Regularity-Preserving Terrain Simplification For Faster Line-of-Sight

Alderson, Troy

UNIVERSITY OF CALGARY

Regularity-Preserving Terrain Simplification

For Faster Line-of-Sight

by

Troy Alderson

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

APRIL, 2014

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Regularity-Preserving Terrain Simplification For Faster Line-of-Sight" submitted by Troy Alderson in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE.

Dr. Faramarz F. Samavati,
Supervisor,
Department of Computer Science

Dr. Usman R. Alim,
Internal Examiner,
Department of Computer Science

Dr. Geoffrey J. Hay,
External Examiner,
Department of Geography

Date

Abstract

Three-dimensional terrain models play a key role in many applications. Line-of-sight queries, which are important operations in some applications (e.g. battlefield simulations), test whether or not two entities can see each other over the terrain. Given enough entities and a large enough terrain, computing these queries can be expensive. Terrain simplification can be used to speed up the queries, with a penalty to accuracy. To take advantage of the especially fast algorithms that exist for regular terrain models, we introduce regularity-preserving terrain simplification methods based on reverse subdivision and examine their effect on query accuracy. Furthermore, we develop a novel feature preserving reverse subdivision scheme that attempts to improve query accuracy over the pre-existing methods.

Additionally, we have examined the problem of where entities should be located after terrain simplification to maximize accuracy. Using iterative methods that attempt to maximize accuracy, we show that room for improvement exists over the standard projection method. Then, we develop practical relocation methods designed to maximize accuracy over regular simplified terrain models, the first taking a hybrid approach between projection and no relocation and the second using residual vectors to map entities onto the simplified terrain. Accuracy improvements over these basic methods can be achieved by making use of the iterative methods in a pseudo-optimization pre-processing step.

Finally, we introduce a practical line-of-sight algorithm based on hierarchies of simplified terrains that is both fast and accurate. Our approach combines two existing algorithms, using each of their strengths to achieve highly efficient line-of-sight queries in local areas.

Acknowledgements

I would be remiss if I did not express my gratitude to my family for all their love and support throughout the course of writing this thesis. My thanks and love go out to my immediate family (Kevin, Bess, and Shauna Alderson), my extended family (Aljohn, Sam, Elleiana, and Kallina Bautista; Ricardo Macabeo; and Benjie Bayon-on, Jingkie Rodriguez, and Lucas Bayon-on), and my unofficial family (Mitch Neil, Jess David, and my godson Dexter; and Emily Strong). Special mention goes to my late grandmother, Irene Fowler Macabeo. Although she was not able to witness the completion of this work, I know she was proud of me even so. She is deeply missed.

Perhaps the only thing more important to the successful completion of this thesis than the support of my family has been the support of the people in and around the Graphics Jungle lab. I am in awe everyday of your collective intelligence and expertise. Thanks go to Adam Runions, Erika Harrsion, Haleh Alemasoom, Hessam Moltaji, Javad Sadeghi, Jeff Packer, Kaveh Ketabchi, Mark Koleszar, Mark Sherlock, Nader Hamekasi, and Shannon Halbert for creating a wonderful working environment, with special mention to Ali Mahdavi-Amiri and Mahmudul Hasan for lending their time to review the thesis.

Without question, the support of Dr. Samavati has been instrumental in making all this possible. I could not have asked for a more patient and amiable supervisor. He believed in me even as I doubted myself, and helped me to grow from an inexperienced undergrad to a published researcher. For his immