERSING A DIRECT CONNECTION IN AN (IDEALIZED) MESH BASED ON $\mathcal{L}$ IMPLIES CROSSING THE BOUNDARY OF EXACTLY TWO ADJACENT VORONOI CELLS. *We will use this fact, in Chapter 8, in our algorithm template for simulating the traversal of a curve by a particle.*

We now define some concepts for minimally generated lattices. Of course, these concepts, then, apply to integrally-representable, minimally generated lattices and well-represented lattices as well.

Definition 4.1.5 *Suppose $\mathcal{D}$ is minimally generated. Suppose P and O are distinct points in $\mathcal{D}$.*

1. *The* nearest neighbors of P*, denoted* $\mathrm{Nbrs}(P)$*, is the set of points Q, in $\mathcal{D}$, so that the vector $(Q - P)$ is a basic direction of $\mathcal{D}$.*

2. *The* Graphical Path Length of P with respect to O, *denoted* GraphPL(P,O), *is defined as:*

$$\mathrm{GraphPL}(P,O) = \begin{cases} 0, & \text{if } P = O; \\ 1 + \min\{\mathrm{GraphPL}(Q,O) | Q \in \mathrm{Nbrs}(P)\}, & \text{otherwise.} \end{cases}$$

3. N *is an* Inner Neighbor of P with respect to O, *iff* $N \in \mathrm{Nbrs}(P)$, and

$$\mathrm{GraphPL}(N,O) = \mathrm{GraphPL}(P,O) - 1.$$

   $Inner_{\mathcal{D}}(P,O)$ *is the set of all points* N*, such that* N *is an inner neighbor of* P *with respect to* O*.*

4. N *is an* Outer Neighbor of P with respect to O, *iff* $N \in \mathrm{Nbrs}(P)$, and

$$\mathrm{GraphPL}(N,O) = \mathrm{GraphPL}(P,O) + 1.$$

   $Outer_{\mathcal{D}}(P,O)$ *is the set of all points* N*, such that* N *is an outer neighbor of* P *with respect to* O*.*

5. *A sequence of points* $P = P_0, P_1, \ldots P_m = Q$, $P_i \in \mathcal{D}, 0 \le i \le m$, *is a* shortest path from P to Q, *iff* $\forall i, 0 < i \le m, P_{i-1} \in Inner_{\mathcal{D}}(P_i, P)$.
   N.B. $\mathrm{GraphPL}(P,Q) = m = \mathrm{GraphPL}(Q,P)$.

6. *The* norm *of* P *with respect to* O*, denoted* Norm(P,O)*, is the square of the euclidean distance between* P *and* O*. The* difference in norms *between two points* P *and* Q *in* $\mathcal{D}$ *with respect to* O*, denoted* NormDiff(P,Q,O)*, is given by*
   $\mathrm{NormDiff}(P,Q,O) = |\mathrm{Norm}(P,O) - \mathrm{Norm}(Q,O)|$.

The following proposition gives a characterization of the graphical path length of a point P, in the case where $\mathcal{D}$ is minimally generated, with respect to a point O, in $\mathcal{D}$, in terms of the basic directions, $\mathrm{Dirs}(\mathcal{D})$, of $\mathcal{D}$.

Proposition 4.1.1 *Suppose $\mathcal{D}$ is minimally generated, and suppose P and O are points in $\mathcal{D}$. Then,*

$$\begin{aligned}\mathrm{GraphPL}(P,O) &= \min\{\sum_{i=0}^{l}\alpha_i \mid \forall i, 0 \le i \le l, \alpha_i \in \mathbf{Z} \textit{ and} \\ &\qquad \sum_{i=0}^{l}\alpha_i \cdot D_i = (P-O), \forall i, 0 \le i \le l, D_i \in \mathrm{Dirs}(\mathcal{D})\}.\end{aligned}$$

Proof. This is clear geometrically and can be proved by a straightforward induction on $\mathrm{GraphPL}(P,O)$. □

4.2 Voronoi Cells and Well-Represented Lattices

Since relevant vectors and, therefore Voronoi cells around points, play an important role in well-represented lattices, in this section we state and prove some technical facts, which we will use later, regarding Voronoi cells in well-represented lattices.

Lemma 4.2.1

$$x \in \overline{\mathrm{Voronoi}(O,\mathcal{L})} \Longleftrightarrow \forall D \in \mathrm{Dirs}(\mathcal{L}), |(x,D)| \le \mathrm{MinLgth}(\mathcal{L})^2/2.$$

Proof. Immediate from requirement 3 (see Definition 4.1.4) of well-represented lattices, i.e., $\mathrm{Dirs}(\mathcal{L}) = \mathrm{Rel}(\mathcal{L})$. □

We will use the following obvious fact in the proof of Proposition 4.2.1 below.

Lemma 4.2.2 *Suppose A, B are real numbers. Then,*

$$(A \le B \textit{ and } [A,B]\cap[0,1] \ne \emptyset) \Longleftrightarrow (\max\{0,A\} \le \min\{1,B\}).$$

□

Proposition 4.2.1 *Let L be a finite subset of $\mathcal{L}$. Let x_1 and x_2 be two points in the convex hull of L. Let*

$$\begin{aligned}\mathrm{Dirs}_0(x_1,x_2,\mathcal{L}) &= \{D \in \mathrm{Dirs}(\mathcal{L}) | ((x_2-x_1),D) = 0\} \textit{ and} \\ \mathrm{Dirs}_+(x_1,x_2,\mathcal{L}) &= \{D \in \mathrm{Dirs}(\mathcal{L}) | ((x_2-x_1),D) > 0\}.\end{aligned}$$

Then, for all points $P \in L$, there is a point in $([x_1, x_2) \cap \overline{\text{Voronoi}(P, \mathcal{L})})$ iff

$$\begin{aligned}
&\max\{0, \max\{\frac{(P - x_1, D) - (\text{MinLgth}(\mathcal{L})^2/2)}{((x_2 - x_1), D)} \mid D \in \text{Dirs}_+(x_1, x_2, \mathcal{L})\}\} \\
&\leq \min\{1, \min\{\frac{(P - x_1, D) + (\text{MinLgth}(\mathcal{L})^2/2)}{((x_2 - x_1), D)} \mid D \in \text{Dirs}_+(x_1, x_2, \mathcal{L})\}\} \\
&< 1, \qquad (17)
\end{aligned}$$

and

$$\begin{aligned}
&\max\{((P - x_1), D) - (\text{MinLgth}(\mathcal{L})^2/2) \mid D \in \text{Dirs}_0(x_1, x_2, \mathcal{L})\} \\
&\leq 0 \\
&\leq \min\{((P - x_1), D) + (\text{MinLgth}(\mathcal{L})^2/2) \mid D \in \text{Dirs}_0(x_1, x_2, \mathcal{L})\}. \qquad (18)
\end{aligned}$$

Proof. Using Lemma 4.2.1, the line segment from x_1 to x_2, both points inclusive, i.e. $[x_1, x_2]$, has a non-empty intersection with $\overline{\text{Voronoi}(P, \mathcal{L})}$, iff there is a real number $t \in [0, 1]$, so that, for each D in $\text{Dirs}(\mathcal{L})$,

$$|((t(x_2 - x_1) + x_1 - P), D)| \leq \text{MinLgth}(\mathcal{L})^2/2. \qquad (19)$$

Notice that the value of the left hand side of (19) remains unchanged if D is replaced by $(-D)$. Therefore, the line through x_1 and x_2 passes through $\overline{\text{Voronoi}(P, \mathcal{L})}$ iff there is a real number t so that, for each D in $(\text{Dirs}_0(x_1, x_2, \mathcal{L}) \cup \text{Dirs}_+(x_1, x_2, \mathcal{L}))$,

$$|((t(x_2 - x_1) + x_1 - P), D)| \leq \text{MinLgth}(\mathcal{L})^2/2.$$

If $D \in \text{Dirs}_0(x_1, x_2, \mathcal{L})$, then (19) reduces to

$$(-\text{MinLgth}(\mathcal{L})^2/2) - ((x_1 - P), D) \leq 0 \leq (\text{MinLgth}(\mathcal{L})^2/2) - ((x_1 - P), D). \qquad (20)$$

If $D \in \text{Dirs}_+(x_1, x_2, \mathcal{L})$, then (19) reduces to

$$\frac{(P - x_1, D) - (\text{MinLgth}(\mathcal{L})^2/2)}{((x_2 - x_1), D)} \leq t \leq \frac{(P - x_1, D) + (\text{MinLgth}(\mathcal{L})^2/2)}{((x_2 - x_1), D)}. \qquad (21)$$

Then, the line segment from x_1 to x_2, not including x_2, passes through $\overline{\text{Voronoi}(P, \mathcal{L})}$ iff

1. for each D in $\mathrm{Dirs}_0(x_1, x_2, \mathcal{L})$, (20) holds, and

2. there is a t in $[0,1)$, so that, for each D in $\mathrm{Dirs}_+(x_1, x_2, \mathcal{L})$, (21) holds.

Now this lemma follows from lemma 4.2.2. $\square$

Proposition 4.2.2 *Suppose P and Q in $\mathcal{L}$ such that $Q = P + D, D \in \mathrm{Dirs}(\mathcal{L})$. Suppose $x \in \overline{\mathrm{Voronoi}(Q, \mathcal{L})}$. Then, $((x - P), D) \geq \mathrm{MinLgth}(\mathcal{L})^2/2$.*

PROOF. By definition of Voronoi cells, we have

$$\begin{aligned} -\mathrm{MinLgth}(\mathcal{L})^2/2 &\leq ((x - Q), D) \\ &= ((x - (P + D)), D) \\ &= ((x - P), D) - (D, D) \\ &= ((x - P), D) - \mathrm{MinLgth}(\mathcal{L})^2. \end{aligned}$$

Therefore, $((x - P), D) \geq \mathrm{MinLgth}(\mathcal{L})^2/2$. $\square$

Lemma 4.2.3 *Let L be a finite subset of $\mathcal{L}$. Suppose a and b are two points in the convex hull of L, so that $a \in \overline{\mathrm{Voronoi}(P, \mathcal{L})}$ and $b \in \overline{\mathrm{Voronoi}(Q, \mathcal{L})}$, $P, Q \in L$. Then, there exists a finite sequence of points $(P =)P_0, P_1, \ldots, P_k(= Q)$, each $P_i, 0 \leq i \leq k$ in $\mathcal{L}$, so that*

1. *$\forall i, 0 \leq i < k, P_i$ is a neighbor of P_{i+1},*

2. *$\forall i, 0 \leq i \leq k, ([a, b) \cap \overline{\mathrm{Voronoi}(P_i, \mathcal{L})}) \neq \emptyset$, and*

3. *$[a, b) \subset \overline{\cup_{i=0}^{k} \mathrm{Voronoi}(P_i, \mathcal{L})}$.*

PROOF. We prove this lemma by describing a process that explicitly produces a sequence of points $P_0, P_1, \ldots, P_k$ satisfying the above three conditions.

let $x_0 = a$;
let $P_0 = P$;
$i := 0$;

while$(x_i \neq b)$ **do**

let $x_{i+1} = \max\{x' \in [x_i, b) | [x_i, x') \in \overline{\text{Voronoi}(P_i, \mathcal{L})}\}$;

if$(x_{i+1} \neq b)$ **then**

let F_{i+1} be a face of Voronoi$(P_i, \mathcal{L})$ such that $x_{i+1} \in F_{i+1}$;

let $P_{i+1} = P_i + D_{F_{i+1}}$,

where $D_{F_{i+1}}$ is the basic direction perpendicular to F_{i+1}

endif;

$i := i + 1$;

endwhile;

Then, the following facts, together, prove that the above process does produce the desired finite sequence of points. The iteration of the while loop when the value of i is k will be called the k^{th} iteration.

1. $P = P_0$ is the first point produced by this process.
2. The process enters the i^{th} iteration of the while loop iff $x_i \neq b$. Then, there is at least one Voronoi cell, other than the one around P_i which contains part of $[x_i, b)$. Therefore, x_{i+1} exists.
3. $x_{i+1} = b$ or $x_{i+1} \neq b$. If $x_{i+1} = b$ then, $P_i = Q$.
4. If $x_{i+1} \neq b$, then x_{i+1} must lie on a face of Voronoi$(P_i, \mathcal{L})$, since there is a point in (x_{i+1}, b) not in Voronoi$(P_{i+1}, \mathcal{L})$. Therefore, F_{i+1} exists.
5. In well-represented lattices, every face on a Voronoi cell is perpendicular to a basic direction.
6. If $x_{i+1} \neq b$ then, for each F, of Voronoi$(P, \mathcal{L})$, that has a non-empty intersection with (x_{i+1}, b), $x_{i+1} \in \overline{\text{Voronoi}((P_{i+1} + D_F), \mathcal{L})}$.
7. Each point, $P_i, i > 0$, that is produced by the process is a neighbor of the last point, P_{i-1}, that was produced by the process.
8. By lemma 3.4.11 the number of Voronoi cells that contain a part of $[a, b)$ is finite, so the while loop must terminate. Therefore, the last point produced, P_k, must be Q.

9. If there is a point P so that $[a, b) \subset \overline{\text{Voronoi}(P, \mathcal{L})}$, then the process produces $P_0 = P$ as the finite sequence of points, and part 3 of the lemma is thus true.

10. If $[a, b)$ is contained in a collection, of size greater than one, of Voronoi cells, then by the choice of x_{i+1} in each iteration, the finite sequence of points produced by the process satisfies part 3 of the lemma.

□

4.3 Suitable Meshes

In Section 3.2.2, we described the construction of idealized meshes based on lattices. Since the classes of minimally generated lattices, integrally-representable minimally generated lattices, and well-represented lattices are special classes of lattices, the same construction, for a lattice from any one of these classes yields an idealized mesh based on that lattice. Note that each lattice in any one of these three classes, by definition, can be generated from its minimal length vectors. Thus by Theorem 3.2.5, idealized meshes based on any of these lattices are connected. Of course, in practice, we would be forced to use a mesh connected computer that has a finite number of processors.

We first define some useful concepts associated with a finite subset, D, of $\mathcal{D}$. In Section 4.3.1, we specify certain assumptions on D. It is *necessary* that D satisfy these assumptions for any of our algorithms to work in $\mathcal{M}(D)$. Algorithm CURVE, and hence specific instances of it, assume additional constraints on the lattice that D is a subset of. Then, in Section 4.3.2, we specify, in general, the assumptions we make, in the design of our algorithms, about the hardware of the processors in a mesh.

Definition 4.3.1 *Suppose $\mathcal{D}$ is minimally generated. Suppose D is a finite, non-empty subset of $\mathcal{D}$.*

1. *Suppose O is a point in D. Then D is called O–*convex *in $\mathcal{D}$ iff $\forall P \in D, Inner_{\mathcal{D}}(P, O)$ is a subset of D.*

2. *The* diameter of D, *denoted* $\mathrm{Diam}(D)$, *is the maximum, over pairs,* (P, Q), *of points in* D, *of* $\lceil d(P, Q) \rceil$. *(N.B.* $\mathrm{Diam}(D)$ *is always an integer.)*

3. $\mathrm{MinEuclDiff}(r, D)$, *is the minimum, over triples,* (O, P, Q), *of points in* D, *of* $|d(P, O) - d(Q, O)|$, *where each of* $d(P, O)$ *and* $d(Q, O)$ *is at most* r, *and* $d(P, O) \neq d(Q, O)$. *We will denote* $\mathrm{MinEuclDiff}(Diam(D), D)$ *by* $\mathrm{MinEuclDiff}(D)$.

4.3.1 Assumption on the Underlying Set, D, of a Mesh $\mathcal{M}(D)$

We first specify the three-part assumption on D, and then briefly justify each of the three parts. Finally, in this section, we note, in Proposition 4.3.1, that there is an abundance of underlying sets D satisfying Assumption 4.3.1.

Assumption 4.3.1 *There is an integrally-representable, minimally generated lattice* $\mathcal{D}'$ *so that*

1. D *is a finite subset of* $\mathcal{D}'$,

2. $\mathrm{MinEuclDiff}(D)$ *is defined, as witnessed by three non-collinear points in* D, *and,*

3. *for each* O *in* D, D *is* O*–convex in* $\mathcal{D}'$.

Finiteness is clearly necessary if one were to build a mesh based on D in practice. Having the lattice $\mathcal{D}'$, of which D is a finite subset, be an integrally-representable, minimally generated lattice guarantees that the corresponding idealized mesh is connected (Theorem 3.2.5), and that every point in $\mathcal{D}'$ *can* be represented using integers. As seen earlier in Section 4.1, having integer representations for points in $\mathcal{D}'$ allows for clean computations involving integers only.

Part 2 of Assumption 4.3.1 ensures a non-trivial mesh by essentially excluding one dimensional meshes, and meshes that are complete graphs. The latter prevents $\mathrm{MinEuclDiff}(D)$ and $\mathrm{MinArcLgth}(K)$ (Definition 5.3.1 below) from being undefined (or infinite).

The relevance of part 3 of Assumption 4.3.1 is captured in the following theorem.

Theorem 4.3.1 *Suppose D satisfies part 2 of Assumption 4.3.1. Then, for any two points in D, every shortest path, in $\mathcal{D}'$, between the two points is contained in D.*

PROOF. Suppose by way of contradiction otherwise. Suppose P and Q are two points in D, $(P =)P_0, P_1, \ldots, P_m(= Q)$ is a shortest path between P and Q, and, suppose, without loss of generality, there is an $i, 0 < i \leq m$ so that $P_i \notin D$ and $P_{i+1} \in D$. Then, by definition of shortest path between P and Q, $P_i \in Inner_{\mathcal{D}'}(P_{i+1}, P)$. Since $P \in D$, and by the hypothesis of this theorem, D is P–convex in $\mathcal{D}'$. Therefore, since $P_{i+1} \in D$, $Inner_{\mathcal{D}'}(P_{i+1}, P) \subset D$, giving us a contradiction. □

The above assumption on D determines the *structure* of the resulting mesh, $\mathcal{M}(D)$. Notice that under this assumption, by Theorem 4.3.1 above, $\mathcal{M}(D)$ is connected, i.e., one can get from any processor in $\mathcal{M}(D)$ to any other processor in $\mathcal{M}(D)$ using a sequence of immediate connections. If $\mathcal{M}(D)$ is not connected in this sense, then the mesh would have at least two disjoint components.

Proposition 4.3.1 *For each $n > 1$ dimensional, integrally-representable, minimally generated lattice $\mathcal{D}'$, there are an infinite number of subsets D of $\mathcal{D}'$ satisfying Assumption 4.3.1.*

PROOF. There is a basis, $\mathcal{B} = \{b_1, \ldots, b_n\}$, for $\mathcal{D}'$ where, for each $i, 1 \leq i \leq n, b_i$ is a minimal length vector in $\mathcal{D}'$. For each integer $k > 0$ define D_k to be

$$D_k = \{\ \alpha_1 \cdot b_1 + \cdots + \alpha_n \cdot b_n \mid (\forall i, 1 \leq i \leq n)\ \alpha_i \in \mathbf{Z} \text{ and } 0 \leq \alpha_i \leq k\ \}.$$

Each D_k is shaped like an n-dimensional parallelepiped, with a vertex at the origin of $\mathcal{D}'$. The vectors of $\mathcal{B}$ define the 1-dimensional edges of these parallelepipeds. □

4.3.2 Assumptions on the Hardware of $\mathcal{M}(D)$

Having decided on the structure of D, we now consider the abilities of the individual elements of $\mathcal{M}(D)$, i.e., the processors and connections in $\mathcal{M}(D)$. For P in D, we will use the notation $\mathcal{M}(P)$ to denote $\mathcal{M}(\{P\})$. The individual processors in $\mathcal{M}(D)$ could have usual Von-Neumann architectures [Man76], but with certain special abilities. Below, we list the general assumptions about the idealized hardware of $\mathcal{M}(D)$.

Assumption 4.3.2 *All connections in $\mathcal{M}(D)$ are identical in their abilities to carry data between processors. We assume that all connections are capable of carrying the same amount of data in the same amount of time.*

Assumption 4.3.3 *There is a common, finite set of basic instructions that every processor in $\mathcal{M}(D)$ can execute. These are the only basic instructions that any processor in $\mathcal{M}(D)$ can execute. It suffices to say, here, that the set of basic instructions must be sufficient to encode, at least, the specific applications of Algorithm Template EuclideanMotion described in Sections 7.1, 7.2, 9.2, and 9.3 below. For example, for receiving and sending immediate interprocessor information it would be helpful to have a system of in and out buffers in each processor.*

Assumption 4.3.4 *The actual time to execute the same basic instruction by any processor in $\mathcal{M}(D)$ is a constant. By* instruction cycle of a processor *we mean the minimum possible time interval within which any single instruction can be fully executed by any processor. Hence, in each processor, each basic instruction takes the same instruction cycle amount of time to execute.*

Assumption 4.3.5 `SimClock` *is the instruction counter for each $\mathcal{M}(P)$. The semantics of the instruction*

$$\textbf{idle till } \texttt{SimClock} = X \tag{22}$$

dictates that, it iterates passes of itself until, if ever, `SimClock`$= X$ *is true; if* `SimClock`$= X$ *is true, then the next instruction after (22) is instruction number $X + 1$. We assume that each pass of the instance of the* **idle till** *in step 3 of EuclideanMotion (Section 6.4.4) takes exactly one instruction cycle.*[1]

Assumption 4.3.6 *All processors in $\mathcal{M}(D)$ execute basic instructions synchronously, i.e., instruction cycles of all processors share a same, actual time interval. This discretizes time, for the mesh as a whole, into intervals of equal length.*

[1] For many hardware designs, for example, that of [CRS90b], it suffices to implement (22) with a suitably chosen number, IT, of instruction cycles per pass. Such implementations are not, in general, particularly straightforward, so for simplicity here, we have assumed a hardware for which IT is 1.

Assumption 4.3.7 *In* one *instruction cycle, each processor,* $\mathcal{M}(P)$ *in* $\mathcal{M}(D)$*, can* simultaneously *send messages to any subset of the processors in* $\mathcal{M}(\mathrm{Nbrs}(P) \cap D)$*, one message to each element of the subset, possibly different messages to different elements of the subset.*

Chapter 5

Resolution in Suitable Meshes

Convention 5.0.1 *Henceforth,* $K \subseteq \mathcal{K}$ *and* $\mathcal{M}(K)$ *is a suitable mesh as witnessed by* $\mathcal{K}$.[1]

Since $\mathcal{K}$ witnesses that K satisfies Assumption 4.3.1, it can be shown that[2]

Proposition 5.0.2 $\mathrm{MinLgth}(K) \geq 1.0$*, and* $\mathrm{Diam}(K)$ *(is an integer)* ≥ 2. $\square$

Suppose we represent a bounded region of euclidean n-space by $\mathcal{M}(K)$. Then, $\mathcal{K}$ and the subset, K, of $\mathcal{K}$ determine the resolution of this representation. In our methodology we try to obtain simulations that run in time linear in the real–time that the phenomenon being simulated takes to unfold. As we shall see, this will interestingly force the maximum speed of simulations to depend, in a complex way, on the choice of K.

We broadly discuss the role that the choice of $\mathcal{M}(K)$ plays in deciding the units of space and time that can be represented (in mesh computer simulations with our methodology). For the sake of all of the analyses in this chapter, we "factor out" the time required for execution of programs built out of basic instructions.

In Section 5.1, we, in effect, discuss the simulation of the phenomenon of radiation from a point source. We first show that an exact simulation, i.e., one which

[1] $K \subseteq \mathcal{K}$ satisfies Assumption 4.3.1 as witnessed by $\mathcal{K}$ and $\mathcal{M}(K)$ satisfies Assumptions 4.3.2–4.3.7.

[2] Re MinLgth see Definition 3.2.2, and re Diam($\cdot$) see Definition 4.3.1.

runs *exactly* proportional to the real–time of the phenomenon of (constant speed) radiation, is not possible in $\mathcal{M}(K)$, and we indicate (Theorem 5.1.2) the sense in which our algorithms WAVE and WAVE$'$ provide wave front generation *virtually* in time proportional to the real–time that the phenomenon of such radiation takes to unfold. In Section 5.2, we "derive" a a reasonable *estimate*, MaxLinSpeed(K), on an upper bound on *mesh* speed of radiation and straight-line motion.

In Section 5.3, we discuss the simulation of the revolution of a particle, in a given plane, about a given point. We motivate a particular general *estimate*, MaxAngle(K), of an upper bound on the maximum angle that can be traversed, in $\mathcal{M}(K)$, in one unit of simulation time, at the furthest possible distance in $\mathcal{M}(K)$ from the point about which the revolution is taking place. Additionally, since we would like to perform all computations using integers only, we would like cos and sin of angles with which we work to be integers or rationals. In Section 5.3, we "derive" the *rational-valued* cos and sin for an angle, MaxTheta(K), that is "close enough" to MaxAngle(K), and we use MaxTheta(K) in place of MaxAngle(K).

The unit of simulation time is the real–time required to complete one instruction cycle. *Simulation time*, in a simulation, is, then, measured in the number of such units elasped since the beginning of the simulation.

One of the absolute limitations of any mesh, satisfying Assumption 4.3.7 on the hardware of the mesh (see Section 4.3.2), is that the fastest that a message can travel between adjacent processors is in 1 simulation time unit, i.e., 1 instruction cycle. Hence, the fastest that a message could be communicated from one processor $\mathcal{M}(P)$, in $\mathcal{M}(K)$, to another processor $\mathcal{M}(Q)$, also in $\mathcal{M}(K)$, is in GraphPL(P, Q) instruction cycles. Now, suppose processor $\mathcal{M}(O)$, in $\mathcal{M}(K)$, could send messages to *two* distinct processors, $\mathcal{M}(P)$ and $\mathcal{M}(Q)$, in $\mathcal{M}(K)$, at this fastest possible speed of communication in the mesh. Suppose $t(P)$ and $t(Q)$ are the respective simulation times at which $\mathcal{M}(P)$ and $\mathcal{M}(Q)$ get their messages. Then, the ratios $d(P, O)/t(P)$ and $d(Q, O)/t(Q)$ *need* not be equal. This consideration naturally presents a problem if one has to correctly simulate uniform radiation from a point source. In fact, in the simulation of the phenomenon of radiation, a message must propagate radially outward from a processor, $\mathcal{M}(O)$, in the mesh, so that the message (relayed by intermediate processors) reaches *every* other processor, $\mathcal{M}(P)$, in the mesh in simulation time (virtually) proportional to the *euclidean* distance between O and P. We will think of the simulation of uniform radiation from a point source, as broadcasting a message from a processor in a mesh to all other processors in the mesh.

5.1 Wavefront Broadcast

We first set up some notation.

Notation 5.1.1

1. *$\mathcal{M}(K')$ ranges over suitable meshes based on $\mathcal{K}$.*

2. *$\mathcal{A}$ will range over algorithms to be executed on $\mathcal{M}(K')$. $\mathcal{A}$ is encoded in the common, finite set of basic instructions that every processor in $\mathcal{M}(K')$ can execute.*

3. *In a simulation, for a processor $\mathcal{M}(P)$, in $\mathcal{M}(K')$,* we say $\mathcal{M}(P)$ sends a message at simulation time t *iff (by definition) $\mathcal{M}(P)$ executes a basic instruction to send a message just after $(t-1)$ instruction cycles have elapsed since the beginning of the simulation.*

4. *In a simulation, for a processor $\mathcal{M}(P)$, in $\mathcal{M}(K')$,* we say $\mathcal{M}(P)$ receives a message at simulation time t *iff (by definition) there is a neighbor, $\mathcal{M}(Q)$, of $\mathcal{M}(P)$ in $\mathcal{M}(K')$ so that $\mathcal{M}(Q)$ sends a message to $\mathcal{M}(P)$ at simulation time $(t-1)$.*

Consider the broadcasting of a message from a processor in $\mathcal{M}(K')$ to all the other processors in $\mathcal{M}(K')$. We will first define (Definition 5.1.1) what we mean by an algorithm so broadcasting.

Definition 5.1.1 $\mathcal{A}$ broadcasts from a processor $\mathcal{M}(O)$ in $\mathcal{M}(K')$ *iff*

1. *$\mathcal{A}$ causes $\mathcal{M}(O)$ to send a message, and*
2. *each processor in $\mathcal{M}(K')$, other than $\mathcal{M}(O)$, eventually receives the message from $\mathcal{M}(O)$.*

For a processor $\mathcal{M}(P)$ in $\mathcal{M}(K')$, $t_{\mathcal{A}}(P)$ will denote the smallest simulation time at which $\mathcal{M}(P)$ receives this message, where $t_{\mathcal{A}}(O) = 0$.

There are several algorithms that broadcast as defined above. For our methodology, we would like broadcasts to be timed so that processors receive messages in simulation time proportional to their *euclidean distance* from the source of the broadcast. We can think of such a broadcast as a wavefront.

In Definition 5.1.2 we say what we mean by an algorithm for *perfectly*-timed wavefront broadcast. Then, we state and prove, in Theorem 5.1.1, the non-existence of such an algorithm.

Definition 5.1.2 $\mathcal{A}$ *is a* perfectly-timed wavefront broadcast from a processor $\mathcal{M}(O)$ in $\mathcal{M}(K')$ *iff*

1. *$\mathcal{A}$ broadcasts from $\mathcal{M}(O)$ in $\mathcal{M}(K')$, and,*
2. *there is a positive real number c such that, for each processor $\mathcal{M}(P)$, other than $\mathcal{M}(O)$, in $\mathcal{M}(K')$,*
$$\frac{d(P,O)}{t_{\mathcal{A}}(P)} = c.$$

Theorem 5.1.1 *There are K' and a point O in K' so that no algorithm is a perfectly-timed wavefront broadcast from $\mathcal{M}(O)$ in $\mathcal{M}(K')$.*

Proof.

Consider the parallelepiped subsets D_k of $\mathcal{D}$ introduced in the proof of Proposition 4.3.1. Analogously define such parallelepiped subsets K_k of $\mathcal{K}$. As seen in the proof of Proposition 4.3.1, each of these parallelepiped subsets satisfies Assumption 4.3.1. By the proof of Corollary 3.3.1, there is an integer $k_0 > 0$ for which, there are points O, P and Q in K_{k_0} so that $d(P,O)/d(Q,O)$ is an irrational number. Suppose, by way of contradiction, $\mathcal{A}$ is a perfectly-timed wavefront broadcast from $\mathcal{M}(O)$ in $\mathcal{M}(K_{l_0})$. Then, from Definition 5.1.2,

$$\frac{d(P,O)}{d(Q,O)} = \frac{t_{\mathcal{A}}(P)}{t_{\mathcal{A}}(Q)}. \tag{23}$$

The right hand side of Equation (23) is a rational number giving us a contradiction. □

It is interesting to note the following proposition which can be proved by simply generalizing the standard proof that $\sqrt{2}$ is irrational.

Proposition 5.1.1 *Any euclidean distance in $\mathcal{K}$ is either an integer or an irrational; hence, any such distance is either an integer multiple or an irrational multiple of* MinLgth($\mathcal{K}$). □

In Sections 7.1 and 7.2 we present two algorithms WAVE and WAVE$'$, each of which provide *virtually* perfectly-timed wavefront broadcast in the sense given by Theorem 5.1.2 below.

Actually it is easiest here to think of WAVE and WAVE$'$ as algorithm schema which become actual algorithms when simple changes are made to take into account dependence on choices of O and K'. In turns out, besides the simple dependence on the dimension n, that the dependence of WAVE and WAVE$'$ on the choice of O and K' involves, in effect, making $\mathcal{M}(O)$ think it received the broadcast message at time $t = 0$ and inserting the constant 2Diam(K') in an appropriate slot. Hence, for convenience, in Theorem 5.1.2 below, we speak of algorithm $\mathcal{A} \in \{\text{WAVE}, \text{WAVE}'\}$ rather than $\mathcal{A} \in$ some *very* simple variant of $\{\text{WAVE}, \text{WAVE}'\}$.

Theorem 5.1.2 *Suppose $\mathcal{M}(O) \in \mathcal{M}(K')$. Then, for each $\mathcal{A} \in \{\text{WAVE}, \text{WAVE}'\}$,*

1. *$\mathcal{A}$ broadcasts from $\mathcal{M}(O)$ in $\mathcal{M}(K')$,*

2. *for processors $\mathcal{M}(P)$ and $\mathcal{M}(Q)$ in $\mathcal{M}(K')$,*

$$d(P,O) < d(Q,O) \Longleftrightarrow t_{\mathcal{A}}(P) < t_{\mathcal{A}}(Q),$$

and

3. *there are positive real numbers c and ϵ such that, for each processor $\mathcal{M}(P)$, other than $\mathcal{M}(O)$, in $\mathcal{M}(K')$,*

   (a) *if $d(P,O)$ is an integer*[3], *then*

   $$\frac{d(P,O)}{t_{\mathcal{A}}(P)} = c; \text{ otherwise,}$$

   (b)

   $$|\frac{d(P,O)}{t_{\mathcal{A}}(P)} - c| < \epsilon; \text{ furthermore,}$$

   (c)

   $$\frac{\epsilon}{c} < \frac{1}{(2 \cdot \mathrm{Diam}(K'))^2}.$$

5.2 Linear Resolution and Wavefront Speed

We derive a simple expression *estimating* an upper bound on the maximum speed with which a wave can be propagated in a mesh. Even the estimated upper bound to be derived may not be reached in certain subsets of certain integrally-representable, minimally generated lattices. Yet, the expression for this estimated upper bound has the advantage that it is simple and does not depend on which specific $\mathcal{K}$ the suitable mesh, $\mathcal{M}(K)$, is based on.

Suppose $\mathcal{A}$ is a virtually perfectly-timed wavefront broadcast, for *example*, as in Theorem 5.1.2. Suppose a subset K of $\mathcal{K}$ satisfies Assumption 4.3.1, and suppose P

[3] *If we had chosen* Diam*(K') from Definition 4.3.1 to instead be the maximum, over pairs, (P,Q), of points in D, of* $\lceil \mathrm{MinLgth}(\mathcal{K}) \cdot d(P,Q) \rceil$*, then 'integer' here would be replaced by 'integer multiple of* MinLgth*(K)', and, as we shall see, our simulations (with appropriate changes to the algorithms) would be slowed down a bit more.*

and Q and O are distinct points in K so that MinEuclDiff(K)) $= d(Q,O) - d(P,O)$.[4] Clearly, $\mathcal{M}(Q)$ should get a message at least one simulation time unit later than $\mathcal{M}(P)$. Hence, a difficult to calculate, actual upper bound on the maximum speed of uniform wave propagation in $\mathcal{M}(K)$ is MinEuclDiff(K) distance per instruction cycle.

Recall that, K is finite, and the square of the euclidean distance of any point, in K, from O is an integer.

Theorem 5.2.1 $\forall r \in \mathbf{R}$ *such that* MinEuclDiff(r, K) *is defined,*

$$\mathrm{MinEuclDiff}(r,K) > \frac{1}{2 \cdot r}.$$

Proof. Let X and Y be elements of $\mathcal{K}$, so that MinEuclDiff(r, K) $= \|X\| - \|Y\|$. Then, since $\|X\| \leq r$, and $\|X\|^2$ and $\|Y\|^2$ are integers,

$$\begin{aligned} \mathrm{MinEuclDiff}(r,K) &\geq \|X\| - \sqrt{\|X\|^2 - 1} \\ &= \frac{1}{\|X\| + \sqrt{\|X\|^2 - 1}} \\ &\geq \frac{1}{r + \sqrt{r^2 - 1}} > \frac{1}{2r}. \end{aligned}$$

□

Corollary 5.2.1

$$\mathrm{MinEuclDiff}(K) = \mathrm{MinEuclDiff}(\mathrm{Diam}(K), K) > \frac{1}{2 \cdot \mathrm{Diam}(K)}.$$

□

In the design of actual algorithms to achieve the desired wave propagation, it turns out to be quite useful to have available the simple estimated bound of $(1/2\mathrm{Diam}(K))$ for a speed limit. Of course this is easy to calculate and depends only on K not on $\mathcal{K}$. In many cases of interest it is no more than $(1/8\mathrm{Diam}(K)^3)$ from MinEuclDiff(K).

[4] Re MinEuclDiff($\cdot$) see Definition 4.3.1.

We note that in general, one cannot bound MinEuclDiff(K) below for all subsets K of a well-represented lattice by a positive constant, since in $\mathbf{Z}^2$, the difference between the lengths of the vectors $n \cdot e_1 + e_2$ and $n \cdot e_1$ tends to zero as n tends to infinity.

$$\|n \cdot e_1 + e_2\| - \|n \cdot e_1\| = \sqrt{n^2+1} - \sqrt{n^2} = \frac{1}{\sqrt{n^2+1}+n},$$

and

$$\lim_{n\to\infty} \frac{1}{\sqrt{n^2+1}+n} = 0 = \mathrm{MinEuclDiff}(\mathbf{Z}^2).$$

Definition 5.2.1 *We define* MaxLinSpeed(K) *to be* $(1/2\mathrm{Diam}(K))$.

We shall see that, for each of our algorithms $\mathcal{A}$, there is an integer $\alpha > 0$ for which (MaxLinSpeed(K)$/\alpha$) is the virtual speed actually attained by $\mathcal{A}$.

5.3 Angular Resolution

Just as in the previous section (Section 5.2) we performed some analysis to get a reasonable, easy to compute bound on linear resolution, so we proceed in this section for *angular* resolution.

Definition 5.3.1 *Suppose $O \in K$.*

1. MinAngleDiff(O, K) *is defined to be the minimum angle subtended by any two non-collinear lines each of which passes through the point, O, and another point in K.*

2. *The* angular resolution *in $\mathcal{M}(K)$, denoted* MinAngleDiff(K)*, is the minimum, over all points O in K, of* MinAngleDiff(O, K)*.*

3. *Suppose P and Q are two points, so that O, P, Q are not collinear, and so that $d(P, O) \leq d(Q, O)$. Let P' be the euclidean point on the line segment from O to Q so that $d(P, O) = d(P', O)$.* MinArcLgth(K) *is, then, defined to be the minimum, over all such triples of points (O, P, Q), of the length of the arc, on the circle of radius $d(P, O)$, between P and P' which arc subtends an angle* MinAngleDiff(K) *at O.*

Definition 5.3.2 *Suppose O is in $\mathcal{K}$, and P and Q are distinct points, other than O, in $\mathcal{K}$. Then, the* perpendicular projection *of the point P on the line defined by Q and O is the point $\alpha \cdot (Q - O), \alpha \in \mathbf{R}$, such that $(\alpha \cdot (Q - O), ((P - O) - \alpha \cdot (Q - O)))) = 0$.*

Notation 5.3.1 *For points O, P, and Q in $\mathcal{K}$ which are not collinear,* $\mathrm{PerpDist}(P, O \rightarrow Q)$ *denotes the euclidean distance between P and the perpendicular projection of P on the line defined by O and Q.*

Theorem 5.3.1

$$\mathrm{MinArcLgth}(K) > \frac{1}{\mathrm{Diam}(K)} = 2\mathrm{MaxLinSpeed}(K).$$

PROOF. Suppose the three non-collinear points O, P and Q, all in K, are such that the angle subtended by P and Q at O is $\mathrm{MinAngleDiff}(K)$. Suppose, without loss of generality, that O is the origin, and that $d(P, O) \leq d(Q, O)$. Then,

$$\begin{aligned} 0 < (\mathrm{PerpDist}(P, O \rightarrow Q))^2 &= \left(\left(P - \frac{(P,Q)}{(Q,Q)} \cdot Q\right), \left(P - \frac{(P,Q)}{(Q,Q)} \cdot Q\right)\right) \\ &= (P,P) - \frac{(P,Q)^2}{(Q,Q)} \\ &= \frac{((P,P)(Q,Q) - (P,Q)^2)}{(Q,Q)} \end{aligned} \tag{24}$$

The numerator in (24) is a (positive) integer since $\mathcal{K}$ integrally-represents itself. Also, since both O and Q are in K, $(Q,Q) = (d(O,Q))^2 \leq \mathrm{Diam}(K)^2$. Therefore,

$$\mathrm{MinArcLgth}(K) > \mathrm{PerpDist}(P, O \rightarrow Q) \geq \frac{1}{\mathrm{Diam}(K)} = 2\mathrm{MaxLinSpeed}(K).$$

□

Hence, the estimate $1/\mathrm{Diam}(K)$ is twice the size of our estimate from the previous section for *linear* resolution. It is useful to choose the smaller of the two for each resolution bound (linear and along arcs) for consistency between line travel and arc travel. Consider a revolution of a point along an arc of this length of $1/(2\mathrm{Diam}(K))$ at a distance $\mathrm{Diam}(K)$ from the point about which the revolution is taking place. Then the *angle* through which the revolution takes place is $1/(2\mathrm{Diam}(K)^2)$ radians.

Definition 5.3.3 *We define* $\mathrm{MaxAngle}(K)$ *to be* $1/(2\mathrm{Diam}(K)^2)$ *radians.*

Inner products are ubiquitous in our calculations. It would, then, be useful if the cosine of $\mathrm{MaxAngle}(K)$ were a rational. It may not be. So, we will choose $\mathrm{MaxTheta}(K)$ to be an angle, "close enough", but less than or equal, to $\mathrm{MaxAngle}(K)$, and so that both its cosine and sine are *rational.*

A reasonable criterion of "close enough" is as follows. Let

$$\rho = \frac{2\pi}{\mathrm{MaxAngle}(K)},$$

i.e., ρ is the (real) number of parts that the angle, equal to $\mathrm{MaxAngle}(K)$ in magnitude, divides a circle into. Then, we would like $\mathrm{MaxTheta}(K)$ to divide the circle into approximately ρ parts. We say that $\mathrm{MaxTheta}(K)$ is *close enough* to $\mathrm{MaxAngle}(K)$ iff the error in the approximation is small enough so that it is "not measurable in $\mathcal{M}(K)$" in the following sense. Suppose, on a circle of radius $\mathrm{Diam}(K)$, a is the length of the arc that subtends an angle $\mathrm{MaxTheta}(K)$ at the center of the circle. Then, the difference between the circumference, $2\pi\mathrm{Diam}(K)$, of the circle and the arclength, $\rho \cdot a$, on the circle, subtending an angle $\rho \cdot \mathrm{MaxTheta}(K)$ at the center of the circle, should be non-negative, but *strictly* less than the linear resolution, $(1/(2\mathrm{Diam}(K)))$, of $\mathcal{M}(K)$. We now show that it is possible to obtain such an angle, $\mathrm{MaxTheta}(K)$.

Ideally, we would like $\mathrm{MaxTheta}(K)$ to be a (small, positive) rational multiple of π, thus ensuring that, if we rotate a point in a plane in angular steps of $\mathrm{MaxTheta}(K)$, then after some number of complete revolutions, the point would be back exactly at its starting position. By the following result, essentially from [Niv56, Page 41], we cannot have arbitrarily small, positive, rational multiples of π with rational sine or cosine.

Theorem 5.3.2 *If* θ *is rational in degrees, say* $\theta = \pi r$ *for some rational number* r*, then the only rational values of the trigonometric functions of* θ *are as follows:* $\sin\theta, \cos\theta = 0, \pm\frac{1}{2}, \pm 1$*;* $\sec\theta, \mathrm{cosec}\ \theta = \pm 1, \pm 2$*;* $\tan\theta, \cot\theta = 0, \pm 1$*.* □

We first prove the following technical lemma to help us guess a good $\mathrm{MaxTheta}(K)$ and then verify that it is close enough to $\mathrm{MaxAngle}(K)$.

Lemma 5.3.1 *Suppose* $0 < \alpha < 1$. *Then,* $0 < \alpha - \tan^{-1}(\alpha) < \frac{\alpha^3}{3}$.

PROOF.

$$\begin{aligned}
\tan^{-1}(\alpha) &= \int_0^\alpha \frac{1}{1+x^2}\mathrm{d}x \\
&= \int_0^\alpha \sum_{i=0}^\infty (-1)^i x^{2i}\ \mathrm{d}x \\
&= \sum_{i=0}^\infty (-1)^i \frac{\alpha^{2i+1}}{2i+1} \\
&= \alpha - \frac{\alpha^3}{3} + \frac{\alpha^5}{5} - \frac{\alpha^7}{7} + \cdots \\
&= (\alpha - \frac{\alpha^3}{3}) + (\frac{\alpha^5}{5} - \frac{\alpha^7}{7}) + (\frac{\alpha^9}{9} - \frac{\alpha^{11}}{11}) + \cdots \\
&= (\alpha - \frac{\alpha^3}{3}) + \sum_{i=1}^\infty (\frac{\alpha^{4i+1}}{4i+1} - \frac{\alpha^{4i+3}}{4i+3}).
\end{aligned}$$

Now then, since $0 < \alpha < 1$, each term in the summation is positive. Therefore,

$$(\tan^{-1}(\alpha) > \alpha - \frac{\alpha^3}{3});$$

hence,

$$(\alpha - \tan^{-1}(\alpha) < \frac{\alpha^3}{3}).$$

Furthermore, it is easy to show that $0 < \alpha - \tan^{-1}(\alpha)$. □

Definition 5.3.4 *We define* $\mathrm{MaxTheta}(K)$ *so that*

$$\mathrm{MaxTheta}(K)/2 = \tan^{-1}(\mathrm{MaxAngle}(K)/2).$$

Since $\mathrm{Diam}(K)$ is an integer (≥ 2, by Proposition 5.0.2), the following result shows how to explicitly obtain rational expressions for each of $\sin(\mathrm{MaxTheta}(K))$ and $\cos(\mathrm{MaxTheta}(K))$.

Theorem 5.3.3

$$\begin{aligned}
\sin(\mathrm{MaxTheta}(K)) &= \frac{8\mathrm{Diam}(K)^2}{16\mathrm{Diam}(K)^4+1} \quad \textit{and} \\
\cos(\mathrm{MaxTheta}(K)) &= \frac{16\mathrm{Diam}(K)^4-1}{16\mathrm{Diam}(K)^4+1}.
\end{aligned}$$

PROOF.

From Definition 5.3.4 just above,

$$\begin{aligned}
\mathrm{MaxTheta}(K)/2 &= \tan^{-1}\left(\frac{1}{4\mathrm{Diam}(K)^2}\right), \textit{and} \\
\sin(\mathrm{MaxTheta}(K)/2) &= \frac{1}{\sqrt{16\mathrm{Diam}(K)^4+1}} \quad \text{and} \\
\cos(\mathrm{MaxTheta}(K)/2) &= \frac{4\mathrm{Diam}(K)^2}{\sqrt{16\mathrm{Diam}(K)^4+1}}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\sin(\mathrm{MaxTheta}(K)) &= 2\sin(\mathrm{MaxTheta}(K)/2)\cdot\cos(\mathrm{MaxTheta}(K)/2) \\
&= \frac{8\mathrm{Diam}(K)^2}{16\mathrm{Diam}(K)^4+1} \quad \text{and} \\
\cos(\mathrm{MaxTheta}(K)) &= \cos^2(\mathrm{MaxTheta}(K)/2) - \sin^2(\mathrm{MaxTheta}(K)/2) \\
&= \frac{16\mathrm{Diam}(K)^4-1}{16\mathrm{Diam}(K)^4+1}.
\end{aligned}$$

□

Lemma 5.3.2

$$\mathrm{MaxTheta}(K) > \left(\mathrm{MaxAngle}(K) - \frac{1}{6\cdot 4^2\mathrm{Diam}(K)^6}\right).$$

PROOF. We let $\alpha = \mathrm{MaxAngle}(K)/2$ in Lemma 5.3.1 to obtain

$$\begin{aligned}
\mathrm{MaxAngle}(K) - \mathrm{MaxTheta}(K) &< 2\cdot\frac{(\mathrm{MaxAngle}(K)/2)^3}{3} \\
&= \frac{1}{6\cdot 4^2\mathrm{Diam}(K)^6}.
\end{aligned}$$

□

Lemma 5.3.3 *Recall that*

$$a = \mathrm{MaxTheta}(K)\cdot\mathrm{Diam}(K).$$

Then,

$$a > \left(\mathrm{MaxAngle}(K) - \frac{1}{6\cdot 4^2\mathrm{Diam}(K)^6}\right)\cdot\mathrm{Diam}(K).$$

PROOF. Follows immediately from Lemma 5.3.2. □

Theorem 5.3.4 MaxTheta(K) *is close enough to* MaxAngle(K).

PROOF. Clearly, $\rho = 4\pi \cdot \mathrm{Diam}(K)^2$. With the notation in Lemma 5.3.3, we need to show that $0 \leq (2\pi\mathrm{Diam}(K) - \rho \cdot a) < 1/2\mathrm{Diam}(K)$. By Lemma 5.3.1 above, we have that, in fact, $0 < (2\pi\mathrm{Diam}(K) - \rho \cdot a)$. By Lemma 5.3.3 above, we have[5]

$$\begin{aligned}
2\pi\mathrm{Diam}(K) &- a\rho \\
&< 2\pi\mathrm{Diam}(K) - 4\pi\mathrm{Diam}(K)^3 \cdot \left(\mathrm{MaxAngle}(K) - \frac{1}{6 \cdot 4^2\mathrm{Diam}(K)^6}\right) \\
&= \frac{4\pi\mathrm{Diam}(K)^3}{6 \cdot 4^2\mathrm{Diam}(K)^6} \\
&= \frac{\pi}{24\mathrm{Diam}(K)^3} \\
&\ll \frac{1}{2\mathrm{Diam}(K)}.
\end{aligned}$$

□

From the immediately above proof we have that

Corollary 5.3.1 *The* relative *error in using* $\rho \cdot \mathrm{MaxTheta}(K)$ *as an approximation to* 2π *is* strictly less than

$$\frac{1}{48\mathrm{Diam}(K)^4}.$$

□

A rough estimate, using the just previous corollary, regarding algorithm REVOLVE in Section 9.3, yields that, for mesh diameter 100, a particle needs to be left revolving for about 40 thousand revolutions for it to be off in angular position as much as 50 millionths of a radian, the value, in this case of MaxAngle(K). In Chapter 11 we mention the intriguing open problem of whether we can find a suitable variant of MaxTheta(K) so that in REVOLVE the error in angular position can be kept bounded.

[5]See Proposition 5.0.2 to justify the last step of the proof.

Chapter 6

The General Algorithm Template

In this chapter, we present a general template for all our algorithms for simulating phenomena. Each of our algorithms, presented in Chapters 7, 8, and 9 is an instance of this *template*, Algorithm EuclideanMotion, presented in this chapter, in Section 6.4. Although we will refer to EuclideanMotion as an "algorithm", it is understood that EuclideanMotion is a *template* for actual algorithms. In Section 6.1, we describe the global iterative structure of EuclideanMotion and point out the important fact that each iteration takes constant execution time.

In Section 6.2, we list specific assumptions for the exposition and validity of Algorithm EuclideanMotion. It is understood that changes in any of these conditions may necessitate corresponding, appropriate changes in Algorithm EuclideanMotion.

In Section 6.3, we discuss the universal constant software entities (constants and types) that Algorithm EuclideanMotion needs.[1] We assume that these entities are pre-defined in the mesh $\mathcal{M}(K)$. Later, in Chapters 7, 8 and 9, we specify the additional (possibly universal) constants and types necessary for the specific instances of Algorithm EuclideanMotion described in those chapters.

Section 6.4 presents Algorithm EuclideanMotion. We describe there, the other (not universal) constants, types, variables, and the executable body of Algorithm

[1]We think of such *universal* constant entities as having to do with the characteristics of the universe in which our algorithms reside, namely suitable meshes.

EuclideanMotion. Since Algorithm EuclideanMotion is a general template for algorithms simulating physical phenomena in $\mathcal{M}(K)$, the description of Algorithm EuclideanMotion is not specific to any particular phenomenon. Details are provided in Chapters 7, 8 and 9.

6.1 Constant Speed Passes in Our Algorithms

In Chapter 5, we saw that to broadcast so that processors get messages in time proportional to their distance from the source of the broadcast, it was reasonable to slow down by a factor $2\text{Diam}(K)$. This slow down is reasonable if each processor can react *instantaneously* to messages received in a particular instruction cycle. Since we intend simulating a variety of physical phenomena in our mesh, it is reasonable to assume that each processor spends *simulation time* to gather and interpret messages, to process the information received, and to send messages.

Each of our algorithms is executed by each processor in $\mathcal{M}(K)$ as a sequence of *passes*. Each pass has the following three-part outline.

1. Read messages, if any. If there are any messages, then initialize necessary parameters and variables.
2. Perform phenomenon-specific computations. Typically this involves determining the neighbors to whom messages need to be sent, computing what messages need to be sent to each such neighbor, and for each such neighbor, determining in *which* pass to send the message to that neighbor[2], and
3. Idle an appropriate time and then send the messages that are to be sent in this pass, to appropriate neighbors.

Definition 6.1.1 *Let $\mathcal{A}$ be one of our algorithms.* $\texttt{CompLen}(\mathcal{A}, p) \stackrel{\text{def}}{=}$ *the number of instruction cycles it takes to execute pass p of $\mathcal{A}$.*

[2]It is essential that the sending *can* be delayed.

Following is an extremely important fact about *our algorithms $\mathcal{A}$ essential* to our solutions to software synchronization problems. It is guaranteed, in part, by the judicious idling just before a send in each pass of $\mathcal{A}$.

Remark 6.1.1 FOR EACH OF OUR ALGORITHMS $\mathcal{A}$, THERE IS A CONSTANT SO THAT, *for each pass p of $\mathcal{A}$*, $\texttt{CompLen}(\mathcal{A}, p)$ IS *that constant*!

Convention 6.1.1 *Henceforth, we shall write* $\texttt{CompLen}(\mathcal{A})$ *for* $\texttt{CompLen}(\mathcal{A}, p)$.

6.2 Assumptions in the Description of Algorithm EuclideanMotion

1. When running an instance of Algorithm EuclideanMotion we think of the algorithms being present in each processor in $\mathcal{M}(K)$.

2. For our purposes of simplicity and exposition, at any time that $\mathcal{M}(K)$ is running one of the instances of Algorithm EuclideanMotion, there is not some unrelated process, e.g., another instance of Algorithm EuclideanMotion, running at the same time. (We do *not* discuss here, for example, the interesting problem of the simulation of two or more particles undergoing different, or perhaps the same, kinds of euclidean motion.)

3. For each instance of EuclideanMotion that we present, there will be a specified processor, $\mathcal{M}(I)$, where the motion simulated by that instance will begin. This $\mathcal{M}(I)$ will be called, *only at the beginning of the simulation*, the *Initiator*.

4. At any time, a processor is said to have completed a *pass* of Algorithm EuclideanMotion iff at that time it completes executing step 3 (see Section 6.4.4) of EuclideanMotion.

5. As soon as a processor, $\mathcal{M}(P)$, has completed a pass of EuclideanMotion, $\mathcal{M}(P)$ goes back to executing step 1 of EuclideanMotion.

6. Algorithm EuclideanMotion, in each processor, $\mathcal{M}(P)$, has its own, predefined, "automatic" variables called `SimClock` and `PassCount` that can be accessed. `SimClock`, at each time, contains the integer number of instruction cycles completed since the beginning of the simulation. `PassCount`, at each time, contains the integer number of passes completed by $\mathcal{M}(P)$ since the beginning of the simulation. `SimClock` is "automatic" in the sense that, with each instruction cycle completed by $\mathcal{M}(P)$, this variable is automatically incremented by one in $\mathcal{M}(P)$. `PassCount` is "automatic" in the sense that, with each pass, of Algorithm EuclideanMotion, completed by $\mathcal{M}(P)$, this variable is automatically incremented by one in $\mathcal{M}(P)$.

7. A simulation begins, exactly when an external (to the mesh) input turns on all the processors to begin executing Algorithm EuclideanMotion *and* that external input initializes the variable `Done` to **false** in the *Initiator* and to **true** in all the other processors in $\mathcal{M}(K)$. The initialization of the *Initiator* will sometimes be referred to as its *activation*.

8. For our purposes of simplicity and exposition, at any time after a simulation begins, no external input which influences the progress of simulation is provided to any processor in the mesh.

In the explanation of the steps of Algorithm EuclideanMotion (Section 6.4.5), we indicate the parts of Algorithm EuclideanMotion that need to be appropriately replaced for each specific instance of EuclideanMotion that is described in Chapters 7, 8, and 9.

6.3 Universal Constants for $\mathcal{M}(K)$

The choice of $\mathcal{M}(K)$ essentially determines some sets of values and some scalar values. We call these values *universal constants*. Adopting the terminology of Pascal, we refer to the sets of values as *pre-defined types*, and to the scalars as *pre-defined constants*.

6.3.1 Pre-defined Types

As seen in Chapter 4, the choice of $\mathcal{K}$ determines the representation of points corresponding to processors in the idealized, n-dimensional mesh based on $\mathcal{K}$, and also determines the neighbors of each processor. Our algorithms need to refer to the corresponding software entities frequently, so we have the following pre-defined types of software entities; our algorithms have variables whose values range over the values of these types.

type `vector`;

Recall that $\mathcal{K}$ is an n-dimensional lattice integrally-representing itself, and that points in $\mathcal{K}$ are represented as n''-tuples of integers. Variables of type `vector` range over points in $\mathcal{K}$, i.e., over n''-tuples of integers.

type `Directions`;

A value of this type is a basic direction of $\mathcal{K}$ (see Section 3.1).

6.3.2 Pre-defined Constants

In this section, we list the constants universal for $\mathcal{M}(K)$ and their values (See Chapter 5 for a justification of these values.).

const `SpaceDil` $= 2 \cdot \mathrm{Diam}(K)$;

const `AngRes`;

This constant, `AngRes`, is a record consisting of three components as follows.

$$\begin{aligned} \texttt{AngRes.SineNum} &= 8\mathrm{Diam}(K)^2 \\ \texttt{AngRes.CosineNum} &= 16\mathrm{Diam}(K)^4 - 1 \quad \text{and} \\ \texttt{AngRes.Denom} &= 16\mathrm{Diam}(K))^4 + 1. \end{aligned} \tag{25}$$

We will use `AngRes` as if it also had a value *inside* trigonometric functions, where, of course

$$\begin{aligned}\sin(\texttt{AngRes}) &= \frac{\texttt{AngRes.SineNum}}{\texttt{AngRes.Denom}} \text{ and}\\ \cos(\texttt{AngRes}) &= \frac{\texttt{AngRes.CosineNum}}{\texttt{AngRes.Denom}}.\end{aligned}$$

It is understood that these trig expressions are merely shorthand for the obvious corresponding rational numbers based on Equations (25).

For simplicity of exposition we will also use `AngRes` in trig expressions such as $\cos((k+1)\cdot\texttt{AngRes})$ where it is understood that this stands for the rational number to be computed out of such rationals as $\cos(\texttt{AngRes})$, $\sin(\texttt{AngRes})$, $\cos(k\cdot\texttt{AngRes})$, and $\sin(k\cdot\texttt{AngRes})$.

6.4 Algorithm EuclideanMotion: The General Form of the Algorithms

We first define the non-universal constants (Section 6.4.1), types (Section 6.4.2), and variables (Section 6.4.3) we use in the algorithm, and give a brief explanation immediately following each definition. Then, in Section 6.4.4, we present the steps in the executable part of the algorithm, followed by an explanation (see Section 6.4.5) of each step in the algorithm. Steps 1, 2, and 3 of the algorithm below correspond to parts 1, 2 and 3, respectively, of the three-part outline of each pass (see Section 6.1) of each of our algorithms. As explained in Sections 6.1 and 6.2, we will assume that after completing the execution of step 3, a processor goes back to execute step 1.

6.4.1 Constants

Suppose $\mathcal{A}$ is an instance of Algorithm EuclideanMotion.

const `CompLen`;

`CompLen`, is an integer constant whose value is set to be $\texttt{CompLen}(\mathcal{A})$. (See Definition 6.1.1, Remark 6.1.1, and Convention 6.1.1, each in Section 6.1.) It is quite important to note the following

Remark 6.4.1 *It turns out*[3] *that, if $\mathcal{A}$ is any one of* our *instances of EuclideanMotion, the user can calculate a correct value of* `CompLen` *to use* before $\mathcal{A}$ *is executed since the execution time of $\mathcal{A}$ is* not *affected by which value this constant receives; hence, a vicious circle is nicely avoided!*

PROOF. The proof is deferred until Section 6.4.4. □

6.4.2 Types

type `VectorSet`= **set of** `vector`;

Values of this type are sets of vectors.

type *MessageType* = ⟨ A record whose fields hold information appropriate for executing Algorithm EuclideanMotion. ⟩;

Values of this type are finite sequences of software entities, which are either integers, or integer vectors. Such values represent the messages that can be sent from a processor to any one of its neighbors. These messages are the only means for a processor to communicate with neighboring processors.

type *DirectionBuffer* = **record**

InMessage, *OutMessage* : *MessageType*;

SendTime : `integer`;

⋮

end;

Values of this type are aggregates of values of type *MessageType*, integers, or integer vectors, and values of basic types (integer and boolean). Values of this type contain all relevant information pertaining to directions in the mesh.

[3]See Sections 7.1.4, 7.2.5, 9.2.4 and 9.3.4

6.4.3 Variables

var `Done`: **boolean**;

At the beginning of a simulation, the boolean variable `Done` in each processor, $\mathcal{M}(P)$, except the *Initiator*, has the value **true**. The *Initiator* has the value **false**. This variable is set to **false** automatically when $\mathcal{M}(P)$ receives a message. It can also be set to **true** or **false** by an algorithm. This latter is used in algorithms TRANSLATE and REVOLVE (See Chapters 8 and 9).

var D : `Directions`;

This is a variable ranging over the values of type `Directions`.

var *InDirs*, *SendDirs*, *SendThisPass* : **set of** `Directions`;

These are variables that range over the subsets of the set of values of type `Directions`. *InDirs* is used to store the set of directions from which a processor gets messages; at the beginning of each simulation it has the value $\emptyset$. *SendDirs* is used to store the set of directions in which the processor needs to send messages, and *SendThisPass* is used to store the set of basic directions in which the processor needs to send messages at the end of the current pass.

var *Buf* : *DirectionBuffer* **indexed by** `Directions`;

This variable stores an indexed set, whose elements are elements of *DirectionBuffer* indexed by an element of `Directions`. We do not assume any ordering on the set `Directions`, and hence no ordering on the elements of this variable. By definition, this set *does not* contain two elements that are indexed by the same element of `Directions`. We note here two ways to implement this variable, but any implementation achieving this effect will do. One way to implement this variable is as an array variable, indexed by the values of type `Directions`, and containing values of type *DirectionBuffer*. An alternate method for implementing this variable is as a function from `Directions` to values of type *DirectionBuffer*. The aim is to be able to associate, with each basic direction, a value of type *DirectionBuffer*. At any stage in the simulation, for each basic direction in the mesh there is a value of type *DirectionBuffer*.

var REPR : **boolean**;

Roughly, in each processor, REPR = **true** during a pass is used to denote whether the phenomenon being simulated is being represented by that processor sometime during that pass. This variable is particularly relevant and useful in the algorithm template CURVE and its specific instances; we discuss the exact meaning of this variable for CURVE in Chapter 8. For algorithms WAVE and WAVE′ we have a choice as to whether to use REPR = **true** to remember that the wavefront was ever there or to remember that it is there now. For convenience, so we don't have to clutter WAVE and WAVE′ in step 2 (see below) with updating REPR to handle the latter, we use REPR in WAVE and WAVE′ only for the former.

REPR is initialized to **false** in each processor, at the beginning of a simulation. Specific instances of EuclideanMotion set and reset this variable in steps 1 and/or 2 of EuclideanMotion.

6.4.4 Executable Part of Algorithm EuclideanMotion

begin

1. **if** `Done` **then goto** step 3;
   else
    if (**not** REPR) **then**
      Initialize Parameters using external input, if any,
        for wave algorithms make REPR = **true** if $InDirs \neq \emptyset$, and
        initialize $\{\ Buf\langle D\rangle.InMessage | D \in InDirs\ \}$, if non-empty;
    endif;
   endif;
2. Compute and update REPR and *SendDirs*and
   for each new $D \in SendDirs$, compute values for and update
     $Buf\langle D\rangle.SendTime$ and $Buf\langle D\rangle.OutMessage$;
   endfor;
   Possibly compute some more;
   $SendThisPass := \{D \in SendDirs \mid Buf\langle D\rangle.SendTime = \texttt{PassCount}\}$;
3. **idle till** $\texttt{SimClock} = (\texttt{CompLen} \cdot \texttt{PassCount}) - 1$;
    simul send

$Buf\langle D\rangle.OutMessage$ in direction D, for each $D \in SendThisPass$;
endsend;

end.

PROOF (OF REMARK 6.4.1 DEFERRED FROM SECTION 6.4.1). First we note the fact that, in any one of *our* instances of EuclideanMotion, the only reference to `CompLen` is in the constant time test on line 1 in step 3 of EuclideanMotion. Secondly, we note that the execution of steps 1, and 2 is bounded by a maximum time, MT instruction cycles, depending on $\mathcal{K}$ and K. We assumed (Assumption 4.3.5) that the time to execute each pass of the **idle till** instruction in step 3 is one instruction cycle. Furthermore, we assumed (Assumption 4.3.7) that the number of instruction cycles to carry out a **simul send** is also one. Hence, we may set `CompLen` $= MT + 2$. □

6.4.5 Explanation of the Steps in Algorithm EuclideanMotion

Step 1. Whenever $\mathcal{M}(P)$ receives a message, say at simulation time t,

1. $InDirs = \{D|\mathcal{M}(P)$ received a message from $\mathcal{M}(P+D)$ at simulation time $t\}$, and,
2. for each D in $InDirs$, $Buf\langle D\rangle.InMessage$ contains the message received by $\mathcal{M}(P)$ from $\mathcal{M}(P+D)$ at simulation time t.

When, at a simulation time t, $\mathcal{M}(P)$ does *not* receive a message, $InDirs$ in $\mathcal{M}(P)$ retains its value from simulation time $t-1$. If at the beginning of a pass, the value of `Done`, in $\mathcal{M}(P)$, is **false**, then $\mathcal{M}(P)$ initializes parameters required in the computations to be done in steps 2 and 3, after extracting the information contained in $Buf\langle D\rangle.InMessage$, for each $D \in InDirs$. The information in $Buf\langle D\rangle.InMessage$ depends on the specific EuclideanMotion algorithm being used. For each *instance* of EuclideanMotion that we present, we specify in that presentation any necessary details regarding the required information content of $Buf\langle D\rangle.InMessage$.

Step 2. Computations performed in this step determine the set, *SendDirs*, of basic directions in which $\mathcal{M}(P)$ needs to send messages. For each basic direction in *SendDirs*, $\mathcal{M}(P)$ also determines the message that needs to be sent to $\mathcal{M}(P+D)$, and the time (pass number) in which to send that message. These quantities are stored in $Buf\langle D\rangle.OutMessage$ and $Buf\langle D\rangle.SendTime$, respectively, for each D in *SendDirs*. It is assumed that the computations in step 2 involve *only integer and integer vector arithmetic*, and depend only on the constants (universal and other) and the information gleaned, in step 1, from the set of incoming messages. For each specific instance, of EuclideanMotion, described in this thesis, we explicitly show a procedure to determine *SendDirs*, and hence show that it is possible to determine *SendDirs* in a bounded number of basic instructions. Finally, $\mathcal{M}(P)$ forms the set, *SendThisPass*, of basic directions in which a message is to be sent at the end of the current pass. Clearly, then, for all D in *SendThisPass*, the value of $Buf\langle D\rangle.SendTime$ is the same, and equals the number of the current pass.

Step 3. To each neighbor $\mathcal{M}(P+D)$, where D is in *SendThisPass*, $\mathcal{M}(P)$ sends the corresponding message, stored in $Buf\langle D\rangle.OutMessage$, at simulation time $\texttt{CompLen}\cdot\texttt{PassCount}-1$. This ensures that each of these neighbors receives its message from $\mathcal{M}(P)$ at simulation time $\texttt{CompLen}\cdot\texttt{PassCount}$.

The choice of `CompLen` then guarantees that it is possible for $\mathcal{M}(P)$ to send the messages at simulation time $\texttt{CompLen}\cdot\texttt{PassCount}$ after completing steps 1 and 2, and the part of step 3 before the actual sending.

Remark 6.4.2

1. *As we shall see, the virtual speed of the simulated objects for each* $\mathcal{A}\in\{\text{WAVE},\text{WAVE}'\}$ *is* $1/\texttt{SpaceDil}$ *units of euclidean distance per the time required to carry out one pass of* $\mathcal{A}$*.*

2. *As we shall also see, there is a lattice and resolution dependent positive integer constant* m *such that the virtual speed of the particle for* TRANSLATE *is*

1/`SpaceDil` *units of euclidean distance per the time required to carry out m passes of $\mathcal{A}$.*

3. *Finally, the virtual angular speed of the simulated particle for* REVOLVE *is* 1/`AngRes` *radians per the time required to carry out m passes of* REVOLVE.[4]

[4] Of course the numerical value intended for the record `AngRes` is that of MaxTheta(K).

Chapter 7

Simulating Radiation

In this chapter, we describe two algorithms, WAVE (Section 7.1) and WAVE$'$ (Section 7.2), to broadcast a message from a processor in $\mathcal{M}(K)$ at the maximum possible uniform radial speed. Algorithm WAVE$'$ has certain advantages over algorithm WAVE, and we discuss these in Section 7.2 before discussing the algorithm WAVE$'$ itself. We call these algorithms WAVE and WAVE$'$ because of the similarity to the simulation of a radially propagating spherical wavefront.

In this chapter, our description of these algorithms will consist of showing that each of the steps of Algorithm EuclideanMotion can be performed, in each of the two cases WAVE and WAVE$'$, using integer and integer vector arithmetic only. In Chapter 10, we present complete algorithms, for WAVE and WAVE$'$, in a Pascal-like specification language. There, we will freely use the equivalences between expressions shown in this chapter.

7.1 Algorithm WAVE

7.1.1 Inputs

The *Initiator* needs to be activated.

7.1.2 Contents of Messages

Each processor, $\mathcal{M}(P)$ ($\neq \mathcal{M}(I)$), must receive in its messages the n'' tuple of integers $(P - I)$.

7.1.3 Computing *SendDirs*

For WAVE, the computation in step 2 of EuclideanMotion should ensure that every processor $\mathcal{M}(P)$ which enters step 2 of EuclideanMotion, *eventually* sends a message to each of its immediate neighbors (if any), $\mathcal{M}(P+D)$, such that $P+D$ is at a greater euclidean distance from I than P. (This, together with the facts that $\mathcal{M}(I)$ enters step 2 and that K is I-convex in $\mathcal{K}$, will ensure that each $\mathcal{M}(P) \neq \mathcal{M}(I) \in \mathcal{M}(K)$ will eventually receive the broadcast message.) *An* algebraic expression, then, of the condition for $\mathcal{M}(P)$ to send a message to $\mathcal{M}(P + D)$ is

$$(d(P, I))^2 < (d((P + D), I))^2. \tag{26}$$

To enter step 2 of EuclideanMotion, $\mathcal{M}(P)$ must have received a message informing it of the value of $P - I$. Hence, we make the actual condition (easily seen to be equivalent to Equation 26) on a basic direction D for it to be added to *SendDirs* in step 2 inside $\mathcal{M}(P)$ to be

$$(P - I, P - I) < ((P - I + D), (P - I + D)) \tag{27}$$

Clearly this condition is easy to compute from the available information and the computation can be carried out using only simple integer and integer vector arithmetic.

7.1.4 Computing *SendTime*

We describe how $\mathcal{M}(P)$ in step 2 next computes $Buf\langle D\rangle.SendTime$, for each new $D \in SendDirs$.

Suppose p is the number of passes completed before the pass under consideration, i.e., suppose $p =$ the current value in `PassCount`. Suppose D is any new element of *SendDirs*.

Recall from Remark 6.4.2 that the virtual speed of the wavefront is $1/\texttt{SpaceDil}$ units of euclidean distance per the time required to carry out one pass of WAVE. We know that the message took p passes to reach $\mathcal{M}(P)$, so it is important to know how much *more* time it should take to reach $\mathcal{M}(P+D)$. We also know that, in traveling from $\mathcal{M}(I)$ (through $\mathcal{M}(P)$) to $\mathcal{M}(P+D)$, the total number of passes the message should take is the closest integer to

$$d(I, P+D)/(1/\texttt{SpaceDil}) = d(I, P+D) \cdot \texttt{SpaceDil}.$$

This integer is greater than p since, by choice of $\texttt{SpaceDil}$,

$$(d(I, P+D) - d(I, P)) > 1/\texttt{SpaceDil}.$$

The number of passes, call it t_D, beyond the p passes already elapsed, that $\mathcal{M}(P)$ should wait to send the message to $\mathcal{M}(P+D)$ is, then, a positive integer such that $(p+t_D)\cdot(1/\texttt{SpaceDil})$ is as close as possible to $d(I, P+D)$. By Proposition 5.1.1 t_D is unique! Since $(d(I, P+D) - d(I, P)) \leq \text{MinLgth}$, $t_D \leq \text{MinLgth}/(1/\texttt{SpaceDil})$. Hence, t_D is the unique integer such that $1 \leq t_D \leq \text{MinLgth}/(1/\texttt{SpaceDil})$ and $(p+t_D)\cdot(1/\texttt{SpaceDil})$ is as close as possible to $d(I, P+D)$. It is straightforward, but tedious to verify that the following integer and integer vector arithmetic procedure calculates t_D. After t_D is found, the algorithm sets

$$\mathit{Buf}\langle D\rangle.\mathit{SendTime} = p + t_D.$$

begin Procedure to find t_D;

1. Find the minimum t such that $1 \leq t \leq \text{MinLgth}(\mathcal{K}) \cdot \texttt{SpaceDil}$, and
$$(p+t)^2 \geq \texttt{SpaceDil}^2 \cdot (((P+D)-I), ((P+D)-I));$$

2. **set**[1]
$$t_D = \begin{cases} t, & \text{if } (2(p+t)-1)^2 < 4\texttt{SpaceDil}^2 \cdot (d((P+D)-I))^2; \\ t-1, & \text{otherwise.} \end{cases}$$

end. {Procedure to find t_D}

[1]It is clear at this stage that the integer multiple of $(1/\texttt{SpaceDil})$ that is closest to $d((P+D), I)$ is either t or $(t-1)$. Here, we use an equivalent (showing the logical equivalence is a relatively straightforward exercise) check: we compare the squares of the average of these two integer multiples of $(1/\texttt{SpaceDil})$ with $(d((P+D), I))^2$, to get the desired value for t_D.

7.1.5 Computing *OutMessage*

For each new $D \in \mathit{SendDirs}$, the algorithm sets $\mathit{Buf}\langle D\rangle.\mathit{OutMessage}= P - I + D$.

7.2 A Physically Natural Alternative to WAVE

In this section, we discuss an alternate algorithm for broadcasting from a processor (or generating a wavefront) in a mesh. This algorithm is proven correct to run on a mesh based on the lattice A_n.

We first motivate this algorithm and discuss some of its possible merits over algorithm WAVE.

Convention 7.2.1 *Henceforth,* $A \subseteq A_n$ *and* $\mathcal{M}(A)$ *is a suitable mesh as witnessed by* A_n*.*

$\mathcal{M}(I)$ is in $\mathcal{M}(A)$. WAVE$'$, then, broadcasts from $\mathcal{M}(I)$ in $\mathcal{M}(A)$.

In physical reality, it seems quite unlikely that a point in Euclidean space "finds out" about its relative coordinates with respect to the origin of a wave. For example, it seems unlikely that a water molecule, on, say, the surface of a lake, "knows" about the origin of an impending wave as it is being affected by that wave. It seems more likely that the point is activated by impulses (informationless, except for magnitude, type, and direction) from neighboring points and these impulses cause a point to be on a wavefront at the appropriate time.

In algorithm WAVE presented in Section 7.1 just above, each processor uses its relative coordinates with respect to the origin of the broadcast. As pointed out in the previous paragraph, this is physically unlikely.

As will be seen below, Algorithm WAVE$'$ does not have this physically unlikely characteristic. Every processor, $\mathcal{M}(P)$, in $\mathcal{M}(A)$ that eventually receives the broadcast message from $\mathcal{M}(I)$, explicitly uses the *set of basic directions* from which $\mathcal{M}(P)$ *receives* messages in "deciding" which of its neighbors to send messages to. Intuitively, one can think of those message *to* $\mathcal{M}(P)$ as a "push" by the wave emanating from $\mathcal{M}(I)$. This seems more physically natural than the mechanism used in WAVE.

While the contents of messages for WAVE$'$ do not involve relative coordinates with respect to the origin (as does WAVE), it is interesting to consider whether there is physical significance to the actual content of these messages (Section 7.2.3). Many physical waves in 3-space carry information about the magnitude of a force. In 3-space the magnitude of many such forces falls off in proportion to the inverse of the square of the distance from the origin of the force wave; also the surface area of the wave front correspondingly increases with this same square of the distance from the origin. It turns out that one of the messages contents for WAVE$'$ is the square of the distance from the origin, but this for any dimension > 1. Actually, we need to send distance information and sending squares of distance is our way to work with integers only. The other two message contents for WAVE$'$, while a bit technical, are also directly or indirectly information about the distance from the origin. Interestingly, one of the message contents for WAVE$'$ is the minimal (possibly meandering) lattice path length distance. If real physical space is lattice-like, one might expect this non-euclidean lattice distance to also be a factor in the falling off of forces with distance as a force wave progresses. An exact working out of the true physical meaning, if any, of all this is an intriguing open problem (Chapter 11).

In the next section, we will state and prove some technical results that we will use in the presentation of WAVE$'$.

7.2.1 Technical Results

The following properties of A_n are explicitly stated or are simple consequences of results in Chapters 3 and 4.

Property 7.2.1 *$A_n \subseteq \mathbf{Z}^{n+1}$, and each point, P, in A_n can be represented uniquely as an $(n+1)$-tuple of integers, $(p_0, p_1, \ldots, p_n)$, so that $\sum_{i=0}^{n} p_i = 0$ (see Section 3.1). We will refer to the i^{th} element of the $(n+1)$-tuple representing a point P by $(P)_i$.*

Property 7.2.2 *(Recall the definition of λ^n_{ij} from part 2 of Example 3.2.1, and the definition of Λ^n from part 1 of Note 3.2.1.) From part 1 of Note 3.2.1 we have that $\Lambda^n = \mathrm{Dirs}(A_n)$.*

Notation 7.2.1 *Since, in this section, n is fixed, We will write λ_{ij} for λ_{ij}^n.*

Property 7.2.3 *For points P and O in A_n,*

$$\mathrm{GraphPL}(P, O) = \sum_{(P-O)_i > 0} (P)_i = - \sum_{(P-O)_i < 0} (P - O)_i$$

This is a simple consequence of Proposition 4.1.1 using Λ^n.

Property 7.2.4 *A_n is an integrally-representable, minimally generated lattice (see Note 4.1.1), which, in fact, integrally-represents itself in $\mathbf{Z}^{n+1}$. Then, for any two points P and Q in A_n, the inner product of these two points, each an $(n+1)$-dimensional vector, is given by*

$$(P, Q) = \sum_{i=0}^{n} (P)_i \cdot (Q)_i$$

Recall the definitions of $Outer_{\mathcal{D}}$ and $Inner_{\mathcal{D}}$ for minimally generated lattices, $\mathcal{D}$ (Definition 4.1.5).

Notation 7.2.2 *In this section, we will abbreviate $Inner_{A_n}$ to Inner and $Outer_{A_n}$ to Outer.*

Definition 7.2.1 *Suppose P and O are points in A_n. Then,*

1. *$InnerDirs(P, O) = \{D \in \mathrm{Dirs}(A_n) \mid (P + D) \in Inner(P, O)\}$.*
2. *$OuterDirs(P, O) = \{D \in \mathrm{Dirs}(A_n) \mid (P + D) \in Outer(P, O)\}$.*

Lemma 7.2.1 *Suppose $P(= (p_0, p_1, \ldots, p_n)) \in A_n$. Then,*

$$(P \neq \vec{0}) \iff (\exists i, j, 0 \leq i \neq j \leq n, \text{ such that } (p_i > 0 \ \&\ p_j < 0)).$$

PROOF. Immediate from Property 7.2.3 above. □

Lemma 7.2.2 *Suppose $P, Q \in A_n$. Let $P = (p_0, p_1, \ldots, p_n)$ and $Q = (q_0, q_1, \ldots, q_n)$. Let $Q = P + \lambda_{ij}$, for some $i \neq j, 0 \leq i, j \leq n$. Then,*

(i) $(Q \in Outer(P, \vec{0}) \iff ((p_i \geq 0)$ and $(p_j \leq 0)))$ *and*

(ii) $(Q \in Inner(P, \vec{0}) \iff ((p_i < 0)$ and $(p_j > 0))$.

PROOF. Suppose $P(= (p_0, p_1, ..., p_n)) \in A_n$. Let $Q \in A_n$ be a neighbor of P. Thus, there is a λ_{ij} such that, $Q = P + \lambda_{ij}$, i.e., $Q = (p_0, p_1, ..., (p_i + 1), ..., (p_j - 1), ..., p_n)$. Then, from the definition of outer and inner neighbors, and Property 7.2.3 above we have this lemma. □

Corollary 7.2.1 *If Q is an outer neighbor of P, with respect to the $\vec{0}$, then, $d(Q, \vec{0}) > d(P, \vec{0})$.*

PROOF. Let $P = (p_0, \ldots, p_n)$. Suppose Q is an outer neighbor of P. It suffices to show that $(d(Q, \vec{0}))^2 > (d(P, \vec{0}))^2$. By Lemma 7.2.2, $Q = P + \lambda_{ij}$ such that $p_i \geq 0$ and $p_j \leq 0$. Then,

$$\begin{aligned}
(d(Q, \vec{0}))^2 &= ((P + \lambda_{ij}), (P + \lambda_{ij})) \\
&= (P, P) + (\lambda_{ij}, \lambda_{ij}) + 2(P, \lambda_{ij}) \\
&= (P, P) + 2 + 2(p_i - p_j) \\
&> (d(P, \vec{0}))^2.
\end{aligned}$$

□

Theorem 7.2.1

$$\begin{aligned}
&(P + \lambda_{ij}) \in Outer(P, \vec{0}) \\
&\iff \lambda_{ij} \in \left[\bigcap_{D \in InnerDirs(P, \vec{0})} \{D' \in \mathrm{Dirs}(A_n) | (D, D') \leq 0\} \right].
\end{aligned}$$

PROOF. Let $P = (p_0, p_1, \ldots, p_n)$. We have the following cases to consider. (In the following cases, i, j, k, and l range between, and inclusive of, 0 and n.)

Case 1 $\exists i, j$, so that $p_i = 0$ and $p_j = 0$

Let $P + \lambda_{kl}$ be an inner neighbor of P. Then, by Lemma 7.2.2, $p_k < 0$ and $p_l > 0$. Then, $\{i, j\} \cap \{k, l\} = \emptyset$, and hence $(\lambda_{ij}, \lambda_{kl}) = 0$. Furthermore, by Lemma 7.2.2, $(P + \lambda_{ij}) \in Outer(P, \vec{0})$.

Case 2 $\exists i, j$, so that $p_i = 0$ and $p_j > 0$

Thus, there exists an inner neighbor $P + \lambda_{kj}, k \neq i$ and $(\lambda_{ij}, \lambda_{kj}) > 0$. Furthermore, by Lemma 7.2.2, $(P + \lambda_{ij}) \notin Outer(P, \vec{0})$.

Case 3 $\exists i, j$, so that $p_i = 0$ and $p_j < 0$

Consider all inner neighbors $P + \lambda_{kl}, l \neq i$.
Then, $l \neq j$ since $p_l > 0$.
$(k = j) \Rightarrow (\lambda_{ij}, \lambda_{jl}) < 0$
$(k \neq j) \Rightarrow (\lambda_{ij}, \lambda_{kl}) = 0$
Furthermore, by Lemma 7.2.2, $(P + \lambda_{ij}) \in Outer(P, \vec{0})$.

Case 4 $\exists i, j$, so that $p_i > 0$ and $p_j = 0$

Consider all inner neighbors $P + \lambda_{kl}, k \neq i$.
Then, $l \neq j$ since $p_l > 0$.
$(l = i) \Rightarrow (\lambda_{ij}, \lambda_{ki}) < 0$
$(l \neq i) \Rightarrow (\lambda_{ij}, \lambda_{kl}) = 0$
Furthermore, by Lemma 7.2.2, $(P + \lambda_{ij}) \in Outer(P, \vec{0})$.

Case 5 $\exists i, j$, so that $p_i > 0$ and $p_j > 0$

Then there exists an inner neighbor $P + \lambda_{kj}, k \neq i$. Then $(\lambda_{ij}, \lambda_{kj}) > 0$.
Furthermore, by Lemma 7.2.2, $(P + \lambda_{ij}) \notin Outer(P, \vec{0})$.

Case 6 $\exists i, j$, so that $p_i > 0$ and $p_j < 0$

Consider all inner neighbors $P + \lambda_{kl}, k \neq i, l \neq j$ since $p_k < 0$ and $p_l > 0$.
$((k = j)$ and $(l \neq i)) \Rightarrow (\lambda_{jl}, \lambda_{ij}) < 0$.
$((k \neq j)$ and $(l = i)) \Rightarrow (\lambda_{ki}, \lambda_{ij}) < 0$.
$((k = j)$ and $(l = i)) \Rightarrow (\lambda_{ji}, \lambda_{ij}) < 0$.
$((k \neq j)$ and $(l \neq i)) \Rightarrow (\lambda_{kl}, \lambda_{ij}) = 0$.
Furthermore, by Lemma 7.2.2, $(P + \lambda_{ij}) \in Outer(P, \vec{0})$.

Case 7 $\exists i, j$, so that $p_i < 0$ and $p_j = 0$.

Then there exists an inner neighbor $P + \lambda_{il}, l \neq j$. Then, $(\lambda_{ij}, \lambda_{il}) > 0$. Furthermore, by Lemma 7.2.2, $(P + \lambda_{ij}) \notin Outer(P, \vec{0})$.

Case 8 $\exists i, j$, so that $p_i < 0$ and $p_j > 0$

Then there exists an inner neighbor $P + \lambda_{ij}$. Then $(\lambda_{ij}, \lambda_{ij}) > 0$. Furthermore, by Lemma 7.2.2, $(P + \lambda_{ij}) \notin Outer(P, \vec{0})$.

Case 9 $\exists i, j$, so that $p_i < 0$ and $p_j < 0$.

Then there exists an inner neighbor $P + \lambda_{il}, l \neq j$. Then $(\lambda_{ij}, \lambda_{il}) > 0$. Furthermore, by Lemma 7.2.2, $(P + \lambda_{ij}) \notin Outer(P, \vec{0})$.

Thus, we have the theorem. □

Lemma 7.2.3 *Suppose $\vec{0} \in A$. For each $P \in A$, let $InDirs(P)$ be a variable containing, at any time during the process mentioned below, a representation of the set of basic directions D such that $\mathcal{M}(P)$ received a message from $\mathcal{M}(P + D)$.*
For each $P \in A$, let $SendDirs(P)$ be a variable, which when assigned a value contains a representation of the set

$$\left[\bigcap_{D \in InDirs(P)} \{D' \in \mathrm{Dirs}(A_n) \mid (D, D') \leq 0\} \right].$$

For each $P \in A$, for each D in $SendDirs(P)$, let $OutMessage(D)$ be a variable containing the message to be sent to $\mathcal{M}(P+D)$. Suppose the assignment of $SendDirs(P)$ and the assignment of each $OutMessage(D), D \in SendDirs(P)$, takes a bounded amount of simulation time, T_M. For each $P \in A$, for each basic direction D, let $InMessage(D)$ be a variable containing the last message received from $\mathcal{M}(P + D)$, if any. When the scheme described below begins, for each $P \in A$, for each basic direction D, $InMessage(D)$ is undefined.

Suppose $t : A \to \mathbf{Z}$ satisfying:

1. $t(\vec{0}) = 0$, *and*

2. $(\forall P \in A)(\forall Q \in (Outer(P, \vec{0}) \cap A))$,

   (a) *if* $Msgs = \{\ InMessage(D) \mid D \in InnerDirs(Q, \vec{0})\ \}$ *has no undefined values, then* $t(Q)$ *can be computed, in a bounded amount of simulation time,* T_W*, from* $InnerDirs(Q, \vec{0})$ *and* $Msgs$ *(We will assume that, if* $Msgs$ *is not completely defined, then the computation of* $t(Q)$ *takes an infinite amount of time.), and*

   (b) $(t(Q) > (t(P) + T_M + T_W)$.

The scheme:

1. $\mathcal{M}(\vec{0})$ *sends a message to all its neighbors* $\mathcal{M}(P)$ *at corresponding times* $t(P)$.

2. *Whenever* $\mathcal{M}(P)$ *in* $\mathcal{M}(A)$ *gets a message,* $\mathcal{M}(P)$

   (a) *computes the value to be assigned to* $SendDirs(P)$ *using the current value of* $InDirs(P)$,

   (b) *for each* $D \in SendDirs(P)$*, computes* $OutMessage(D)$*, and*

   (c) *for each* $D \in SendDirs(P)$*, computes* $t(P + D)$.

   Then, for each direction $D \in SendDirs(P)$*,* $\mathcal{M}(P)$ *sends the message* $OutMessage(D)$ *to* $\mathcal{M}(P + D)$ *at simulation time* $t(P + D)$*, if possible; if not possible,* $\mathcal{M}(P)$ *does not send a message to* $\mathcal{M}(P + D)$*, and*

3. *No other messages are sent or received except as described in (1) and (2) just above.*

Then*, for each* P *in* A*,*

(C1) $(\mathcal{M}(P)$ *receives all its messages at simulation time* $t(P))$*, and*

(C2) $InDirs(P) = InnerDirs(P, \vec{0})$.

Proof. Suppose the hypotheses. Then,

$$(\forall P, Q \in A)[(P - Q) \in SendDirs(Q) \iff (Q - P) \in InDirs(P)]. \tag{28}$$

By definition of t in the lemma, there exists a finite sequence $t_0(=0), t_1, \ldots, t_k$ of integers such that

1. $(\forall i, 0 \leq i < k)[t_i < t_{i+1}]$, and
2. $(\forall i, 0 \leq i \leq k)(\exists P \in A)[t(P) = t_i]$.

We now prove the following claim using mathematical induction on this sequence of integers.

Claim 7.2.1 $(\forall i, 0 \leq i \leq k)(\forall P \in A$ *such that* $t(P) = t_i)$, [**(C1)** *and* **(C2)**].

PROOF CLAIM.

Base case: i = 0

In this case, $t_i = 0$. From the definition of t,

$$(\forall P \in A)[(t(P) = 0) \Longleftrightarrow (P = \vec{0})].$$

Then, [**(C1)** and **(C2)**] is trivially satisfied.

Induction Hypothesis (IH): Suppose $m, 0 \leq m < k$ such that the claim is true for all $i \leq m$.

Suppose $P \in A$ such that $t(P) = t_{m+1}$.

By Clause 2 of the description of the sending scheme, if $\mathcal{M}(P)$ gets any messages, it gets them all at time $t(P)$, and so

$$P \text{ satisfies } \textbf{(C1)}. \tag{29}$$

From the definition of t and $Inner$,

$$(\forall P' \in Inner(P, \vec{0}))[t(P') < t(P)]. \tag{30}$$

From (IH) and (30),

$$(\forall P' \in (Inner(P, \vec{0}) \cap A))[\ \textbf{(C2)}\ ]. \tag{31}$$

Since A is $\vec{0}$–convex,

$$(Inner(P,\vec{0}) \cap A) = Inner(P,\vec{0}). \tag{32}$$

From (31), (32), the definition of $SendDirs$ in the statement of the lemma, and Theorem 7.2.1,

$$(\forall P' \in Inner(P,\vec{0}))[SendDirs(P') = OuterDirs(P',\vec{0})]. \tag{33}$$

From the definition of $Inner$ and $Outer$ (Definition 4.1.5),

$$(\forall P, Q \in A)\left[(Q \in Outer(P,\vec{0})) \Longleftrightarrow (P \in Inner(Q,\vec{0}))\right]. \tag{34}$$

From (34) and definition of $OuterDirs$ Definition 7.2.1,

$$(\forall P' \in Inner(P,\vec{0}))\left[(P - P') \in OuterDirs(P',\vec{0})\right]. \tag{35}$$

From (33) and (35),

$$(\forall P' \in Inner(P,\vec{0}))\left[(P - P') \in SendDirs(P')\right]. \tag{36}$$

From (36) and (28),

$$(\forall P' \in Inner(P,\vec{0}))\left[(P' - P) \in InDirs(P)\right]. \tag{37}$$

From (37) and definition of $InnerDirs$,

$$InnerDirs(P,\vec{0}) \subseteq InDirs(P). \tag{38}$$

Suppose, by way of contradiction, $\exists Q \in A$ such that

$$((Q - P) \in InDirs(P)), \text{ and} \tag{39}$$

$$(Q \notin Inner(P,\vec{0})). \tag{40}$$

From (28) and (39),

$$(P - Q) \in SendDirs(Q). \tag{41}$$

Since $\mathcal{M}(Q)$ sent a message to $\mathcal{M}(P)$, from clause 2 of the scheme described in the lemma, $t(Q) < t(P)$, and so (IH) holds for Q. Then from **(C2)** and Theorem 7.2.1,

$$SendDirs(Q) = OuterDirs(Q, \vec{0}). \tag{42}$$

From (42), the definition of OuterDirs, and (41),

$$P \in Outer(Q, \vec{0}), \tag{43}$$

giving a contradiction to (40) using the (34).

Thus, $InDirs(P) = InnerDirs(P, \vec{0})$, completing the induction step.

□ CLAIM

Now the lemma follows. □

We define the following sets for each P in A.

Definition 7.2.2

$$\begin{aligned} Opp(P, \vec{0}) &= \{P + \lambda_{ij} \mid \lambda_{ji} \in InnerDirs(P, \vec{0})\ \} \\ Rt(P, \vec{0}) &= \{P + \lambda_{ij} \mid \forall \lambda_{kl} \in InnerDirs(P, \vec{0}), (\lambda_{ij}, \lambda_{kl}) = 0\} \\ Obtuse(P, \vec{0}) &= Outer(P, \vec{0}) \setminus (Opp(P, \vec{0}) \cup Rt(P, \vec{0})) \end{aligned}$$

The following lemma follows from Definition 7.2.2 just above and Lemma 7.2.2.

Lemma 7.2.4

1. *$Opp(P, \vec{0}) \subseteq Outer(P, \vec{0})$,*

2. *$Rt(P, O) \subset Outer(P, \vec{0})$,*

3. *$Opp(P, \vec{0}) \cap Rt(P, \vec{0}) = \emptyset$, and*

4. *$Opp(P, \vec{0}) \cup Rt(P, \vec{0}) \cup Obtuse(P, \vec{0}) = Outer(P, \vec{0})$.*

□

Below, we prove three important lemmas (7.2.5, 7.2.6 and 7.2.7) regarding the squares of the euclidean distances of the outer neighbors of a point P in A. Let $P = (p_0, p_1, \ldots, p_n)$. Recall the definition of NormDiff$(\cdot,\cdot,\cdot)$ (Definition 4.1.5).

Lemma 7.2.5 *Suppose $Q \in Opp(P, \vec{0})$. Let $Q' = (P - (Q - P))$, i.e., the inner neighbor of P, with respect to $\vec{0}$, opposite Q. Then,*

$$\mathrm{NormDiff}(P, Q, \vec{0}) = \mathrm{NormDiff}(P, Q', \vec{0}) + 4.$$

Proof. Suppose $Q = P + \lambda_{ij}$, and $Q \in Opp(P, \vec{0})$. Then, $Q' = (P + \lambda_{ji})$. Then, by Lemma 7.2.2, $p_i > 0$ and $p_j < 0$. Therefore,

$$\mathrm{NormDiff}(P, Q, \vec{0}) = \left[\sum_{t=0}^{n} {p_t}^2 + 2p_i + 1 - 2p_j + 1 - \sum_{t=0}^{n} {p_t}^2\right] = [2p_i - 2p_j + 2]$$

and

$$\mathrm{NormDiff}(P, Q', \vec{0}) = \left[\sum_{t=0}^{n} {p_t}^2 - (\sum_{t=0}^{n} {p_t}^2 - 2p_i + 1 + 2p_j + 1)\right] = [2p_i - 2p_j - 2]\,.$$

Therefore, $\mathrm{NormDiff}(P, Q, \vec{0}) = \mathrm{NormDiff}(P, Q', \vec{0}) + 4$. □

Lemma 7.2.6 *Suppose $Q \in Rt(P, \vec{0})$. Then,*

$$\mathrm{NormDiff}(P, Q, \vec{0}) = 2.$$

Proof. Suppose $Q = (P + \lambda_{ij})$, and $Q \in Rt(P, \vec{0})$. Then, from the proof of Theorem 7.2.1, $(p_i = 0)$ and $(p_j = 0)$. Therefore,

$$\mathrm{NormDiff}(P, Q, \vec{0}) = \left[\sum_{t=0}^{n} {p_t}^2 + 2p_i + 1 - 2p_j + 1 - \sum_{t=0}^{n} {p_t}^2\right] = 2.$$

□

Lemma 7.2.7 *For each* Q $(= P + \lambda_{ij},$ *for some* $\lambda_{ij}) \in Obtuse(P, \vec{0})$, *let*

$$A_Q \stackrel{\text{def}}{=} \{\lambda_{kl} \in InnerDirs(P, \vec{0}) \mid (\lambda_{ij}, \lambda_{kl}) = -1\}.$$

Let

$$Y_Q \stackrel{\text{def}}{=} \sum_{v \in A_Q} \mathrm{NormDiff}(P, P + v, \vec{0}).$$

Then,

$$\mathrm{NormDiff}(P, Q, \vec{0}) = \frac{Y_Q - 2 \cdot \mathrm{GraphPL}(P, \vec{0})}{|A_Q|} + 4.$$

PROOF. Q $(= P + \lambda_{ij})$ $\in Obtuse(P, \vec{0}) \Rightarrow (((p_i = 0)$ and $(p_j < 0))$ or $((p_i > 0)$ and $(p_j = 0)))$.

We consider the two cases in the disjunction on the right hand side separately.

1. $(p_i = 0)$ and $(p_j < 0)$

   Then $((\lambda_{kl} \in A_Q) \Longleftrightarrow ((k = j)$ and $(l \neq i))$.

   Consider all points $S_v = P + \lambda_{jl_v}, \lambda_{jl_v} \in A_Q$.

$$\begin{aligned} \mathrm{NormDiff}(S_v, P, \vec{0}) &= \left[\sum_{t=0}^{n} {p_t}^2 - (\sum_{t=0}^{n} {p_t}^2 + 2p_j + 1 - 2p_{l_v} + 1)\right] \\ &= [2p_{l_v} - 2p_j - 2]. \end{aligned}$$

   Then,

$$Y_Q = \left[\sum_{\lambda_{jl_v} \in A_Q} \|P\|^2 - \|S_v\|^2\right] = 2 \cdot \left[\sum_{p_{l_v} > 0} p_{l_v} - |A_Q| \cdot p_j - |A_Q|\right].$$

   Therefore, $Y_Q = 2 \cdot (\mathrm{GraphPL}(P, \vec{0}) - |A_Q| \cdot p_j - |A_Q|)$.

$$\begin{aligned} \mathrm{NormDiff}(P, Q, \vec{0}) &= \left[\sum_{t=0}^{n} {p_t}^2 + 2p_i + 1 - 2p_j + 1 - \sum_{t=0}^{n} {p_t}^2\right] \\ &= [2(p_i - p_j + 1)] \\ &= 2(-p_j + 1). \end{aligned}$$

   Therefore,

$$\mathrm{NormDiff}(Q, P, \vec{0}) = \frac{Y_Q - 2 \cdot \mathrm{GraphPL}(P, \vec{0})}{|A_Q|} + 4.$$

2. $((p_i > 0)$ and $(p_j = 0))$
   Then, $(\lambda_{kl} \in A_Q) \iff ((k \neq j)$ and $(l = i))$.
   Consider all points $S_v = P + \lambda_{k_v i}, \lambda_{k_v i} \in A_Q$.

$$\text{NormDiff}(S_v, P, \vec{0}) = \left[\sum_{t=0}^{n} {p_t}^2 - (\sum_{t=0}^{n} {p_t}^2 + 2p_{k_v} + 1 - 2p_i + 1)\right] = 2p_i - 2p_{k_v} - 2.$$

Then,

$$Y_Q = \left[\sum_{\lambda_{k_v i} \in A_Q} \|P\|^2 - \|S_v\|^2\right] = 2 \cdot \left[|A_Q| \cdot p_i - \sum_{p_{k_v} < 0} p_{k_v} - |A_Q|\right].$$

Therefore, $Y_Q = 2 \cdot (|A_Q| \cdot p_i - (-\text{GraphPL}(P, \vec{0})) - |A_Q|)$.

$$\begin{aligned} \text{NormDiff}(Q, P, \vec{0}) &= \left[\sum_{t=0}^{n} {p_t}^2 + 2p_i + 1 - 2p_j + 1 - \sum_{t=0}^{n} {p_t}^2\right] \\ & 2(p_i - p_j + 1) = 2(p_i + 1). \end{aligned}$$

Therefore,

$$\text{NormDiff}(Q, P, \vec{0}) = \frac{Y - 2 \cdot \text{GraphPL}(P, \vec{0})}{|A_Q|} + 4.$$

□

7.2.2 Inputs

The *Initiator* needs to be activated.

7.2.3 Contents of Messages

Each processor, $\mathcal{M}(P)$, must get the following entities in its message, if any, from a neighbor in a basic direction D.

1. $(d(P, I))^2$,

2. NormDiff$(P, (P + D), I)$, and

3. GraphPL(P, I).

7.2.4 Computing *SendDirs*

In this section, we will first specify how each processor computes *SendDirs*. Then, we will justify this method of computing *SendDirs* by using Lemma 7.2.3, and show that Theorem 5.1.2 is true about this algorithm.

Each processor $\mathcal{M}(P)$ in $\mathcal{M}(A)$ computes *SendDirs* to be

$$\left[\bigcap_{D' \in InDirs} \{\ D'' \in \texttt{Directions} \mid (D', D'') \leq 0\ \}\right].$$

For each $P \in A$, define $t(P)$ (for Lemma 7.2.3) to be the unique integer t' so that

$$\forall t'' \in \mathbf{N}, |(t'/\texttt{SpaceDil}) - d(P, I))| \leq |(t''/\texttt{SpaceDil}) - d(P, I)|.$$

Then, it is easily seen that t satisfies all requirements on t specified in Lemma 7.2.3.

As seen in Lemmas 7.2.5, 7.2.6 and 7.2.7, *Msgs* from Lemma 7.2.3 can be computed in bounded time provided all messages from all inner neighbors arrive.

The algorithm for WAVE′ follows the scheme of Lemma 7.2.3.

Thus, all the hypotheses of Lemma 7.2.3 are satisfied, and so, for WAVE′,

> each processor $\mathcal{M}(P)$ in $\mathcal{M}(A)$ gets messages from all and only its inner neighbors simultaneously at time $t(P)$. The difference between $d(P, I)$ and $t(P)$ is bounded by $1/\texttt{SpaceDil}$.

The above conclusion about WAVE′ also guarantees that for any processor $\mathcal{M}(P)$, in $\mathcal{M}(A)$, all messages from all inner neighbors arrive, thus ensuring that *Msgs* can be computed in bounded time.

7.2.5 Computing *SendTime*

Recall, from Section 7.1.4, the computation of *SendTime* we presented for WAVE. Almost an identical computation is done here, in WAVE′. The only difference is in the method of computing the square of the euclidean distance of the neighbor to whom a message is to be sent. In WAVE, we used the relative coordinates, with respect to I, of the neighbor $P + D$ to compute its euclidean distance from I.

Here, we present *only* the computation of the square of the euclidean distance of a neighbor given by a direction in *SendDirs*. Then, the reader is referred to Section 7.1.4 for a procedure to compute the pass number in which to send a message to a particular neighbor.

Recall from Definition 7.2.2 and Lemma 7.2.4 actual computation that for a processor $\mathcal{M}(P)$, the sets $Opp(P, I), Rt(P, I)$ and $Obtuse(P, I)$ (suitably defined with the second parameter to be I instead of $\vec{0}$) are pairwise disjoint, and that their union is exactly *SendDirs* as computed above (Section 7.2.4). So, it suffices to be able to calculate the square of the euclidean distance of $P + D$, for D in one of Opp, Rt or $Obtuse$. We use expressions from Lemmas 7.2.5, 7.2.6 and 7.2.7 to calculate these quantities.

First $\mathcal{M}(P)$ computes the three sets outer neighbors $Opp(P, I), Rt(P, I)$, and $Obtuse(P, I)$. Suppose $Q = (P + D), D \in SendDirs$.

1. If $(P + D) \in Opp(P, I)$, then $\mathrm{NormDiff}(P, Q, I) = \mathrm{NormDiff}(P, (P - D), I) + 4$.

2. If $(P + D) \in Rt(P, I)$, then $\mathrm{NormDiff}(P, Q, I) = 2$.

3. If $(P + D) \in Obtuse(P, I)$, then
   let $O(D) = \{D' \in InDirs \mid (D, D') = -1\}$, and
   let $x_D = \sum_{d' \in O(D)} \mathrm{NormDiff}(P, (P + D')I)$. Then,

$$\mathrm{NormDiff}(P, Q, I) = 4 + \frac{x_D - 2 \cdot \mathrm{GraphPL}(P, I)}{|O(D)|}.$$

Then, for each neighbor, $P + D, D \in SendDirs$,

$$(d((P + D), I))^2 = (d(P, D))^2 + \mathrm{NormDiff}(P, (P + D), I).$$

Notice in each of the three cases above that the computations can be carried out *without* information about the coordinates of I, or the coordinates of P, absolute or relative to I.

As is evident from Section 7.2.3 above, $\mathrm{GraphPL}(P, I)$, and the quantities $\mathrm{NormDiff}(P, (P - D), I)$ (in case 1) and $\mathrm{NormDiff}(P, (P + D'), I)$ (in case 3, for each suitable D' as defined in case 3), are values that are received in the messages that $\mathcal{M}(P)$ gets. Clearly, then, the difference in the squares of euclidean distances, from I, of P and a neighbor $(P+D), D$ in *SendDirs*, can be computed without any reference to the coordinates of P and $(P+D)$ with respect to any particular basis. Also, the simulation time required to perform all the computations mentioned above is bounded and can be determined prior to executing WAVE$'$.

7.2.6 Computing *OutMessage*

As can be seen from the previous section, for each neighbor, $P+D, D \in$ *SendDirs*, the values of $(d((P+D),I))^2$ and NormDiff$(P,(P+D),I)$ are computed by $\mathcal{M}(P)$ during the computation of *SendTime*. These values are passed on in the message sent to $\mathcal{M}(P+D)$. Also, the graphical path length, with respect to $\mathcal{M}(I)$, of $\mathcal{M}(P+D)$, by choice of *SendDirs* is one more than that of $\mathcal{M}(P)$. $\mathcal{M}(P)$ receives its graphical path length in a message. So, $\mathcal{M}(P)$ can easily compute the graphical path length of $\mathcal{M}(P+D)$. This value is also sent in the message to $\mathcal{M}(P+D)$.

Chapter 8

Simulating Curve Traversal by a Particle

In this section, we describe our algorithm template, CURVE, a refinement of the more general algorithm template EuclideanMotion. CURVE is for simulating the motion of particles that trace certain curves in euclidean space. Then, in Chapters 9.2, and 9.3, we describe two specific algorithms, instances of CURVE, for the simulation of the translation, at virtually constant speed, of a particle, and for the simulation, at virtually constant angular speed, of the revolution of a particle about a fixed point in a given 2-dimensional plane of revolution.

For convenience we repeat Convention 4.1.1 as

Convention 8.0.2 *$\mathcal{L}$ denotes an arbitrary n-dimensional lattice, $n > 1$, which well-represents* itself *in $\mathbf{Z}^{n'}$, for $n' \geq n$. Each point (vector) in $\mathcal{L}$ can thus be represented as a unique n'-tuple of integers. We suppose $\mathbf{E}^n$ is the perfect representation of $\mathbf{R}^n$ that contains $\mathcal{L}$.*

Convention 8.0.3 *Henceforth, $L \subseteq \mathcal{L}$ and $\mathcal{M}(L)$ is a suitable mesh as witnessed by $\mathcal{L}$.*[1]

Suppose a particle starts at a point I in L, and traces, over a period of time, a curve in $\mathbf{E}^n$ in which the points (in $\mathbf{E}^n$) that the curve passes through are given by

[1] $L \subseteq \mathcal{L}$ satisfies Assumption 4.3.1 as witnessed by $\mathcal{L}$ and $\mathcal{M}(L)$ satisfies Assumptions 4.3.2–4.3.7.

$\rho(t)$, $0 \leq t \in \mathbf{R}$, where ρ is a differentiable function of real–time. We would like to simulate the motion of this particle in $\mathcal{M}(L)$. We will further assume that the particle moves along ρ at a constant speed.

Since we will be performing the simulation in a mesh consisting of discrete points, we need to define the representation, in the mesh, of the curve ρ. We would like each lattice point to represent a region of euclidean space around it. We will call this region of euclidean space around a lattice point P, the *region of influence of* P. We would, then, like to have a lattice point be a part of the representation of a curve iff the representation of that curve passes through the region of influence of that lattice point. A natural region of influence, of a lattice point P, that one might choose is the set of points x, in $\mathbf{E}^n$ so that x is closer, or as close, to P than every other lattice point. Recall, from Section 3.4 the definition of Voronoi cells around lattice points. Then, the region of influence of a lattice point we choose is just the closure of the Voronoi cell around that lattice point.

All our simulations must progress in discrete steps.

In this section we use s as a parameter for the length of the tiniest arc length of curve represented, where $0 < s$. For the particular instances of CURVE, TRANSLATE and REVOLVE, we take $s = (1/\texttt{SpaceDil})$. Below (Propositions 8.1.1 and 8.1.2), we obtain some fundamental constants[2] for $0 < s < (\mathrm{MinLgth}(\mathcal{L}))/2$ in the cases where $\mathcal{L} \in \{\mathbf{Z}^n, A_n\}$.

Given a curve ρ, we approximate ρ by the sequence of points, C, on ρ, so that $\rho(0) \in C$ and the length of the arc segment of ρ between any two consecutive points, along ρ, in C, is s. Then, we treat ρ as "equivalent" to the curve, ρ_s, which is made up (for each k, a non-negative integer) of straight line segments between consecutive points, $\rho_s(k)$ and $\rho_s(k+1)$ in C.[3] Our algorithms, actually, will simulate the motion of a particle, whose motion is given by ρ, as if the particle were moving along ρ_s, but with the discrete timing of moving along equal arc length segments of ρ. (In particular, for our present version of REVOLVE, while the angular speed is virtually constant, the circles traveled are approximated by representations of *inscribed* polygons with

[2]These constants are the m's from Remark 6.4.2.

[3]We assume that, for all $k \geq 0$ $\rho_s(k+1) \neq \rho_s(k)$.

a very large number of sides. In Chapter 11 we mention the interesting problem of developing practical variants of REVOLVE based on alternative representations of circles.)

Remark 8.0.1 *We adopt as our (static) representation of ρ_s in $\mathcal{M}(L)$ the set*

$$\{X \in L \mid \exists x \in \mathbf{E}^n, x \in \mathit{range}(\rho_s) \text{ and } x \in \overline{\mathrm{Voronoi}(X, \mathcal{L})}\}.$$

Assumption 8.0.1 *We will suppose that for* CURVE *the Initiator $\mathcal{M}(I)$ is such that $\rho(0) = I$.*

8.1 Further Temporal Dilation: $\mathrm{MPL}_s(\mathcal{L})$

The progress along a curve in $\mathcal{M}(L)$ is achieved in the simulation by the passage of messages from processor to neighboring processor. It may be the case that at simulation time t_s the curve is in the Voronoi cell of the lattice point P, and at simulation time t_s+1, the curve is in the Voronoi cell of a lattice point which is *not* a neighbor of P. Also, by Lemma 3.4.11, every segment $[\rho_s(k), \rho_s(k+1))$ must intersect but a finite number of Voronoi cells around lattice points.

Definition 8.1.1 *Suppose the non-trivial straight line segment $[a, b] \subseteq \mathbf{E}^n$.*

1. *A sequence of points from $\mathcal{L}$, $\langle P_1, \ldots, P_k \rangle$, is a* covering path for $[a, b]$ in $\mathcal{L}$ *(written:* $\mathrm{Cov}(\langle P_1, \ldots, P_k \rangle, [a, b], \mathcal{L})$*) iff*

   (a) $(\forall i, 1 \leq i < k)[P_i$ *is a neighbor of* $P_{i+1}]$,

   (b) $(a \in \overline{\mathrm{Voronoi}(P_1, \mathcal{L})}$ *and* $b \in \overline{\mathrm{Voronoi}(P_k, \mathcal{L})})$,

   (c) $(\forall i, 1 \leq i \leq k)[([a, b] \cap \overline{\mathrm{Voronoi}(P_i, \mathcal{L})}) \neq \emptyset)$, *and*

   (d) $[a, b] \subseteq \cup_{1=1}^{k} \overline{\mathrm{Voronoi}(P_i, \mathcal{L})}$.

2. *The length of such a covering path $\langle P_1, \ldots, P_k \rangle$ is (by definition) k and is denoted by* $\mathrm{CPL}(\langle P_1, \ldots, P_k \rangle)$.

3. $\mathrm{MinCov}([a, b], \mathcal{L}) \stackrel{\mathrm{def}}{=} \min\{\mathrm{CPL}(\langle P_1, \ldots, P_k \rangle) \mid \mathrm{Cov}(\langle P_1, \ldots, P_k \rangle, [a, b], \mathcal{L})\}$.

4. $\mathrm{MPL}_s(\mathcal{L}) \stackrel{\mathrm{def}}{=}$

$$\max\{\mathrm{MinCov}([a', b'], \mathcal{L}) \mid [a', b'] \textit{ is a non-trivial line segment in } \mathbf{E}^n \textit{ and } d(a', b') \leq s\}.$$

In this section we use s as a parameter representing the euclidean distance, in $\mathcal{M}(L)$, traversed in $m = \mathrm{MPL}_s(\mathcal{L})$ passes of the algorithm template CURVE.

The speed of the simulated particle, though, would be slower by a factor of, at least $\mathrm{MPL}_s(\mathcal{L})$.

It may be difficult to calculate the value of $\mathrm{MPL}_s(\mathcal{L})$, for arbitrary well-represented lattices $\mathcal{L}$. It is likely possible, though not easy, to calculate $\mathrm{MPL}_s(\mathcal{L})$ for $\mathcal{L}$ in certain families of lattices (e.g., root lattices [CS88]) since, in these cases, there exist (often tedious) algorithms to generate the set, $V(\mathcal{L})$, of vertices of Voronoi cells around the origin in $\mathcal{L}$. Nonetheless, we can calculate $\mathrm{MPL}_s(\mathbf{Z}^n)$ and $\mathrm{MPL}_s(A_n)$ rather easily.

Clearly,

$$V(\mathbf{Z}^n) = \{(z_1, \ldots, z_n) \mid \forall i, 1 \leq i \leq n, z_i = \pm 1/2\}.$$

From [CS88] the set $V(A_n)$ is given by

$$\begin{aligned} V(A_n) \quad = \quad & \{(s_0, \ldots, s_n) \mid \exists B \subset \{0, \ldots, n\}, \\ & \quad (B \neq \emptyset) \;\&\; (\forall i \in B, s_i = (n+1-|B|)/(n+1)) \\ & \quad\quad \&\; (\forall i \in (\{0, \ldots, n\} \setminus B), s_i = -|B|/(n+1))\}. \end{aligned}$$

(See [CS88] for additional details about Voronoi cells around points in $\mathbf{Z}^n$ and A_n.)

In Propositions 8.1.1 and 8.1.2 below, we give the values for $\mathrm{MPL}(\mathbf{Z}^n)$ and $\mathrm{MPL}(A_n)$, respectively. In the proofs, $2\mathrm{Voronoi}(\vec{0}, \mathcal{L})$ will denote the magnification of the Voronoi cell around the origin, in $\mathcal{L}$, by a factor of 2.

Proposition 8.1.1 $\forall s < 1/2, \mathrm{MPL}_s(\mathbf{Z}^n) = (n+1)$.

PROOF. For all $s < 1/2$, every point within s units of $\mathrm{Voronoi}(\vec{0}, \mathbf{Z}^n)$ is contained in $2\mathrm{Voronoi}(\vec{0}, \mathbf{Z}^n)$.

Now consider, C^n, the union of the standard lattice cubes, each of side 1, and each with the origin of $\mathbf{Z}^n$ as one of its vertices. Then, clearly, $2\mathrm{Voronoi}(\vec{0}, \mathbf{Z}^n)$ is contained in C^n.

Therefore, for any segment $[a,b]$ of euclidean length $< 1/2$, so that $a,b \in \mathbf{R}^n$ and $a \in \mathrm{Voronoi}(\vec{0}, \mathbf{Z}^n)$,

$$\mathrm{MinCov}([a,b], \mathbf{Z}^n) \leq (n+1).$$

Furthermore, consider a segment $[a', b']$ so that its length is less than $1/2$, $a' \in \mathrm{Voronoi}(\vec{0}, \mathbf{Z}^n)$, and the segment contains the point of intersection of the Voronoi cell around the origin and around the point whose co-ordinates are all 1's. Clearly,

$$\mathrm{MinCov}([a', b'], \mathbf{Z}^n) = (n+1).$$

Therefore,

$$\forall s < 1/2, \mathrm{MPL}_s(\mathbf{Z}^n) = (n+1).$$

$\square$

Recall that

$$A_n = \{P(= (p_0, p_1, \ldots, p_n)) \in \mathbf{Z}^{n+1} \mid \sum_{i=0}^{n} p_i = 0\}.$$

Lemma 8.1.1 *Suppose $x \in 2\mathrm{Voronoi}(\vec{0}, A_n)$ and $x \in \overline{\mathrm{Voronoi}(P, A_n)}$, where $P = (p_0, p_1, \ldots, p_n)$. Then, $\forall i, 0 \leq i \leq n, |p_i| \leq 1$.*

Proof. Since $x \in 2\mathrm{Voronoi}(\vec{0}, A_n)$, and $\mathrm{MinLgth}(A_n) = \sqrt{2}$,

$$\forall \lambda_{ij}, -2 < (x, \lambda_{ij}) < 2. \tag{44}$$

Since $x \in \overline{\mathrm{Voronoi}(P, A_n)}$, and $\mathrm{MinLgth}(A_n) = \sqrt{2}$,

$$\forall \lambda_{ij}, -1 \leq ((x-P), \lambda_{ij}) = (x, \lambda_{ij}) - (P, \lambda_{ij}) \leq 1.$$

Therefore,

$$\forall \lambda_{ij}, -1 - (x, \lambda_{ij}) \leq -(P, \lambda_{ij}) \leq 1 - (x, \lambda_{ij}). \tag{45}$$

Combining (44) and (45), we have

$$\forall \lambda_{ij}, -3 < (P, \lambda_{ij}) < 3. \tag{46}$$

Now, suppose by way of contradiction, there is an $i, 0 \leq i \leq n$, so that, say, $p_i \geq 2$. Then, by Lemma 7.2.1, there is a $j \neq i, 0 \leq j \leq n$, so that $p_j < 0$. Since P is represented as an $(n+1)$-tuple of integers, p_i and p_j must be integers. Therefore, $p_j \leq -1$. Then, clearly, $(P, \lambda_{ij}) \geq 3$, giving us a contradiction to (46) above. □

Proposition 8.1.2 $\forall s < \mathrm{MinLgth}(A_n)/2, \mathrm{MPL}_s(A_n) = \lfloor\frac{n+1}{2}\rfloor + 1.$

PROOF. For all $s < \mathrm{MinLgth}(A_n)/2$, every point within s units of Voronoi$(\vec{0}, A_n)$ is contained in 2Voronoi$(\vec{0}, A_n)$. Then, by Lemma 8.1.1, the only lattice points, $P = (p_0, \ldots, p_n)$, such that the Voronoi cell around P has a non-empty intersection with 2Voronoi$(\vec{0}, A_n)$, must be so that $\forall i, 0 \leq i \leq n, |p_i| \leq 1$. Then, the farthest, amongst these points, with respect to graphical path length, from $\vec{0}$ must be the one with the maximum number of $+1$'s and the maximum number of -1's. But, by definition of A_n, since the sum of the co-ordinates of P is 0, the number of 1's and -1's must be equal. Therefore, the farthest, with respect to graphical path length, point from $\vec{0}$ must have $\lfloor\frac{n+1}{2}\rfloor$ 1's and an equal number of -1's as its co-ordinates. Therefore, for any segment $[a, b]$ in the n-dimensional euclidean space containing A_n so that the euclidean length of $[a, b]$ is less than MinLgth$(A_n)/2$ and $a \in$ Voronoi$(\vec{0}, A_n)$,

$$\mathrm{MinCov}([a, b], A_n) \leq \lfloor\frac{n+1}{2}\rfloor + 1.$$

Consider the segment $[a', b']$ so that $a' \in$ Voronoi$(\vec{0}, A_n)$ and so that the segment contains the point of intersection of the closures of the Voronoi cells around the origin, and a point in A_n having $\lfloor\frac{n+1}{2}\rfloor$ 1's and an equal number of -1's as its co-ordinates. Then,

$$\mathrm{MinCov}([a', b'], A_n) = \lfloor\frac{n+1}{2}\rfloor + 1.$$

Therefore,

$$\forall s < \mathrm{MinLgth}(A_n)/2, \mathrm{MPL}_s(A_n) = \lfloor\frac{n+1}{2}\rfloor + 1.$$

□

8.2 Algorithm CURVE

The algorithm template for simulating the motion of a particle, tracing a curve in euclidean n-space is itself a refinement of the template, EuclideanMotion, described in Section 6.4 of our algorithms. We will assume all the constants, types, and variables already defined in Section 6.4 in the description of Algorithm CURVE below. In addition, algorithm CURVE requires, for its execution, the following constants and variables.

8.2.1 Universal Constant

const `PathLgthDil`;

This is the extra temporal dilation factor $\mathrm{MPL}_s(\mathcal{L})$. introduced in Section 8.1 above.

8.2.2 Variables

var *leftend*;

This variable is initialized externally for the *Initiator*, $\mathcal{M}(I)$, to 0. When, if ever, a processor, $\mathcal{M}(P)$, receives a message from another processor, $\mathcal{M}(P)$ initializes the value of this variable from an appropriate field in that message. If the value received is k, this means that the particle being simulated is on the line segment $[\rho_s(k), \rho_s(k+1))$. $\mathcal{M}(P)$ can use and set *leftend*.

var t_{in};

This variable is initialized externally for the *Initiator*, $\mathcal{M}(I)$, to 0. When, if ever, a processor, $\mathcal{M}(P)$, receives a message, $\mathcal{M}(P)$ initializes the value of this variable from an appropriate field in that message. If the value received is k, this means that this message was sent, to $\mathcal{M}(P)$, at the end of k passes. $\mathcal{M}(P)$ can use and set t_{in}.

We describe algorithm CURVE as if it were being executed by processor $\mathcal{M}(P)$.

8.2.3 Executable Part of Algorithm CURVE

begin

1. **if** `Done` **then goto** step 3;
   else
   if (**not** REPR) **then**
   Initialize Parameters using external input, if any, and
   $\{ \; Buf\langle D\rangle.InMessage | D \in InDirs \; \}$, if non-empty;
   endif;
   endif;
2. **if** $[\rho_s(leftend), \rho_s(leftend+1)) \cap \overline{\text{Voronoi}(P,\mathcal{L})} = \emptyset$ **then**
   REPR := **false**;
   Done := **true**;
   goto step 3;
   else
   REPR := **true**;
   $SendDirs := \emptyset$;
   if not $(\rho_s(leftend+1) \in \text{Voronoi}(P,\mathcal{L}))$ **then**
   $SendDirs := \{D | ((\rho_s(leftend+1) - P), D) \geq \text{MinLgth}(\mathcal{L})^2/2\}$;
   if $(\rho_s(leftend+1) \in \overline{\text{Voronoi}(P,\mathcal{L})})$ **then**
   for each $D \in SendDirs$,
   $Buf\langle D\rangle.OutMessage.t_{in} :=$
   `PathLgthDil` $\cdot \, (leftend+1)$;
   $Buf\langle D\rangle.OutMessage.leftend := leftend + 1$;
   endfor;
   else
   for each $D \in SendDirs$,
   $Buf\langle D\rangle.OutMessage.t_{in} := t_{in} + 1$;
   $Buf\langle D\rangle.OutMessage.leftend := leftend$;
   endfor;
   endif;
   endif;
   endif;
   $t_{in} := t_{in} + 1$;
   $leftend := (t_{in}$ **div** `PathLgthDil`);

$SendThisPass$:=
$\{D \in SendDirs \mid Buf\langle D\rangle.OutMessage.t_{in} = \texttt{PassCount}\}$;

3. **idle till** $\texttt{SimClock} = (\texttt{CompLen} \cdot \texttt{PassCount}) - 1$;
   simul send
   $Buf\langle D\rangle.OutMessage$ in direction D, for each $D \in SendThisPass$;
   endsend;

end.

Theorem 8.2.1, below, expresses the correctness of CURVE under the assumption that the values of all boolean expressions and all declared variables can be computed exactly.

To specify the Chapter 9 instances of CURVE, it turns out to suffice to further specify:

1. additional contents of messages, and
2. how to perform the tests in step 2.

In Section 8.2.4, we elaborate on the tests that need to be performed in step 2 in algorithm CURVE, and show that all these tests involve rational expressions in $\rho_s(k)$ and $\rho_s(k+1)$. Later, in Sections 9.2 and 9.3, we will give the exact expressions for $\rho_s(k)$, for the specific instances of CURVE. There, we will also show that, in these specific instances, the values of all boolean expressions and declared variables can be computed exactly using integers only.

We note the following facts about the above algorithm, and introduce some notation useful in proving the correctness of the algorithm.

1. The Initiator executes step 2 because it's variable `Done` is initially set to **false**from the outside.
2. A processor, $\mathcal{M}(P)$, other than the Initiator, starts executing step 2 of the algorithm if and only if it gets a message from some neighbor.
3. A message contains at least the two fields, t_{in}, and *leftend*.

4. If during a simulation time[4] interval $(t_s, t_s + 1]$, REPR is **true**, in the copy of the program (encoding Algorithm CURVE) being executed by $\mathcal{M}(P)$, then we say that $\mathcal{M}(P)$ is in the state REPR at simulation time $(t_s + 1)$.

5. $REPR(P)$ is the set of all simulation time units at which $\mathcal{M}(P)$ is in the state REPR.

6. We say that $\mathcal{M}(P)$ *is executing the k-th segment passes* when and only when it is executing a pass in which *leftend* $= k$, and before the end of that pass, $\mathcal{M}(P)$ is in state REPR.

7. We will refer to the beginning of the first pass in the execution, by $\mathcal{M}(P)$, of the k-th segment passes of the algorithm, as the *beginning of the kth segment passes* for $\mathcal{M}(P)$.

8. If during a pass in the execution of the k-th segment passes of the algorithm by $\mathcal{M}(P)$, REPR is **true**, then that pass is the first pass, executed by $\mathcal{M}(P)$ in the k-th segment passes.

9. If $\mathcal{M}(P)$ executes the k-th segment passes of the algorithm, then $[\rho_s(k), \rho_s(k + 1))$ has a non-empty intersection with the Voronoi cell around P.

10. We will refer to the value of t_{in} in $\mathcal{M}(P)$ at the beginning of the k-th segment passes for $\mathcal{M}(P)$, by $t^k_{in}(P)$.

11. If, in the k-th segment passes, $\mathcal{M}(P)$ computes *SendDirs* to be a non-empty set, then the common value to which $Buf\langle D\rangle.OutMessage.t_{in}$ is set, for each D in *SendDirs* will be referred to as $t^k_{out}(P)$.

12. In the k-th segment passes, *SendDirs* computed by $\mathcal{M}(P)$ is non-empty iff $\rho_s(k+1) \notin \text{Voronoi}(P, \mathcal{L})$, as implied by the computation of *SendDirs* in the algorithm and Lemma 4.2.1. Furthermore, if *SendDirs* is non-empty, then $\mathcal{M}(P)$

[4] Recall that basic simulation time is measured in instruction cycles.

sends messages to all those neighbors whose Voronoi cells might possibly contain $[\rho_s(k), \rho_s(k+1))$, again as implied by the computation of *SendDirs* in the algorithm, and Lemma 4.2.1.

We now prove a few lemmas, which we use in proving Theorem 8.2.1. Theorem 8.2.1 characterizes the integer time interval during which each processor, $\mathcal{M}(P)$, in the static representation of ρ, is in the state REPR.

Notation 8.2.1 *For notational convenience, we will abbreviate* `PathLgthDil` *by PL, and* `CompLen` *by T. For natural numbers $a < b$, $(a, b]$ will denote the set $\{x \in \mathbf{N} | a < x \leq b\}$.*[5] *For $S \subseteq \mathbf{N}$, and natural numbers $a < b$, $(a, b] \sqsubseteq S$ will denote the statement $(\exists y \in \mathbf{N})(\forall x \in \mathbf{N}, a < y \leq x \leq b)[x \in S]$.*[6] *Note that, if $(a, b] \subseteq S$ then $(a, b] \sqsubseteq S$.*

Lemma 8.2.1 *Suppose $\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})}$ and $\rho_s(k+1) \in \overline{\text{Voronoi}(Q, \mathcal{L})}$. Suppose $REPR(P) \neq \emptyset$. Then, $t_{in}^k(Q) \leq t_{in}^k(P) + PL$.*

Proof. If $\rho_s(k+1) \in \text{Boundary}(\text{Voronoi}(P, \mathcal{L}))$, then the algorithm sets the value of t_{in} to be sent to $\mathcal{M}(Q)$ to $PL \cdot (k+1)$. Hence, the largest difference between $t_{in}^k(Q)$ and $t_{in}^k(P)$ is PL.

Otherwise, $\rho_s(k+1) \notin \overline{\text{Voronoi}(P, \mathcal{L})}$. Therefore, by Lemma 4.2.3, there exists a finite sequence of points $P = P_1, P_2, \ldots, P_l = Q$ satisfying the conditions in Lemma 4.2.3 above. Furthermore, the choice of the length of segments in ρ_s guarantees that each segment passes through at most PL Voronoi cells. Therefore, $(l-1) < PL$. Then, each of $\mathcal{M}(P_i), 1 \leq i < l$ computes the value of t_{in} to be sent to $\mathcal{M}(P_{i+1})$ to be $t_{in}^k(P_i)+1$. Therefore, $t_{in}^k(Q) = t_{in}^k(P_l) = t_{in}^k(P)+(l-1) < t_{in}^k(P)+PL$. □

Lemma 8.2.2 *Suppose $[\rho_s(k), \rho_s(k+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset$. Then,*

1. *$[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset \Longrightarrow (T \cdot PL \cdot k, T \cdot PL \cdot (k+1)] \subseteq REPR(P)$, and*

---
[5] Context will resolve any ambiguity as to whether intervals are real or integer.

[6] Of course $(a, b] \sqsubseteq S$ is mathematically equivalent to $b \in S$.

2. $[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} = \emptyset \Longrightarrow (T \cdot PL \cdot k, T \cdot PL \cdot (k+1)] \sqsubseteq REPR(P)$.

Proof. We prove the lemma by proving, by induction on k, the following four statements.

1. $(\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})}$ and $[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset) \Longrightarrow (T \cdot PL \cdot k, T \cdot PL \cdot (k+1)] \subseteq REPR(P)$,

2. $(\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})}$ and $[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} = \emptyset) \Longrightarrow (T \cdot PL \cdot k, T \cdot PL \cdot (k+1)] \sqsubseteq REPR(P)$,

3. $(\rho_s(k) \notin \overline{\text{Voronoi}(P, \mathcal{L})}$ and $[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset) \Longrightarrow (T \cdot PL \cdot k, T \cdot PL \cdot (k+1)] \subseteq REPR(P)$, and

4. $(\rho_s(k) \notin \overline{\text{Voronoi}(P, \mathcal{L})}$ and $[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} = \emptyset) \Longrightarrow (T \cdot PL \cdot k, T \cdot PL \cdot (k+1)] \sqsubseteq REPR(P)$.

We start the induction at $k = 1$. We assume that $s < \text{MinLgth}(\mathcal{L})/2$ and $\rho_s(0) = I$. Therefore, $\rho_s(1) \in \text{Voronoi}(I, \mathcal{L})$, and so I satisfies the hypothesis of this lemma. Furthermore, $\rho_s(0) \in \overline{\text{Voronoi}(I, \mathcal{L})}$. Therefore, statements 2, 3 and 4 of this lemma are vacuously true for $P = I$.
Since, $[\rho_s(0), \rho_s(1)) \cap \overline{\text{Voronoi}(I, \mathcal{L})} \neq \emptyset$, $\mathcal{M}(I)$ executes the 0-th segment passes of the algorithm, and since the hypothesis of this lemma is true for $P = I$, $\mathcal{M}(I)$ executes the 1-th segment passes of the algorithm. Therefore, $(T \cdot PL \cdot 1, T \cdot PL \cdot 2] \subseteq \text{REPR}(I)$, and so statement 1 of this lemma is true for $P = I$.
Suppose $P \neq I$ satisfies the hypothesis of this lemma for $k = 1$. Since $\rho_s(1) \in \text{Voronoi}(I, \mathcal{L})$, statements 1, 2 and 3 are vacuously true for P. Also, since

1. the hypothesis of this lemma is true for P and $k = 1$,

2. each segment of ρ_s is a straight line segment, and

3. Voronoi cells are convex sets,

$\rho_s(2) \notin \overline{\text{Voronoi}(I, \mathcal{L})}$. Then, by the algorithm, $\mathcal{M}(I)$ computes the value of $t^1_{out}(I)$ as $t^1_{in}(I) + 1$. Also, by the algorithm, $t^1_{in}(I) = PL \cdot 1$. Lemma 4.2.3 applies, and we have a finite path from I to P of length less than or equal to PL. Then, by the second case in the proof of Lemma 8.2.1, $\mathcal{M}(P)$ receives a message at time $\leq T \cdot (PL + (PL-1))$, and then goes to the state REPR in less than T units of simulation time after having got the message. Also, by the algorithm, $\mathcal{M}(P)$ stays in the state REPR at least till simulation time $T \cdot PL \cdot 2$. Therefore, $(T \cdot PL \cdot 1, T \cdot PL \cdot 2] \sqsubseteq \text{REPR}(P)$.

Now, suppose this lemma is true for all values of $k < l$. We then show that this lemma is true for $k = l$.
Suppose P satisfies the hypothesis of this lemma for $k = l$.

Statements 1 and 3 Since $[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset$, $\mathcal{M}(P)$ executes the $(k-1)$-th segment passes of the algorithm, and then the k-th segment passes of the algorithm. Therefore, $\mathcal{M}(P)$ is is state REPR at simulation time $T \cdot PL \cdot k$, and stays in the state REPR through simulation time $T \cdot PL \cdot (k+1)$. Therefore statements 1 and 3 of this lemma are true.

Statement 2 Since $\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})}$ and $[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} = \emptyset$, $\rho_s(k) \in \text{Boundary}(\text{Voronoi}(P, \mathcal{L}))$, and (since each segment of ρ_s is a straight line segment, and Voronoi cells are convex bodies) $\rho_s(k-1) \notin \overline{\text{Voronoi}(P, \mathcal{L})}$. Lemma 4.2.3 applies, and there is a Q, so that $\rho_s(k-1) \in \overline{\text{Voronoi}(Q, \mathcal{L})}$, and we have a finite path from Q to P of length less than or equal to PL. Then, by induction, and (proof of) Lemma 8.2.1, $\mathcal{M}(P)$ receives a message at time $\leq T \cdot (PL \cdot k + (PL-1))$, and then goes to the state REPR in less than T units of simulation time after having got the message. Also, by the algorithm, $\mathcal{M}(P)$ stays in the state REPR at least till simulation time $T \cdot PL \cdot (k+1)$. Therefore, $(T \cdot PL \cdot k, T \cdot PL \cdot (k+1)] \sqsubseteq \text{REPR}(P)$. Hence, statement 2 of this lemma.

Statement 4 Suppose $[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} = \emptyset$. Then, by Lemma 4.2.1 and the construction of *SendDirs* in the algorithm, $\mathcal{M}(P)$ must have got a message from some neighbor $\mathcal{M}(Q)$, in which message the value of *leftend* was k. Since $\mathcal{M}(Q)$ sent a message, it must have executed one of the **for each** loops in step 2 of the algorithm.

Suppose $\mathcal{M}(Q)$ executed the first **for each** loop. Then, $\mathcal{M}(Q)$ must have sent the message during the $(k-1)$-th segment passes. Then, by the algorithm, $\rho_s(k) \in \text{Boundary}(\text{Voronoi}(Q, \mathcal{L}))$. Then, since, $(P - Q)$ is a direction in *SendDirs* computed by $\mathcal{M}(Q)$, as a consequence of Proposition 4.2.2, $\rho_s(k) \in \text{Boundary}(\text{Voronoi}(P, \mathcal{L}))$. This contradicts the hypothesis of this statement, i.e., $\rho_s(k) \notin \overline{\text{Voronoi}(P, \mathcal{L})}$.

Therefore, $\mathcal{M}(Q)$ must have executed the second **for each** loop in step 2 of the algorithm, and sent the message in the k-th segment passes. Since, the hypothesis of this lemma is true, $\rho_s(k+1) \notin \overline{\text{Voronoi}(Q, \mathcal{L})}$. Then, the value of t_{in}, computed by $\mathcal{M}(Q)$, to be sent to $\mathcal{M}(P)$ is $t_{in}^k(Q) + 1$.

Case 1 Suppose $\rho_s(k) \in \overline{\text{Voronoi}(Q, \mathcal{L})}$, then, by the algorithm, $t_{in}^k(Q) = PL \cdot k$.

Case 2 Suppose $\rho_s(k) \notin \overline{\text{Voronoi}(Q, \mathcal{L})}$. Then, Lemma 4.2.3 applies, and there is a Q', so that $\rho_s(k) \in \overline{\text{Voronoi}(Q', \mathcal{L})}$, and we have a finite path from Q' to P of length less than or equal to PL. Then, by induction, and (the proof of) Lemma 8.2.1, $\mathcal{M}(Q)$ receives a message at time $\leq T \cdot (PL \cdot k + (PL - 2))$, and hence $t_{in}^k(Q) \leq PL \cdot (k+1) - 2$.

In either case, $\mathcal{M}(Q)$ sends a message to $\mathcal{M}(P)$ at simulation time x, where $T \cdot PL \cdot k < x < T \cdot PL \cdot (k+1)$. Therefore, $\mathcal{M}(P)$ goes to the state REPR after simulation time $T \cdot PL \cdot k$ and before simulation time $T \cdot PL \cdot (k+1)$. Furthermore, by the algorithm, $\mathcal{M}(P)$ stays in the state REPR through the simulation time $T \cdot PL \cdot (k+1)$. Hence, statement 4 of this lemma is true.

□

Lemma 8.2.3 *Suppose* $(T \cdot PL \cdot k, T \cdot PL \cdot (k+1)] \sqsubseteq REPR(P)$. *Then,*

$$[\rho_s(k), \rho_s(k+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset.$$

PROOF. The hypothesis of this lemma is true for $k = 0$ if and only if $P = I$. So is the conclusion of this lemma.

For all $t > 0$, if $\mathcal{M}(P)$ is in state REPR at simulation time t it must have received a message (by external input, or from a neighbor) at simulation time $t' < t$. Suppose $\mathcal{M}(P)$ is in state REPR at time $t, T \cdot PL \cdot k < t \leq T \cdot PL \cdot (k+1)$. Suppose $\mathcal{M}(P)$ received a message at time $t' < t$.

Case 1: $t' \leq T \cdot PL \cdot k$ Then, by the induction hypothesis, $[\rho_s(k-1), \rho_k)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset$. By the algorithm, if $\mathcal{M}(P)$ is in state REPR through simulation time $T \cdot PL \cdot (k+1)$, then, $\mathcal{M}(P)$ must have executed the k-th segment passes, and thus the statement of this lemma is true.

Case 2: $t' > T \cdot PL \cdot k$ Then, the value of t_{in}, to be sent in a message, computed by the neighbor, $\mathcal{M}(Q)$, that sent the message must be greater than or equal to $PL \cdot k$ and less than $PL \cdot (k+1)$. Therefore, by the algorithm, $[\rho_s(k-1), \rho_s(k)) \cap \overline{\text{Voronoi}(Q, \mathcal{L})} \neq \emptyset$, and $\rho_s(k) \notin \text{Voronoi}(P, \mathcal{L})$. Furthermore, since $\mathcal{M}(P)$ stays in the state REPR through $T \cdot PL \cdot (k+1)$, $\mathcal{M}(P)$ must have executed the k-th segment passes. Therefore, the statement of this lemma is true.

□

Now, we have the following theorem, which characterizes the simulation times at which processors in the static representation of ρ are in the state REPR, as a direct consequence of the above two lemmas.

Theorem 8.2.1 $[\rho_s(k), \rho_s(k+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset \Longleftrightarrow (T \cdot PL \cdot k, T \cdot PL \cdot (k+1)] \sqsubseteq$ *REPR*(P).

□

Corollary 8.2.1 *All and only those processors, $\mathcal{M}(P)$, so that P is in the static representation of the curve ρ are in the state REPR at some simulation time.*

□

8.2.4 Elaboration of steps in Algorithm CURVE

As mentioned in Section 8.2.3, the only step that needs elaboration is step 2. To do this, it is enough to elaborate upon:

1. performing the check $[\rho_s(k), \rho_s(k+1)) \cap \overline{\mathrm{Voronoi}(P, \mathcal{L})} \neq \emptyset$,
2. performing the check $\rho_s(k) \in \mathrm{Voronoi}(P, \mathcal{L})$,
3. performing the check $\rho_s(k) \in \overline{\mathrm{Voronoi}(P, \mathcal{L})}$, and
4. computing *SendDirs*.

We assume here that the computations are being performed by processor $\mathcal{M}(P)$.

Performing the check, $[\rho_s(k), \rho_s(k+1)) \cap \overline{\mathrm{Voronoi}(P, \mathcal{L})} \neq \emptyset$, in step 2

We assume that $\rho_s(k)$ can be computed for any $k \in \mathbf{N}$. Notice that the verification of inequalities (17) and (18), in Proposition 4.2.1, involves evaluating expressions that are rational in $\rho_s(k)$ and $\rho_s(k+1)$ when these are substituted in for x_1 and x_2, respectively.

In the instances, described in Chapter 9, of algorithm CURVE we will see that this check can be performed using only integer and integer vector arithmetic, although $\rho_s(k)$ itself cannot be exactly computed for most k. What will be important is that the expressions involved in the verification are rational in $\rho_s(k)$, and that $\rho_s(k)^2$ is an integer.

Performing the checks $\rho_s(k) \in \mathrm{Voronoi}(P, \mathcal{L})$ and $\rho_s(k) \in \overline{\mathrm{Voronoi}(P, \mathcal{L})}$.

By Lemma 4.2.1,

$$\rho_s(k) \in \mathrm{Voronoi}(P, \mathcal{L}) \Longleftrightarrow \forall D \in \mathrm{Dirs}(\mathcal{L}), |((\rho_s(k) - P), D)| < \mathrm{MinLgth}(\mathcal{L})^2/2, \text{ and}$$

$$\rho_s(k) \in \overline{\mathrm{Voronoi}(P, \mathcal{L})} \Longleftrightarrow \forall D \in \mathrm{Dirs}(\mathcal{L}), |((\rho_s(k) - P), D)| \leq \mathrm{MinLgth}(\mathcal{L})^2/2.$$

Again, notice that the verification of each inequality for a direction D involves evaluating expressions that are rational in $\rho_s(k)$.

Computing *SendDirs*

In algorithm CURVE

$$\begin{aligned} \mathit{SendDirs} \quad := \quad & \{D \in \mathrm{Dirs}(\mathcal{L}) \mid ((\rho_s(k) - P), D) \geq \mathrm{MinLgth}(\mathcal{L})^2/2\}, \\ & \text{for some value of } k. \end{aligned}$$

Clearly, for each D, the evaluation of the condition for the inclusion of D in *SendDirs* involves expressions that are rational in $\rho_s(k)$.

Inspection of the expressions involved in performing the checks $\rho_s(k) \in \mathrm{Voronoi}(P, \mathcal{L})$, and $\rho_s(k) \in \overline{\mathrm{Voronoi}(P, \mathcal{L})}$ shows that, if these checks can be performed using integers only, then so can *SendDirs*. So, we will *not* elaborate on the computation of *SendDirs* in the specific instances, TRANSLATE and REVOLVE, of algorithm CURVE.

Chapter 9

Simulating Translation and Revolution

In this section, we describe two specific instances, algorithms TRANSLATE (Section 9.2) and REVOLVE (Section 9.3), of algorithm template CURVE described in Chapter 8. Here, our description of these algorithms will consist of showing that each of the steps of algorithm CURVE can be performed, in each of the two cases TRANSLATE and REVOLVE, using integer and integer vector arithmetic only. In Chapter 10, we will present complete algorithms, for TRANSLATE and REVOLVE, in a Pascal-like specification language. There, we will freely use the equivalences in expressions shown in this section.

9.1 An Elementary Fact

In this section we state and prove an elementary fact which we will often use in showing that the computations that need to be performed in algorithms TRANSLATE and REVOLVE below, can be performed using integers only.

Lemma 9.1.1 *Let q_1, q_2, q_3 and q_4 be rational numbers. Let*

$$a = \sqrt{q_1} \cdot q_2, \;\; \textit{and } b = \sqrt{q_3} \cdot q_4$$

where $\sqrt{q_1}$ *and* $\sqrt{q_3}$ *are positive square roots of* q_1 *and* q_3*, respectively. Then, the check* $(a > b)$ *can be performed using integers only.*

Proof. Consider the following equivalence. $(a > b)$ iff

1. $(a > 0)$ and $(b \leq 0)$, or
2. $(a = 0)$ and $(b < 0)$, or
3. $(a > 0)$ and $(b > 0)$ and $(a^2 > b^2)$, or
4. $(a < 0)$ and $(b < 0)$ and $(a^2 < b^2)$.

Then, to perform the check $(a > b)$ using integers only, it suffices to be able to perform all of the checks: (1) $(a > 0)$, (2) $(b > 0)$, (3) $(a = 0)$, (4) $(b = 0)$, and (5) $(a^2 > b^2)$.

Since by hypothesis, $\sqrt{q_1}$, and $\sqrt{q_3}$, are always positive, checks 1 through 4 above can be performed using integers only. Also, by hypothesis, since the squares of $\sqrt{q_1}$ and $\sqrt{q_3}$ are rationals, check 5 above can be performed using integers only. $\square$

9.2 Algorithm TRANSLATE

We assume that a particle is being translated from a *Initiator* $\mathcal{M}(I)$, in $\mathcal{M}(L)$, along the direction, given by the point *Destination*, in L.

Let $\mathcal{M}(P')$ denote an arbitrary processor in L.

Convention 9.2.1 *To avoid a plethora of expressions of the form* $P' - I$ *and* $Destination - I$ *in formulas, we make the following conventions that* in this section

1. P *denotes* $P' - I$ *and*
2. *Dest denotes* $Destination - I$.

9.2.1 Contents of Messages

Processor $\mathcal{M}(P')$ must receive the following in its messages.

1. the representation of the point P ($= P' - I$).

2. the representation of the point $Dest$ ($= Destination' - I$).

3. an integer value, t, in the field t_{in} such that $\texttt{CompLen} \cdot t$ is the simulation time at which $\mathcal{M}(P)$ was sent this message.

4. an integer value, l, in the field *leftend* so that the interval $[\rho_s(l), \rho_s(l+1))$ is the interval on ρ_s that the simulation is attempting to represent when $\mathcal{M}(P)$ executes step 2.

Given the values *leftend* and $(t_{in} - \mathit{leftend} \cdot \texttt{PathLgthDil})$, and assuming that the value of the constant `PathLgthDil` is accessible to every processor, it is possible to evaluate t_{in}. Actual implementations could use this fact to reduce the number of bits used in passing the information in items 3 and 4 above.

9.2.2 Choice of ρ_s

In the elaboration of step 2 of algorithm TRANSLATE as an instance of the algorithm template CURVE, we will use the following expression for $\rho_s(k)$, and we will show that all the checks, and computations that need to be performed in step 2 are computable. Note, however, that the numerical value of $\rho_s(k)$ itself may not be rational. As previously indicated we take here $s = (1/\texttt{SpaceDil})$.

$$\rho_s(k) = \frac{s \cdot k}{\|Dest\|} Dest$$

9.2.3 Testing: $[\rho_s(k), \rho_s(k+1)) \cap \overline{\mathrm{Voronoi}(P, \mathcal{L})} \neq \emptyset$

This section is a massive application of Proposition 4.2.1.

A. Determining $\mathrm{Dirs}_+(\rho_s(k), \rho_s(k+1), \mathcal{L})$ **and** $\mathrm{Dirs}_0(\rho_s(k), \rho_s(k+1), \mathcal{L})$

Lemma 9.2.1 *Suppose* $D \in \mathrm{Dirs}(\mathcal{L})$. *Then*

1. $D \in \mathrm{Dirs}_+(\rho_s(k), \rho_s(k+1), \mathcal{L}) \iff (Dest, D) > 0$, *and*

2. $D \in \mathrm{Dirs}_0(\rho_s(k), \rho_s(k+1), \mathcal{L}) \iff (Dest, D) = 0)$.

Proof.

$$
\begin{aligned}
& D \in \mathrm{Dirs}_+(\rho_s(k), \rho_s(k+1), \mathcal{L}) \\
\iff & ((\rho_s(k+1) - \rho_s(k)), D) > 0 \\
\iff & ((\frac{s \cdot (k+1)}{\|Dest\|} Dest - \frac{s \cdot k}{\|Dest\|} Dest), D) > 0 \\
\iff & s(Dest, D) > 0 \\
\iff & (Dest, D) > 0.
\end{aligned}
$$

Similarly, $D \in \mathrm{Dirs}_0(\rho_s(k), \rho_s(k+1), \mathcal{L}) \iff (Dest, D) = 0$. □

Since $Dest$ and D are vectors in the well-represented lattice, $\mathcal{L}$, computing $(Dest, D)$ involves only integers.

B. Computing the maxes and mins in (17) **and** (18) **of Lemma 4.2.1**

It is enough to prove the following two lemmas.

Lemma 9.2.2 *If* $D \in \mathrm{Dirs}_+(\rho_s(k), \rho_s(k+1), \mathcal{L})$, *then it is possible to perform the following comparisons using integers only.*

1. $$\frac{-\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} > \frac{-\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D') - (\rho_s(k), D')}{((\rho_s(k+1) - \rho_s(k)), D')},$$

2. $\dfrac{\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} > \dfrac{\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D') - (\rho_s(k), D')}{((\rho_s(k+1) - \rho_s(k)), D')}$,

3. $\dfrac{-\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} > 0$, *and*

4. $\dfrac{\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} > 1$.

PROOF.

We first derive an equivalent expression for $((\rho_s(k+1) - \rho_s(k)), D)$.

$$\begin{aligned}
& ((\rho_s(k+1) - \rho_s(k)), D) \\
= \; & \frac{s}{\|Dest\|} \cdot ((k+1) - k) \cdot (Dest, D) \\
= \; & \frac{s}{\|Dest\|} \cdot (Dest, D).
\end{aligned}$$

We derive equivalent conditions involving rational numbers in each of the four cases. We will combine the derivations for cases 1 and 2, and the derivations for cases 3 and 4.

Cases 1 and 2

$$\begin{aligned}
& \frac{\pm\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} \\
& \quad > \frac{\pm\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D') - (\rho_s(k), D')}{((\rho_s(k+1) - \rho_s(k)), D')} \\
\Longleftrightarrow \; & \frac{\pm\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D) - \frac{s \cdot k}{\|Dest\|}(Dest, D)}{\frac{s}{\|Dest\|}(Dest, D)} \\
& \quad > \frac{\pm\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D') - \frac{s \cdot k}{\|Dest\|}(Dest, D')}{\frac{s}{\|Dest\|}(Dest, D')} \\
\Longleftrightarrow \; & (Dest, D')[\pm\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D) - \frac{s \cdot k}{\|Dest\|}(Dest, D)] \\
& \quad > (Dest, D)[\pm\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D') - \frac{s \cdot k}{\|Dest\|}(Dest, D')] \\
\Longleftrightarrow \; & (Dest, D')[\pm\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D)] \\
& \quad > (Dest, D)[\pm\mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D')].
\end{aligned}$$

Cases 3 and 4 Suppose x is an integer.

$$\begin{aligned}
& \frac{\pm \mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} > x \\
\iff \; & \pm \mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D) - \frac{s \cdot k}{\|Dest\|}(Dest, D) \\
& > \frac{s}{\|Dest\|}(Dest, D) \cdot x \\
\iff \; & \|Dest\|(\pm \mathrm{MinLgth}(\mathcal{L})^2/2 + (P, D)) > s(Dest, D)(1 + x).
\end{aligned}$$

The equivalent expression for the first case above involves only rational numbers and hence can be checked using integers only. The equivalent expression for the second case involves rational numbers only, except for $\|Dest\|$, which is always positive. Further, $(\|Dest\|)^2$ is always an integer. Therefore, by Lemma 9.1.1, this expression has an equivalent expression involving integers only. Notice that the expressions in cases 3 and 4 above have equivalent expressions involving integers only, even if the inequality is reversed, or replaced by an equality. This helps us in checking the condition that the minima has to be less than 1 in (17). $\square$

Lemma 9.2.3 *If $D \in \mathrm{Dirs}_0(\rho_s(k), \rho_s(k+1), \mathcal{L})$, then it is possible to perform the following comparisons using integers only.*

1. $((P - \rho_s(k)), D) \pm \mathrm{MinLgth}(\mathcal{L})^2 > ((P - \rho_s(k)), D') \pm \mathrm{MinLgth}(\mathcal{L})^2$, *and*
2. $((P - \rho_s(k)), D) \pm \mathrm{MinLgth}(\mathcal{L})^2 > 0$.

Proof. We derive equivalent conditions involving rational numbers in each of the two cases.

1.

$$\begin{aligned}
& ((P - \rho_s(k)), D) \pm \mathrm{MinLgth}(\mathcal{L})^2 > ((P - \rho_s(k)), D') \pm \mathrm{MinLgth}(\mathcal{L})^2 \\
\iff \; & ((P - \rho_s(k)), D) > ((P - \rho_s(k)), D') \\
\iff \; & (P, (D - D')) > \frac{s \cdot k}{\|Dest\|}(Dest, (D - D')) \\
\iff \; & \|Dest\|(P, (D - D')) > s \cdot k(Dest, (D - D')).
\end{aligned}$$

2.

$$
\begin{aligned}
& ((P-\rho_s(k)),D) \pm \mathrm{MinLgth}(\mathcal{L})^2 > 0 \\
\iff\ & (P,D) \pm \mathrm{MinLgth}(\mathcal{L})^2 > \frac{s\cdot k}{\|Dest\|}(Dest,D) \\
\iff\ & \|Dest\|[(P,D) \pm \mathrm{MinLgth}(\mathcal{L})^2] > s\cdot k(Dest,D).
\end{aligned}
$$

The equivalent expressions in both the cases involve rational numbers only, except for $\|Dest\|$, which is always positive. Further, $(\|Dest\|)^2$ is always an integer. Therefore, by Lemma 9.1.1, these expressions have equivalent expressions involving integers only. □

9.2.4 Tests: $\rho_s(k) \in \mathrm{Voronoi}(P,\mathcal{L})$ and $\rho_s(k) \in \overline{\mathrm{Voronoi}(P,\mathcal{L})}$.

We need to show that the condition $|((\rho_s(k)-P),D)| \le \mathrm{MinLgth}(\mathcal{L})^2/2$ can be checked, using integers only, for each direction D in $\mathrm{Dirs}(\mathcal{L})$. Suppose $D \in \mathrm{Dirs}(\mathcal{L})$. Then,

$$
\begin{aligned}
& |((\rho_s(k)-P),D)| \le \mathrm{MinLgth}(\mathcal{L})^2/2 \\
\iff\ & |(\frac{s\cdot k}{\|Dest\|}Dest,D)-(P,D)| \le \mathrm{MinLgth}(\mathcal{L})^2/2 \\
\iff\ & -\mathrm{MinLgth}(\mathcal{L})^2 + 2(P,D) \\
& \quad \le 2\frac{s\cdot k}{\|Dest\|}(Dest,D) \\
& \quad\quad \le \mathrm{MinLgth}(\mathcal{L})^2 + 2(P,D) \\
\iff\ & \|Dest\|[-\mathrm{MinLgth}(\mathcal{L})^2 + 2(P,D)] \\
& \quad \le 2\cdot s\cdot k(Dest,D) \\
& \quad\quad \le \|Dest\|[\mathrm{MinLgth}(\mathcal{L})^2 + 2(P,D)].
\end{aligned}
$$

Then, each of the three expressions above has only rational numbers, except for $\|Dest\|$, which is always positive. Furthermore, $(\|Dest\|)^2$ is always an integer. Therefore, by Lemma 9.1.1, both these inequalities can be checked using integers only.

Notice that, even if both the inequalities in the last expression are replaced by "less than", the condition can be checked using integers only.

9.3 Algorithm REVOLVE

We assume that a particle, at $I \in L$, is being revolved about a point $O \in L$, in a plane given by O, I and a point $T' \in \mathcal{L}$, so that $(I - O)$ and $(T' - O)$ are perpendicular to each other. Corollary 9.3.1 (to Lemma 9.3.1) below shows that such a point T' always exists for any plane passing through O, I and a point in $\mathcal{L}$. We will assume that the particle is being revolved, in the simulation, at the angular speed MaxAngSpeed(L).

We assume that a particle is being revolved from a *Initiator* $\mathcal{M}(I)$, in $\mathcal{M}(L)$.

Let $\mathcal{M}(P')$ denote an arbitrary processor in L.

Convention 9.3.1 *To avoid a plethora of expressions of the form $P' - O$, $I' - O$, and $T' - O$, in formulas, we make the following conventions that* in this section

1. *P denotes $P' - O$,*

2. *I denotes $I' - O$, and*

3. *T denotes $T' - O$.*

Lemma 9.3.1 *Suppose I'', and X are distinct points in $\mathcal{L}$. Then there is a point T'', in the sublattice (of $\mathcal{L}$) generated by I'' and X, so that T'' is perpendicular to I''.*

Proof. Let $T'' = \alpha \cdot I'' + \beta \cdot X$. Then,

$$\begin{aligned}(I'', T'') &= (I'', (\alpha \cdot I'' + \beta \cdot X)) \\ &= \alpha(I'', I'') + \beta(I'', X).\end{aligned}$$

Let $\alpha = (I'', X)$ and $\beta = -(I'', I'')$. Then, both α and β are integers since $\mathcal{L}$ is a well-represented lattice. Therefore T'' is in the sublattice (of $\mathcal{L}$) generated by I'' and X. Also, $(I'', T'') = 0$. □

Corollary 9.3.1 *Suppose O'', I'' and X are distinct points in $\mathcal{L}$. Then there is a point T'', in the sublattice (of $\mathcal{L}$) generated by $(I'' - O'')$ and $(X - O'')$, so that $(T'' - O'')$ is perpendicular to $(I'' - O'')$.*

Proof. Translating the origin to O'' in the proof of the above lemma, we have this corollary. □

9.3.1 Contents of Messages

Each processor, $\mathcal{M}(P')$, must receive the following in its messages.

1. the representation of the point P ($= P' - O$).

2. the representation of the point I ($= I' - O$).

3. the representation of the point T ($= T' - O$).

4. an integer value, t, in the field t_{in} such that $\texttt{CompLen} \cdot t$ is the simulation time at which $\mathcal{M}(P)$ got this message.

5. an integer value, l, in the field *leftend* so that the interval $[\rho_s(l), \rho_s(l+1))$ is the interval on ρ_s that the simulation is attempting to represent when $\mathcal{M}(P)$ reads this message.

6. the numerator and denominator of $\cos(\mathit{leftend} \cdot \texttt{AngRes})$, and

7. the numerator and denominator of $\sin(\mathit{leftend} \cdot \texttt{AngRes})$.

9.3.2 Choice of $\rho_s(k)$

In the elaboration of step 2 of algorithm REVOLVE as an instance of the general algorithm CURVE, we will use the following expression for $\rho_s(k)$, and we will show that all the checks, and computations that need to be performed in step 2 are computable. Note, however, that the numerical value of $\rho_s(k)$ itself may not be rational.

$$\rho_s(k) = \cos(k \cdot \texttt{AngRes})I + \frac{\|I\|}{\|T\|}\sin(k \cdot \texttt{AngRes})T,$$

where, using the standard trigonometric identities,

$$\begin{aligned} \cos(k \cdot \texttt{AngRes}) &= \cos(\texttt{AngRes})\cos((k-1)\texttt{AngRes}) \\ &\qquad - \sin(\texttt{AngRes})\sin((k-1)\texttt{AngRes}), \end{aligned}$$

and

$$\begin{aligned} \sin(k \cdot \texttt{AngRes}) &= \sin(\texttt{AngRes})\cos((k-1)\texttt{AngRes}) \\ &\qquad + \cos(\texttt{AngRes})\sin((k-1)\texttt{AngRes}). \end{aligned}$$

Notice that, since $\cos(\texttt{AngRes})$ and $\sin(\texttt{AngRes})$ are both rational quantities, so will be $\cos(k\cdot\texttt{AngRes})$ and $\sin(k\cdot\texttt{AngRes})$, for all positive values of k. The above equations help in computing $\cos(k\cdot\texttt{AngRes})$ and $\sin(k\cdot\texttt{AngRes})$ using the sine and cosine values (rational) of the universal constant AngRes (see Section 6.3) and the values $\cos((k-1)\texttt{AngRes})$ and $\sin((k-1)\texttt{AngRes})$. As seen from the contents of messages above, each processor gets the sine and cosine values for the angle $\mathit{leftend}\cdot\texttt{AngRes}$. Thus each processor can, iteratively, compute the sine and cosine values for the angles $k\cdot\texttt{AngRes}$, $k > \mathit{leftend}$.

9.3.3 Testing: $[\rho_s(k),\rho_s(k+1))\cap\overline{\mathrm{Voronoi}(P,\mathcal{L})}\neq\emptyset$

This section is another massive application of Proposition 4.2.1.

A. Determining $\mathrm{Dirs}_+(\rho_s(k),\rho_s(k+1),\mathcal{L})$ **and** $\mathrm{Dirs}_0(\rho_s(k),\rho_s(k+1),\mathcal{L})$

Lemma 9.3.2 *Suppose $D\in\mathrm{Dirs}(\mathcal{L})$. The conditions $D\in\mathrm{Dirs}_+(\rho_s(k),\rho_s(k+1),\mathcal{L})$ and $D\in\mathrm{Dirs}_0(\rho_s(k),\rho_s(k+1),\mathcal{L})$ can be checked using integers only.*

PROOF.

1.

$$\begin{aligned}
& D\in\mathrm{Dirs}_+(\rho_s(k),\rho_s(k+1),\mathcal{L})\\
\iff\ & ((\rho_s(k+1)-\rho_s(k)),D)>0\\
\iff\ & (((\cos((k+1)\cdot\texttt{AngRes})I+\frac{\|I\|}{\|T\|}\sin(k\cdot\texttt{AngRes})T)\\
& -(\cos(k\cdot\texttt{AngRes})I+\frac{\|I\|}{\|T\|}\sin(k\cdot\texttt{AngRes})T)),D)>0\\
\iff\ & \|T\|[(\cos((k+1)\texttt{AngRes})-\cos(k\texttt{AngRes}))(I,D)]\\
& \quad > -\|I\|[(\sin((k+1)\texttt{AngRes})-\sin(k\texttt{AngRes}))(T,D)].
\end{aligned}$$

2.

$$D\in\mathrm{Dirs}_0(\rho_s(k),\rho_s(k+1),\mathcal{L})$$

$$\begin{aligned}
&\iff ((\rho_s(k+1) - \rho_s(k)), D) = 0 \\
&\iff (((\cos((k+1)\cdot \texttt{AngRes})I + \frac{\|I\|}{\|T\|}\sin(k \cdot \texttt{AngRes})T) \\
&\qquad -(\cos(k\cdot \texttt{AngRes})I + \frac{\|I\|}{\|T\|}\sin(k\cdot \texttt{AngRes})T)), D) = 0 \\
&\iff \|T\|[(\cos((k+1)\texttt{AngRes}) - \cos(k\texttt{AngRes}))(I, D)] \\
&\qquad = -\|I\|[(\sin((k+1)\texttt{AngRes}) - \sin(k\texttt{AngRes}))(T, D)].
\end{aligned}$$

The equivalent expressions in both the cases involve rational numbers only, except for $\|T\|$ and $\|I\|$, which are always positive. Further, $(\|T\|)^2$ and $(\|I\|)^2$ are always integers. Therefore, by Lemma 9.1.1, these expressions have equivalent expressions involving integers only. □

B. Computing the maxes and mins in (17) and (18) of Lemma 4.2.1

For convenience, for a fixed P, and a fixed k, we make the following abbreviations.

$$\begin{aligned}
c_n &= \texttt{AngRes.CosineNum}, \\
s_n &= \texttt{AngRes.SineNum}, \\
h &= \texttt{AngRes.Denom}, \\
c_n(k) &= \text{ the numerator of } \cos(\textit{leftend} \cdot \texttt{AngRes}), \\
s_n(k) &= \text{ the numerator of } \sin(\textit{leftend} \cdot \texttt{AngRes}), \\
h(k) &= \text{ the common denominator of } \sin(\textit{leftend} \cdot \texttt{AngRes}), \\
&\quad\ \text{ and } \cos(\textit{leftend} \cdot \texttt{AngRes}), \\
a(D)_+ &= h\cdot(h(k)(2(P,D)+M) - 2c_n(k)(I,D)), \\
a(D)_- &= h\cdot(h(k)(2(P,D)-M) - 2c_n(k)(I,D)), \\
b(D) &= 2h\cdot s_n(k), \\
c(D) &= 2(c_n(k)\cdot(c_n - h) - s_n(k)\cdot s_n)(I,D), \\
d(D) &= 2(s_n(k)\cdot(c_n - h) + c_n(k)\cdot s_n)(T,D).
\end{aligned}$$

A processor $\mathcal{M}(P)$ initializes the values for $c_n(k)$, $s_n(k)$, and $h(k)$ from fields in the messages it receives.

Notice that each of $a(D)_+, a(D)_-, b(D), c(D)$, and $d(D)$ are integer expressions. Then, it is easy to check that

$$\frac{((P-\rho_s(k)),D)+\mathrm{MinLgth}(\mathcal{L})^2/2}{((\rho_s(k+1)-\rho_s(k)),D)}=\frac{\|T\|a(D)_+-\|I\|b(D)}{\|T\|c(D)-\|I\|d(D)},\text{ and}$$

$$\frac{((P-\rho_s(k)),D)-\mathrm{MinLgth}(\mathcal{L})^2/2}{((\rho_s(k+1)-\rho_s(k)),D)}=\frac{\|T\|a(D)_--\|I\|b(D)}{\|T\|c(D)-\|I\|d(D)}.$$

The following two lemmas show that each of the checks involved in computing the maximum and minimum in equations (1) and (2) in Lemma 4.2.1 can be performed using only integer and integer vector arithmetic.

Lemma 9.3.3 *Suppose p and q are integers, $q \neq 0$. If $D \in \mathrm{Dirs}_+(\rho_s(k),\rho_s(k+1),\mathcal{L})$, then,*

1.

$$\begin{aligned}&\frac{\|T\|a(D)_+-\|I\|b(D)}{\|T\|c(D)-\|I\|d(D)}>\frac{\|T\|a(D')_+-\|I\|b(D')}{\|T\|c(D')-\|I\|d(D')}\\ \Longleftrightarrow\ &(\|T\|)^2(a(D)_+c(D')-a(D')_+c(D))+(\|I\|)^2(b(D)d(D')-b(D')d(D))\\ &\quad>\|T\|\cdot\|I\|(b(D)c(D')+a(D)_+d(D')-c(D)b(D')-a(D')_+d(D)).\end{aligned}$$

2.

$$\begin{aligned}&\frac{\|T\|a(D)_--\|I\|b(D)}{\|T\|c(D)-\|I\|d(D)}>\frac{\|T\|a(D')_--\|I\|b(D')}{\|T\|c(D')-\|I\|d(D')}\\ \Longleftrightarrow\ &(\|T\|)^2(a(D)_-c(D')-a(D')_-c(D))+(\|I\|)^2(b(D)d(D')-b(D')d(D))\\ &\quad>\|T\|\cdot\|I\|(b(D)c(D')+a(D)_-d(D')-c(D)b(D')-a(D')_-d(D)).\end{aligned}$$

3.

$$\begin{aligned}&\frac{\|T\|a(D)_+-\|I\|b(D)}{\|T\|c(D)-\|I\|d(D)}>p/q\\ \Longleftrightarrow\ &\|T\|(a(D)_+q-c(D)p)>\|I\|(b(D)q-d(D)p).\end{aligned}$$

4.

$$\begin{aligned}&\frac{\|T\|a(D)_--\|I\|b(D)}{\|T\|c(D)-\|I\|d(D)}>p/q\\ \Longleftrightarrow\ &\|T\|(a(D)_-q-c(D)p)>\|I\|(b(D)q-d(D)p).\end{aligned}$$

PROOF. We derive equivalent conditions involving rational numbers in each of the two cases.

1.

$$\begin{aligned}
& \frac{\|T\|a(D)-\|I\|b(D)}{\|T\|c(D)-\|I\|d(D)} > \frac{\|T\|a(D')-\|I\|b(D')}{\|T\|c(D')-\|I\|d(D')} \\
\iff\ & (\|T\|)^2(a(D)c(D')-a(D')c(D))+(\|I\|)^2(b(D)d(D')-b(D')d(D)) \\
& \quad > \|T\|\cdot\|I\|(b(D)c(D')+a(D)d(D')-c(D)b(D')-a(D')d(D)).
\end{aligned}$$

2.

$$\begin{aligned}
& \frac{\|T\|a(D)-\|I\|b(D)}{\|T\|c(D)-\|I\|d(D)} > p/q \\
\iff\ & \|T\|(a(D)q-c(D)p) > \|I\|(b(D)q-d(D)p).
\end{aligned}$$

The equivalent expressions in both the cases involve rational numbers only, except for $\|T\|$ and $\|I\|$, which are always positive. Further, $(\|T\|)^2$ and $(\|I\|)^2$ are always integers. Therefore, by Lemma 9.1.1, these expressions have equivalent expressions involving integers only. Notice that the expressions in each of the cases above have equivalent expressions involving integers only, even if the inequality is reversed, or replaced by an equality. This helps us in checking the condition that the minima has to be less than 1 in (17). □

Lemma 9.3.4 *If* $D \in \mathrm{Dirs}_0(\rho_s(k), \rho_s(k+1), \mathcal{L})$*, then*

1.

$$\begin{aligned}
& ((P-\rho_s(k)), D) + \mathrm{MinLgth}(\mathcal{L})^2 > ((P-\rho_s(k)), D') + \mathrm{MinLgth}(\mathcal{L})^2 \\
& \iff \|T\|(a(D)_+ - a(D')_+) > \|I\|(b(D)-b(D')).
\end{aligned}$$

2.

$$\begin{aligned}
& ((P-\rho_s(k)), D) - \mathrm{MinLgth}(\mathcal{L})^2 > ((P-\rho_s(k)), D') - \mathrm{MinLgth}(\mathcal{L})^2 \\
& \iff \|T\|(a(D)_- - a(D')_-) > \|I\|(b(D)-b(D')).
\end{aligned}$$

3.

$$((P - \rho_s(k)), D) + \mathrm{MinLgth}(\mathcal{L})^2 > 0 \Longleftrightarrow \|T\|a(D)_+ > \|I\|b(D).$$

4.

$$((P - \rho_s(k)), D) - \mathrm{MinLgth}(\mathcal{L})^2 > 0 \Longleftrightarrow \|T\|a(D)_- > \|I\|b(D).$$

PROOF. We derive equivalent conditions involving rational numbers in each of the two cases. We use the abbreviations above.

1.

$$\begin{aligned} & ((P - \rho_s(k)), D) \pm \mathrm{MinLgth}(\mathcal{L})^2 > ((P - \rho_s(k)), D') \pm \mathrm{MinLgth}(\mathcal{L})^2 \\ \Longleftrightarrow\ & \|T\|a(D) - \|I\|b(D) > \|T\|a(D') - \|I\|b(D') \\ \Longleftrightarrow\ & \|T\|(a(D) - a(D')) > \|I\|(b(D) - b(D')). \end{aligned}$$

2.

$$\begin{aligned} & ((P - \rho_s(k)), D) \pm \mathrm{MinLgth}(\mathcal{L})^2 > 0 \\ \Longleftrightarrow\ & \|T\|a(D) - \|I\|b(D) > 0 \\ \Longleftrightarrow\ & \|T\|a(D) > \|I\|b(D). \end{aligned}$$

The equivalent expressions in both the cases involve rational numbers only, except for $\|T\|$ and $\|I\|$, which are always positive. Further, $(\|T\|)^2$ and $(\|I\|)^2$ are always integers. Therefore, by Lemma 9.1.1, these expressions have equivalent expressions involving integers only. □

9.3.4 Tests: $\rho_s(k) \in \mathrm{Voronoi}(P, \mathcal{L})$ and $\rho_s(k) \in \overline{\mathrm{Voronoi}(P, \mathcal{L})}$.

In the following lemma shows that the condition $|((\rho_s(k) - P), D)| \leq \mathrm{MinLgth}(\mathcal{L})^2/2$ can be checked, using integers only, for each direction D in $\mathrm{Dirs}(\mathcal{L})$. Suppose $D \in$

Dirs($\mathcal{L}$). Then,

$$\begin{aligned} & |((\rho_s(k) - P), D)| \leq \mathrm{MinLgth}(\mathcal{L})^2/2 \\ \iff & -\mathrm{MinLgth}(\mathcal{L})^2 \leq 2((\rho_s(k) - P), D) \leq \mathrm{MinLgth}(\mathcal{L})^2 \\ \iff & 0 \leq 2((\rho_s(k) - P), D) + \mathrm{MinLgth}(\mathcal{L})^2 \leq 2\mathrm{MinLgth}(\mathcal{L})^2. \end{aligned}$$

Then by case 2 of Lemma 9.3.3, and case 2 of Lemma 9.3.4, we have that each of the two inequalities above can be checked using integers only.

Notice that, even if both the inequalities in the last expression are replaced by "less than", the condition can be checked using integers only. □

Chapter 10

The Algorithms

In this section we present the algorithms WAVE, WAVE$'$, TRANSLATE, and REVOLVE in a Pascal-like specification language.

Convention: For expositional convenience we will abbreviate MinLgth$(\mathcal{L})^2$ by M. (Recall, from Section 4, that the square of the minimal length in a well-represented lattices is an integer; therefore, M is an integer.) Also, recall from Section 5, that MaxLinSpeed$(L) = 1/(2\text{Diam}(L))$. We will abbreviate Diam(L) by R.

For expositional cleanness, we have used procedure calls in the algorithms. Each of the called procedures has been described before it is called. In an actual implementation, it is conceivable that these procedures are implemented as macros, with their local variables made global. The procedures we call are RatGt, RatEq, RatGtEq, CutsVCTr, and CutsVCRev.

10.1 Useful Algorithms for Rational Numbers

procedure RatGt($N_1, D_1, N_2, D_2, N_3, D_3, N_4, D_4$: **integer**) : **boolean**;

(∗ Given the integers $N_1, \ldots, N_4$, and $D_1, \ldots, D_4$, this procedure compares the two expressions: (i)$\sqrt{(N_1/D_1)} \cdot (N_2/D_2)$, and (ii) $\sqrt{(N_3/D_3)} \cdot (N_4/D_4)$, under the assumption that the two square roots are positive. This procedure returns the value **true** if expression (i) is greater than to expression (ii), and returns **false** otherwise. ∗)

```
begin (* procedure RatGt *)
    if
        ((N2 > 0) and (N4 < 0)) or
        ((N2 > 0) and (N3 · N4 = 0)) or
        ((N1 · N2 = 0) and (N4 < 0)) or
        (((N2 > 0) and (N4 > 0))  and
            (N1 · N2^2 · D3 · D4^2 > N3 · N4^2 · D1 · D2^2)) or
        (((N2 < 0) and (N4 < 0))  and
            (N1 · N2^2 · D3 · D4^2 < N3 · N4^2 · D1 · D2^2))
    then
        return true;
    else
        return false;
    endif;
end; (* procedure RatGt*)
```

procedure RatEq($N_1, D_1, N_2, D_2, N_3, D_3, N_4, D_4$: **integer**) : **boolean**;

(∗ Given the integers $N_1, \ldots, N_4$, and $D_1, \ldots, D_4$, this procedure compares the two expressions: (i)$\sqrt{(N_1/D_1)} \cdot (N_2/D_2)$, and (ii) $\sqrt{(N_3/D_3)} \cdot (N_4/D_4)$, under the assumption that the two square roots are positive. This procedure returns the value **true** if expression (i) is equal to expression (ii), and returns **false** otherwise. ∗)

```
begin (* procedure RatEq*)
    if (RatGt(N1, D1, N2, D2, N3, D3, N4, D4) or
       RatGt(N3, D3, N4, D4, N1, D1, N2, D2)) then
```

```
        return false;
    else
        return true;
    endif;
end; (* procedure RatEq*)
```

procedure RatGtEq($N_1, D_1, N_2, D_2, N_3, D_3, N_4, D_4$: **integer**) : **boolean**;

(* Given the integers $N_1, \ldots, N_4$, and $D_1, \ldots, D_4$, this procedure compares the two expressions: (i)$\sqrt{(N_1/D_1)} \cdot (N_2/D_2)$, and (ii) $\sqrt{(N_3/D_3)} \cdot (N_4/D_4)$, under the assumption that the two square roots are positive. This procedure returns the value **true** if expression (i) is greater than or equal to expression (ii), and returns **false** otherwise. *)

```
begin (* procedure RatGtEq*)
    if (RatGt(N1, D1, N2, D2, N3, D3, N4, D4) or
       RatGt(N3, D3, N4, D4, N1, D1, N2, D2) or
           RatEq(N3, D3, N4, D4, N1, D1, N2, D2)) then
        return false;
    else
        return true;
    endif;
end; (* procedure RatGtEq*)
```

10.2 Algorithm WAVE

```
type MessageType = record
                     CoOrds: vector;
                   end;

var
    D_0 : Directions;
    P : vector;
    T : integer;
    Distsq : array Directions of integer;
    PosFlag : array Directions of boolean;
    InDirs, SendDirs, SendThisPass : set of Directions;

begin (* Algorithm WAVE *)
    1. if Done then goto step 3;endif;
    2. if (not REPR) then
         Initialize Parameters using external input, if any, and
             { Buf⟨D⟩.InMessage|D ∈ InDirs }, if non-empty;
         Let D_0 such that D_0 ∈ InDirs;
         P := Buf⟨D⟩.InMessage.CoOrds;
         T := Buf⟨D⟩.SendTime;
       endif;
       SendDirs := {D|(P, P) < ((P + D), (P + D))};
       for each D ∈ SendDirs
           Distsq(D) := SpaceDil^2 · ((P + D), (P + D));
       endfor;
       for each D ∈ SendDirs, PosFlag(D) := false endfor;
       if (SendDirs ≠ ∅) then
       for each D ∈ SendDirs
           if (not PosFlag(D)) then
               if(1 ≥ 2(Distsq(D) − (T + PassCount)(T + PassCount + 1))) then
                   Buf⟨D⟩.SendTime:= PassCount;
                   Buf⟨D⟩.OutMessage.CoOrds:= (P + D);
                   SendDirs := SendDirs \ {D};
               else
```

$\texttt{PosFlag}(D) := (\texttt{Distsq}(D) - (T + \texttt{PassCount} + 1)^2) > 0;$

endif;

else

$\mathit{Buf}\langle D\rangle.\mathit{SendTime} := \texttt{PassCount};$

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\texttt{CoOrds} := (P + D);$

$\mathit{SendDirs} := \mathit{SendDirs} \setminus \{D\};$

endif;

endfor;

else

REPR := **false**;

endif;

$\mathit{SendThisPass} := \{D \in \mathit{SendDirs} \mid \mathit{Buf}\langle D\rangle.\mathit{SendTime} = \texttt{PassCount}\};$

3. **idle till** $\texttt{SimClock} = (\texttt{CompLen} \cdot \texttt{PassCount}) - 1;$

simul send

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}$ in direction D, for each $D \in \mathit{SendThisPass}$;

endsend;

end. (∗ Algorithm WAVE ∗)

10.3 Algorithm WAVE′

type *MessageType* = **record**
    Norm : `integer`;
    NormDiff : `integer`;
    GraphPL : `integer`;
  end;

var
  `Distsq`, G, T, `Ddiff` : `integer`;
  `DistsqArr`, N, Y, m : **array** `Directions` **of** `integer`;
  D' : `Directions`;
  `Opp`, `Rt`, `Obtuse`, A : `VectorSet`;
  `PosFlag` : **array** `Directions` **of boolean**;
  InDirs, *SendDirs*, *SendThisPass* : **set of** `Directions`;

begin (∗ Algorithm WAVE′ ∗)
  1. **if** `Done` **then goto** step 3;**endif**;
  2. **if** (**not** REPR) **then**
    Initialize Parameters using external input, if any, and
      { $Buf\langle D\rangle.InMessage | D \in InDirs$ }, if non-empty;
    Let D_0 such that $D_0 \in InDirs$;
    `Distsq` := $Buf\langle D\rangle.InMessage$.Norm;
    G := $Buf\langle D\rangle.InMessage$.GraphPL;
    T := $Buf\langle D\rangle.SendTime$;
    for each $D \in InDirs$
      $N(D)$:= $Buf\langle D\rangle.InMessage$.NormDiff;
    endfor;
  endif;
    $SendDirs := \left[\bigcap_{D \in InDirs} \{ D' \in \texttt{Directions} \mid (D, D') \leq 0 \} \right]$;
    `Opp` := $\{D | D \in InDirs\}$;
    `Rt` := $\{D | \forall D' \in InDirs, (D, D') = 0\}$;
    `Obtuse` := $SendDirs \setminus (\texttt{Opp} \cup \texttt{Rt})$;
    for each $D \in$ `Opp`
      $Buf\langle D\rangle.OutMessage$.NormDiff:= (`DistsqArr`$(-D) + 4$);

$\texttt{DistsqArr}(D) := \texttt{Distsq} + \texttt{SpaceDil}^2 \cdot (\texttt{DistsqArr}(-D) + 4);$

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\mathrm{Norm}{:=}\ \texttt{DistsqArr}(D);$

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\mathrm{GraphPL}{:=}\ G + 1;$

endfor;

for each $D \in \texttt{Rt}$

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\mathrm{NormDiff}{:=}\ 2;$

$\texttt{DistsqArr}(D) := \texttt{Distsq} + 2\texttt{SpaceDil}^2;$

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\mathrm{Norm}{:=}\ \texttt{DistsqArr}(D);$

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\mathrm{GraphPL}{:=}\ G + 1;$

endfor;

for each $D \in \texttt{Obtuse}$

$A(D) := \{D' \in \mathit{InDirs} | (D, D') = -1\};$

$Y(D) := \sum_{D \in A(D)} N(D);$

$m(D) := |A(D)|;$

$\texttt{Ddiff} := ((Y(D) - 2 \cdot G)\ \mathbf{div}\ m(D)) + 4;$

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\mathrm{NormDiff}{:=}\ \texttt{Ddiff};$

$\texttt{DistsqArr}(D) := \texttt{Distsq} + \texttt{SpaceDil}^2 \cdot \texttt{Ddiff}$

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\mathrm{Norm}{:=}\ \texttt{DistsqArr}(D);$

$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\mathrm{GraphPL}{:=}\ G + 1;$

endfor;

for each $D \in \mathit{SendDirs}, \texttt{PosFlag}(D) :=$ **false endfor**;

if $(\mathit{SendDirs} \neq \emptyset)$ **then**

for each $D \in \mathit{SendDirs}$

if (**not** $\texttt{PosFlag}(D)$) **then**

if$(1 \geq 2(\texttt{DistsqArr}(D) - (T + \texttt{PassCount})(T + \texttt{PassCount} + 1)))$ **then**

$\mathit{Buf}\langle D\rangle.\mathit{SendTime}{:=}\ \texttt{PassCount};$

$\mathit{SendDirs} := \mathit{SendDirs} \setminus \{D\};$

else

$\texttt{PosFlag}(D) := (\texttt{DistsqArr}(D) - (T + \texttt{PassCount} + 1)^2) > 0;$

endif;

else

$\mathit{Buf}\langle D\rangle.\mathit{SendTime}{:=}\ \texttt{PassCount};$

$\mathit{SendDirs} := \mathit{SendDirs} \setminus \{D\};$

endif;

endfor;

endif;

$\mathit{SendThisPass} := \{D \in \mathit{SendDirs} \mid \mathit{Buf}\langle D\rangle.\mathit{SendTime} = \texttt{PassCount}\}$;

3. **idle till** $\texttt{SimClock} = (\texttt{CompLen} \cdot \texttt{PassCount}) - 1$;

   simul send

   $\mathit{Buf}\langle D\rangle.\mathit{OutMessage}$ in direction D, for each $D \in \mathit{SendThisPass}$;

   endsend;

end. (∗ Algorithm WAVE′ ∗)

10.4 Algorithm TRANSLATE

type *MessageType* = **record**

  Dest : `vector`;

  `CoOrds` : `vector`;

  t_{in} : `integer`;

  leftend : `integer`;

end;

var

InVC, OnBorder, InSideVC : **boolean**;

t_{in}, *leftend* : `integer`;

P, *Dest* : `vector`;

procedure CutsVCTr (k : **natural**; InVC, OnBorder, InSideVC : **boolean**);

(* Given a natural number, k, the following are true at the end of the execution of this procedure.

1. InVC is **true** iff $[\rho_s(k), \rho_s(k+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset$,
2. OnBorder is **true** iff $\rho_s(k+1) \in \text{Boundary}(\text{Voronoi}(P, \mathcal{L}))$, and
3. InSideVC is **true** iff $\rho_s(k+1) \in \text{Voronoi}(P, \mathcal{L})$,

where the curve ρ is the straight line defined by the origin, and the point *Dest*. *)

var

InVC, OnBorder, InSideVC : **boolean**;

D1gt0, D2gt1, D1lt1, D1gtD2, D2lt1, D2lt0, D3leD4: **boolean**;

DirsPlus, DirsZero : `VectorSet`;

begin (* procedure CutsVCTr *)

InVC := **false**;

OnBorder := **false**;

InSideVC := **false**;

DirsPlus := $\{D \in \text{Dirs}(\mathcal{L}) | (D, Dest) > 0\}$;

DirsZero := $\{D \in \text{Dirs}(\mathcal{L}) | (Dest, D) = 0\}$;

Let $D_1 \in$ DirsPlus so that, $\forall D \in$ DirsPlus,

$(-M+2(P,D_1))(Dest,D) \geq (-M+2(P,D_1))(Dest,D_1)$;

Let $D_2 \in$ DirsPlus so that, $\forall D \in$ DirsPlus,

$(M+2(P,D_2))(Dest,D) \leq (M+2(P,D))(Dest,D_2)$

Let $D_3 \in$ DirsZero so that, $\forall D \in$ DirsZero,

$\mathrm{RatGt}((Dest,Dest),1,(P,(D_3-D)),1,1,1,(k\cdot(Dest,(D_3-D))),2R)$;

Let $D_4 \in$ DirsZero so that, $\forall D \in$ DirsZero,

$\mathrm{RatGt}((Dest,Dest),1,(P,(D-D_4)),1,1,1,(k\cdot(Dest,(D-D_4))),2R)$;

D1gt0 := $\mathrm{RatGt}((Dest,Dest),1,2R(2(P,D_1)-M),1,1,1,(Dest,D_1),1)$;

D2gt1 := $\mathrm{RatGt}((Dest,Dest),1,2R(2(P,D_2)+M),1,1,1,2(Dest,D_2),1)$;

D1lt1 := $\mathrm{RatGt}(1,1,2(Dest,D_1),1,(Dest,Dest),1,2R(2(P,D_1)-M),1)$;

D1gtD2 := $((Dest,D_2)(2(P,D_1)-M) > (Dest,D_2)(2(P,D_2)+M))$;

D2lt1 := $\mathrm{RatGt}(1,1,2(Dest,D_1),1,(Dest,Dest),1,2R(2(P,D_2)+M),1)$;

D2lt0 := $\mathrm{RatGt}(1,1,(Dest,D_2),1,(Dest,Dest),1,2R(2(P,D_2)+M),1)$;

D3leD4 := $\mathrm{RatGtEq}((Dest,Dest),1,2R((P,D_4)+M),1,1,1,k(Dest,D_4),1)$ **and**
 $\mathrm{RatGtEq}(1,1,k(Dest,D_3),1,(Dest,Dest),1,2R((P,D_3)-M))$;

if (D3leD4) **then**
  if (D1gt0) **then**
  if (D2gt1) **then**
      if (D1lt1) **then**
        InVC := **true**; InSideVC := **true**;
      endif;
    else
      if (**not** D1gtD2) **then**
        InVC := **true**; InSideVC := **false**;
      endif;
    endif;
  else
    if (D2gt1) **then**
      InVC := **true**; InSideVC := **true**;
    else
      if (D2lt1) **then**
        if (**not** D2lt0) **then**
          InVC := **true**; InSideVC := **false**;
        endif;

else
InVC := **true**; InSideVC := **false**;
endif;
endif;
endif;
OnBorder := (**not**(D2lt1 **or** D2gt1) **and** InVC);
endif;
end; (∗ procedure CutsVCTr ∗)

Body of Algorithm TRANSLATE
begin
1. **if** `Done` **then goto** step 3;**endif**;
2. **if** (**not** REPR) **then**
Initialize Parameters using external input, if any, and
$\{ \mathit{Buf}\langle D\rangle.\mathit{InMessage} | D \in \mathit{InDirs} \}$, if non-empty;
Let $D_0 \in \mathit{InDirs}$;
$t_{in} := \mathit{Buf}\langle D_0\rangle.\mathit{InMessage}.t_{in}$;
$P := \mathit{Buf}\langle D_0\rangle.\mathit{InMessage}.\texttt{CoOrds}$;
$\mathit{Dest} := \mathit{Buf}\langle D_0\rangle.\mathit{InMessage}.\mathit{Dest}$;
endif;
CutsVCTr (*leftend*, InVC, OnBorder, InSideVC);
if InVC **then**
REPR := **false**;
`Done` := **true**;
goto step 4;
else
REPR := **true**;
$\mathit{SendDirs} := \emptyset$;
if not InSideVC **then**
$\mathit{SendDirs} := \{D | ((\rho_s(\mathit{leftend}+1) - P), D) \geq \mathrm{MinLgth}(\mathcal{L})^2/2\}$;
if OnBorder **then**
for each $D \in \mathit{SendDirs}$,
$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.t_{in} := \texttt{PathLgthDil} \cdot (\mathit{leftend}+1)$;
$\mathit{Buf}\langle D\rangle.\mathit{OutMessage}.\mathit{leftend} := \mathit{leftend}+1$;

$\quad\quad\quad\quad\quad\quad$ *Buf*$\langle D\rangle$.*OutMessage*.*Dest* := *Dest*;

$\quad\quad\quad\quad$ **endfor**;

$\quad\quad\quad$ **else**

$\quad\quad\quad\quad$ **for each** $D \in$ *SendDirs*,

$\quad\quad\quad\quad\quad\quad$ *Buf*$\langle D\rangle$.*OutMessage*.$t_{in} := t_{in} + 1$;

$\quad\quad\quad\quad\quad\quad$ *Buf*$\langle D\rangle$.*OutMessage*.*leftend* := *leftend*;

$\quad\quad\quad\quad\quad\quad$ *Buf*$\langle D\rangle$.*OutMessage*.*Dest* := *Dest*;

$\quad\quad\quad\quad$ **endfor**;

$\quad\quad\quad$ **endif**;

$\quad\quad$ **endif**;

$\quad$ **endif**;

$\quad$ $t_{in} := t_{in} + 1$;

$\quad$ *leftend* := (t_{in} **div** `PathLgthDil`);

$\quad$ *SendThisPass* := $\{D \in$ *SendDirs* $\mid$ *Buf*$\langle D\rangle$.*OutMessage*.t_{in} = `PassCount`$\}$;

3. **idle till** `SimClock` = (`CompLen` $\cdot$ `PassCount`) $- 1$;

$\quad$ **simul send**

$\quad\quad$ *Buf*$\langle D\rangle$.*OutMessage* in direction D, for each $D \in$ *SendThisPass*;

$\quad$ **endsend**;

end.

10.5 Algorithm REVOLVE

type *MessageType* = **record**

`CoOrds: vector;`
`Initiator: vector;`
`Tangent: vector;`
t_{in} : `integer;`
leftend : `integer;`
`CosNum : integer;`
`SinNum : integer;`
`Denom : integer;`

end;

var

InVC, OnBorder, InSideVC : **boolean**;
t_{in}, *leftend* : `integer;`
P, I, T, *Dest* : `vector;`
$c_n(k)$, $s_n(k)$, $h(k)$: `integer;`
`NewCn, NewSn, NewhK : integer;`

procedure CutsVCRev (k : **natural**; InVC, OnBorder, InSideVC : **boolean**);

(∗ Given a natural number, k, the following are true at the end of the execution of this procedure,

1. InVC is **true** iff $[\rho_s(k), \rho_s(k+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset$,
2. OnBorder is **true** iff $\rho_s(k+1) \in \text{Boundary}(\text{Voronoi}(P, \mathcal{L}))$, and
3. InSideVC is **true** iff $\rho_s(k+1) \in \text{Voronoi}(P, \mathcal{L})$,

where the curve ρ is the circle of radius $\|I\|$, centered at the origin, and in the plane given by the perpendicular vectors I and T. ∗)

var

InVC, OnBorder, InSideVC : **boolean**;
D1gt0, D2gt1, D1lt1, D1gtD2, D2lt1, D2lt0, D3leD4: **boolean**;
DirsPlus, DirsZero : `VectorSet;`
D_1, D_2, D_3, D_4 : `Directions;`
TempL, TempR : **array** `Directions` **of** `integer;`

A, B : **array** `Directions`, `Directions` **of** `integer`;

begin (* procedure CutsVCRev *)

InVC := **false**;

OnBorder := **false**;

InSideVC := **false**;

for each $D \in \mathrm{Dirs}(\mathcal{L})$,

$\mathrm{TempL}(D) := (c_n(k)(c_n - h) - s_n(k) \cdot s_n) \cdot (I, D)$;

$\mathrm{TempR}(D) := (s_n(k)(h - c_n) - c_n(k) \cdot s_n) \cdot (T, D)$;

endfor;

$\mathrm{DirsPlus} := \{D \in \mathrm{Dirs}(\mathcal{L})|$

$\mathrm{RatGt}((T,T), 1, \mathrm{TempL}(D), 1, (I,I), 1, \mathrm{TempR}(D), 1)\}$;

$\mathrm{DirsZero} := \{D \in \mathrm{Dirs}(\mathcal{L})|$

$\mathrm{RatEq}((T,T), 1, \mathrm{TempL}(\mathrm{D}), 1, (I,I), 1, \mathrm{TempR}(D), 1)\}$;

for each $D, D' \in \mathrm{DirsPlus}$,

$A(D,D') := (T,T)(a(D)_-c(D') - a(D')_-c(D)) + (I,I)(b(D)d(D') - b(D')d(D))$;

$B(D,D') := (b(D)c(D') + a(D)_-d(D') - c(D)b(D') - a(D')_-d(D))$;

endfor;

Let $D_1 \in \mathrm{DirsPlus}$ so that, $\forall D \in \mathrm{DirsPlus}$,

$\mathrm{RatGt}(1, 1, A(D_1, D), 1, (T,T) \cdot (I,I), 1, B(D_1, D), 1)$;

for each $D, D' \in \mathrm{DirsPlus}$,

$A(D,D') := (T,T)(a(D)_+c(D') - a(D')_+c(D)) + (I,I)(b(D)d(D') - b(D')d(D))$;

$B(D,D') := (b(D)c(D') + a(D)_+d(D') - c(D)b(D') - a(D')_+d(D))$;

endfor;

Let $D_2 \in \mathrm{DirsPlus}$ so that, $\forall D \in \mathrm{DirsPlus}$,

$\mathrm{RatGt}(1, 1, A(D, D_2), 1, (T,T) \cdot (I,I), 1, B(D, D_2), 1)$;

Let $D_3 \in \mathrm{DirsZero}$ so that, $\forall D \in \mathrm{DirsZero}$,

$\mathrm{RatGtEq}((T,T), 1, (a(D_3)_- - a(D)_-), 1, (I,I), 1, (b(D_3) - b(D)), 1)$;

Let $D_4 \in \mathrm{DirsZero}$ so that, $\forall D \in \mathrm{DirsZero}$,

$\mathrm{RatGtEq}((T,T), 1, (a(D_4)_+ - a(D)_+), 1, (I,I), 1, (b(D_4) - b(D)), 1)$;

$\mathrm{D1gt0} := \mathrm{RatGt}((I,I), 1, a(D_1)_-, 1, (T,T), 1, b(D_1), 1)$;

$\mathrm{D2gt1} := \mathrm{RatGt}((I,I), 1, (a(D_2)_+ - c(D_2)), 1, (T,T), 1, (b_(D_2) - d(D_2)), 1)$;

$\mathrm{D1lt1} := \mathrm{RatGt}((T,T), 1, (b(D_1) - d(D_1)), 1, (I,I), 1, (a(D_1)_- - c(D_1)), 1)$;

$A(D_1, D_2) := (T,T)(a(D_1)_-c(D_2) - a(D_2)_+c(D_1))$
$\quad +(I,I)(b(D_1)d(D_2) - b(D_2)d(D_1))$;

$B(D_1, D_2) := (b(D_1)c(D_2) + a(D_1)_{-}d(D_2) - c(D_1)b(D_2) - a(D_2)_{+}d(D_1))$;
D1gtD2 := RatGt$(1, 1, A(D_1, D_2), 1, (T, T) \cdot (I, I), 1, B(D_1, D_2), 1)$;
D2lt1 := RatGt$((T, T), 1, (b(D_2) - d(D_2)), 1, (I, I), 1, (a(D_2)_{+} - c(D_1)), 1)$;
D2lt0 := RatGt$((T, T), 1, b(D_2), 1, (I, I), 1, a(D_2)_{+}, 1)$;
D3leD4 := RatGtEq$((I, I), 1, b(D_3), 1, (T, T), 1, a(D_3)_{-}, 1)$ and
  RatGtEq$((T, T), 1, a(D_4)_{+}, 1, (I, I), 1, b(D_4))$;
if (D3leD4) **then**
  if (D1gt0) **then**
    if (D2gt1) **then**
      if (D1lt1) **then**
        InVC := **true**; InSideVC := **true**;
      endif;
    else
      if (**not** D1gtD2) **then**
        InVC := **true**; InSideVC := **false**;
      endif;
    endif;
  else
    if (D2gt1) **then**
      InVC := **true**; InSideVC := **true**;
    else
      if (D2lt1) **then**
        if (**not** D2lt0) **then**
          InVC := **true**; InSideVC := **false**;
        endif;
      else
        InVC := **true**; InSideVC := **false**;
      endif;
    endif;
  endif;
  OnBorder := (**not**(D2lt1 **or** D2gt1) **and** InVC);
endif;
end; (∗ procedure CutsVCRev ∗)

Body of Algorithm REVOLVE

begin

1. **if** `Done` **then goto** step 3;**endif**;
2. **if** (**not** REPR) **then**

    Initialize Parameters using external input, if any, and

        $\{$ *Buf*$\langle D\rangle$.*InMessage*$|D \in$ *InDirs* $\}$, if non-empty;

        Let $D_0 \in$ *InDirs*;

        t_{in} := *Buf*$\langle D_0\rangle$.*InMessage*.t_{in};

        P := *Buf*$\langle D_0\rangle$.*InMessage*.`CoOrds`;

        I := *Buf*$\langle D_0\rangle$.*InMessage*.`Initiator`;

        T := *Buf*$\langle D_0\rangle$.*InMessage*.`Tangent`;

        Dest:= *Buf*$\langle D_0\rangle$.*InMessage*.*Dest*;

        $c_n(k)$:= *Buf*$\langle D_0\rangle$.*InMessage*.`CosNum`;

        $s_n(k)$:= *Buf*$\langle D_0\rangle$.*InMessage*.`SinNum`;

        $h(k)$:= *Buf*$\langle D_0\rangle$.*InMessage*.`Denom`;

    endif;

    CutsVCRev (*leftend*, InVC, OnBorder, InSideVC);

    `NewCn` := $c_n(k) \cdot c_n - s_n(k) \cdot s_n$;

    `NewSn` := $s_n(k) \cdot c_n + c_n(k) \cdot s_n$;

    `NewhK` := $h(k) \cdot h$;

    if InVC **then**

        REPR := **false**;

        `Done` := **true**;

        goto step 4;

    else

        REPR := **true**;

        SendDirs := $\emptyset$;

        if not InSideVC **then**

            SendDirs := $\{D|((\rho_s(\mathit{leftend}+1) - P), D) \geq \text{MinLgth}(\mathcal{L})^2/2\}$;

            if OnBorder **then**

                for each $D \in$ *SendDirs*,

                        Buf$\langle D\rangle$.*OutMessage*.t_{in} := `PathLgthDil` $\cdot$ (*leftend* + 1);

                        Buf$\langle D\rangle$.*OutMessage*.*leftend* := *leftend* + 1;

                        Buf$\langle D\rangle$.*OutMessage*.`CoOrds` := $(P + D)$;

                        Buf$\langle D\rangle$.*OutMessage*.`Initiator` := I;

$Buf\langle D\rangle.OutMessage.$`Tangent` $:= T$;
$Buf\langle D\rangle.OutMessage.$`CosNum` $:=$ `NewCn`;
$Buf\langle D\rangle.OutMessage.$`SinNum` $:=$ `NewSn`;
$Buf\langle D\rangle.OutMessage.$`Denom` $:=$ `NewhK`;
endfor;
else
for each $D \in SendDirs$,
$Buf\langle D\rangle.OutMessage.t_{in} := t_{in} + 1$;
$Buf\langle D\rangle.OutMessage.leftend := leftend$;
$Buf\langle D\rangle.OutMessage.$`CoOrds` $:= (P + D)$;
$Buf\langle D\rangle.OutMessage.$`Initiator` $:= I$;
$Buf\langle D\rangle.OutMessage.$`Tangent` $:= T$;
$Buf\langle D\rangle.OutMessage.$`CosNum` $:= c_n(k)$;
$Buf\langle D\rangle.OutMessage.$`SinNum` $:= s_n(k)$;
$Buf\langle D\rangle.OutMessage.$`Denom` $:= h(k)$;
endfor;
endif;
endif;
endif;
$t_{in} := t_{in} + 1$;
$leftend := (t_{in}$ **div** `PathLgthDil`$)$;
$c_n(k) :=$ `NewCn`;
$s_n(k) :=$ `NewSn`;
$h(k) :=$ `NewhK`;
$SendThisPass := \{D \in SendDirs \mid Buf\langle D\rangle.OutMessage.t_{in} = \texttt{PassCount}\}$;

3. **idle till** `SimClock` $= ($`CompLen` $\cdot$ `PassCount`$) - 1$;
simul send
$Buf\langle D\rangle.OutMessage$ in direction D, for each $D \in SendThisPass$;
endsend;

end.

Chapter 11

Open Problems

- Handle the multiwave, multiparticle cases with particles possibly close to one another, particles possibly doing different things from one another.

- Extend algorithm CURVE to not necessarily constant speed cases and find algorithms which are instances of the extended version for curves more general than straight lines and circles.

- Is there an interesting relation between the root lattices [CS88] and our well-represented lattices?

- Show how to compute $\mathrm{MPL}_s(\mathcal{L})$ more generally.

- Do simple physics problems with our methodology in meshes, e.g., motion of particles subjected to constant or other forces, *three* dimensional fluid flow problems, ...

- Get analogs of WAVE$'$ for TRANSLATE and REVOLVE.

- Work out, if possible, the exact physical meaning of the contents of the messages for WAVE$'$ Section 7.2.3 and then, use this to get physically natural algorithms.

- Develop practical versions of REVOLVE with various different representations of a circle other than inscribed polygonal.

- Significantly speed up simulations by various sharpenings (including sharpenings of our individual processor design in [CRS90b]).
- Simplify simulations presented herein or produce new ones by examining "regions of influence" other than Voronoi cells.
- Examine the applicability, if any, to our methodology, of superimposing the dual of a lattice on a lattice.
- Clarify more exactly the mathematical bounding relationships between MaxLinSpeed(K), for suitable K, and the least upper bound, in $\mathcal{M}(K)$, on the constant speed of virtually perfectly-timed wavefront broadcasts—especially for "nicely" shaped K, perhaps in desirably restricted lattices, and as Diam(K) gets large. The idea is to assess how good is the estimate of MaxLinSpeed(K) on the "speed of light" in "universes" $\mathcal{M}(K)$.

Bibliography

[AC$^+$86] Judy Anderson, William S. Coates, et al. FAIM-1: The General Architecture. Technical report, Schlumberger Palo Alto Research, April 1986. AI Technical Report No. 61.

[AG89] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing*. The Benjamin/Cummings Publishing Company, Inc., 1989.

[Akl89] Selim G. Akl. *The Design and Analysis of Parallel Algorithms*. Prentice-Hall, Inc., 1989.

[AS84] N. Ahuja and S. Swamy. Multiprocessor pyramid architecture for bottom-up image analysis. *IEEE Trans. on PAMI*, PAMI-6:463–474, 1984.

[Aur90] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. Technical Report B-90-9, Institute for Computer Science, Dept. of Mathematics, Freie Universität Berlin, November 1990.

[Bro81] R. Brooks. Symbolic reasoning among 3–d models and 2–d images. *Artificial Intelligence*, 17:285–348, 1981.

[Bur70] A. W. Burks, editor. *Essays on Cellular Automata*. Univ. of Illinois Press, 1970.

[BWW81] Adrian Bowyer, Philip J. Willis, and John R. Woodwark. A multiprocessor architecture for solving spatial problems. *The Computer Journal*, 24(4):353–357, 1981.

[Cod68] E. F. Codd. *Cellular Automata*. Academic Press, 1968.

[Cox73] H. S. M. Coxeter. *Regular Polytopes.* Dover Publications, 1973.

[CRS90a] J. Case, D. S. Rajan, and A. M. Shende. Optimally representing euclidean space discretely for analogically simulating physical phenomena. In *Foundations of Software Technology & Theoretical Computer Science*, volume 472 of *Lecture Notes in Computer Science*, pages 190–203. Springer Verlag, Berlin, December 1990.

[CRS90b] John Case, Dayanand S. Rajan, and Anil M. Shende. Parallel processor computer, December 1990. U.S. Patent Application submitted by SUNY, Research Foundation.

[CS88] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups.* Springer Verlag, 1988.

[DL81] M. J. B. Duff and S. Levialdi, editors. *Languages and Architectures for Image Processing.* Academic Press, 1981.

[Dye] C. R. Dyer. A quadtree machine for parallel image processing. Technical Report KSL 51, Univ. of Illinois at Chicago Circle.

[Dye81] C. R. Dyer. A VLSI pyramid machine for hierarchical parallel image processing. PRIP, pages 381–386, 1981.

[Fey82] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7), 1982.

[FHP86] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the navier stokes equation. *Physical Review Letters*, 56(14):1505–1508, April 1986.

[For81] K. Forbus. A study of qualitative and geometric knowledge in reasoning about motion. Technical Report AI.TR.615, MIT, AI–Lab, February 1981.

[For83] K. Forbus. Qualitative reasoning about space and motion. In D. Gentner and A. Stevens, editors, *Mental Models*. Lawrence Erlbaum Associates, Hillsdale, N.J., 1983.

[FT82] Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4), 1982.

[Fun85] Brian V. Funt. Problem–solving with diagrammatic representations. In Ronald Brachman and Hector J. Levesque, editors, *Readings in Knowledge Representation*. Morgan Kaufmann Publishers, Inc., 1985.

[GL87] Peter M. Gruber and Gerrit Lekkerkerker. *Geometry of Numbers*. North-Holland Mathematical Library, 1987.

[Gre72] Donald Greenspan. A discrete numerical approach to fluid dynamics. *Information Processing*, 71:1297–1304, 1972.

[Gre82] Donald Greenspan. Deterministic computer physics. *International Journal of Theoretical Physics*, 21(6/7):505–523, 1982.

[Has87] Brosl Hasslacher. Discrete fluids. *Los Alamos Science*, (15):175–217, 1987. Special Issue.

[Hei58] Werner Heisenberg. *Physics and Philosophy*. Harper & Brothers Publishers, New York, 1958.

[Hil55] E. L. Hill. Relativistic theory of discrete momentum space and discrete space–time. *Physical Review*, 100(6), December 1955.

[Hil84] W. Daniel Hillis. The connection machine: A computer architecture based on cellular automata. *Physica 10D*, pages 213–228, 1984.

[Hin79] G. Hinton. Some demonstrations of the effects of structural descriptions in mental imagery. *Cognitive Science*, 3:231–250, 1979.

[Kos80] Stephen M. Kosslyn. *Image and Mind*. Harvard Univ. Press, Cambridge, Massachusetts, 1980.

[Kos83] Stephen M. Kosslyn. *Ghosts in the Mind's Machine: Creating and Using Images in the Brain.* Harvard Univ. Press, Cambridge, Massachusetts, 1983.

[Lev85] S. Levialdi, editor. *Integrated Technology for Parallel Image Processing.* Academic Press, Inc., 1985.

[LG74] Robert A. LaBudde and Donald Greenspan. Discrete mechanics—a general treatment. *Journal of Computational Physics*, 15:134–167, 1974.

[LS87] J. Larkin and H. Simon. Why a picture is (sometimes) worth ten thousand words. *Cognitive Science*, 1987.

[Mac86] Thinking Machines. Introduction to data level parallelism. Technical Report 86.14, Thinking Machines, April 1986.

[Man] Tom Manuel. Supercomputers: The proliferation begins. *Electronics*, pages 51–56. March 3, 1988.

[Man76] M. Morris Mano. *Computer System Architecture.* Prentice-Hall, Inc., New Jersey, 1976.

[Mar84] Norman Margolus. Physics–like models of computation. *Physica 10D*, pages 81–95, 1984.

[MD84] D. McDermott and E. Davis. Planning routes through uncertain territory. *Artificial Intelligence*, 22:107–156, 1984.

[Mil89] Susan E. Miller. A survey of parallel computing 2^{nd} edition. Technical Report RADC-TR-89-68, Rome Air Development Center, June 1989.

[Min82] Marvin Minsky. Cellular vacuum. *International Journal of Theoretical Physics*, 21(6/7), 1982.

[MS88] Russ Miller and Quentin F. Stout. Efficient parallel convex hull algorithms. *IEEE Transactions on Computers*, C-37(12):1605–1618, December 1988.

[MS89] Russ Miller and Quentin F. Stout. Mesh computer algorithms for computational geometry. *IEEE Transactions on Computers*, C-38(3):321–340, March 1989.

[Neu66] J. Von Neumann. *Theory of Self–Reproducing Automata*. Univ. Illinois Press, 1966. edited and compiled by A. W. Burks.

[Niv56] Ivan Niven. *Irrational Numbers*. Number 11 in The Carus Matematical Monographs. The Mathematical Association of America, 1956.

[PD84] Kendall Preston and M. J. B. Duff. *Modern Cellular Automata: Theory and Applications*. Plenum Publishers, 1984.

[Pea77] Marshall C. Pease III. The indirect binary n-cube microprocessor array. *IEEE Transactions on Computers*, C-26(5):458–473, May 1977.

[PF+85] John Poulton, Henry Fuchs, et al. PIXEL–PLANES: Building a VLSI–based graphic system. In *Chapel Hill Conference on VLSI*, 1985.

[PU82] Kendall Preston and Leonard Uhr, editors. *Multicomputers and Image Processing: Algorithms and Programs*. Academic Press, 1982.

[Qui87] Michael J. Quinn. *Designing Efficient Algorithms for Parallel Computers*. McGraw-Hill Book Company, 1987.

[SC82] R. N. Shepard and L. A. Cooper. *Mental Images and Their Transformations*. MIT Press, Cambridge, Massachusetts, 1982.

[SC+83] R. S. Stepleman, M. Carver, et al., editors. *Scientific Computing*. North-Holland Publishing Company, 1983.

[Sei85] C. Seitz. The cosmic cube. *Communications of the ACM*, 28:22–23, 1985.

[Sel88] B. Selman. Analogues. Technical report, University of Toronto, 1988. Technical Note CSRI–47.

[SF72] R. N. Shepard and C. Feng. A chronometric study of mental paperfolding. *Cognitive Psychology*, 3:228–243, 1972.

[Sie85] Howard Jay Siegel. *Interconnection Networks for Large-Scale Parallel Processing.* Lexington Books, 1985.

[Sim63] G. F. Simmons. *Introduction to Topology and Modern Analysis.* McGraw-Hill, 1963.

[Slo71] Aaron Sloman. Interactions between philosophy and A.I.—the role of intuition and non–logical reasoning in intelligence. *Proceedings* 2nd *IJCAI*, 1971.

[SM71] R. N. Shepard and J. Metzler. Mental rotation of three–dimensional objects. *Science*, 171(4):701–703, 1971.

[Sny47] Hartland S. Snyder. Quantized space–time. *Physical Review*, 71(1):38–41, 1947.

[SS+81] H. J. Siegel, L. J. Siegel, et al. PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition. *IEEE Trans. on Computers*, C-30:934–947, 1981.

[Svo86] Karl Svozil. Are quantum fields cellular automata? *Physics Letters A*, 119(4), December 1986.

[SW86] James B. Salem and S. Wolfram. Thermodynamics and hydrodynamics with cellular automata. In S. Wolfram, editor, *Theory and Applications of Cellular Automata.* World Scientific, 1986.

[TM87] Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines.* MIT Press, 1987.

[Tof77a] Tommaso Toffoli. Cellular automata machines. Technical Report 208, Comp. Comm. Sci. Dept., University of Michigan, 1977.

[Tof77b] Tommaso Toffoli. Computation and construction universality of reversible cellular automata. *Journal of Computer and System Sciences*, 15:213–231, 1977.

[Tof84] Tommaso Toffoli. CAM: A high–performance cellular–automaton machine. *Physica 10D*, pages 195–204, 1984.

[Uhr72] L. Uhr. Layered 'recognition cone' networks that preprocess, classify, and describe. *IEEE Trans. on Computers*, C-21:758–768, 1972.

[Ung58] S. H. Unger. A computer oriented towards spatial problems. In *Proceedings of the IRE*, volume 46, pages 1744–1750, 1958.

[Vic84] Gérard Y. Vichniac. Simulating physics with cellular automata. *Physica 10D*, pages 96–116, 1984.

[Wol83] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, July 1983.

[Zus69] K. Zuse. *Rechnender Raum.* Vieweg, Braunshweig, 1969. Translated as Calculating Space, Tech. Transl. AZT–70–164–GEMIT, MIT Project MAC, 1970.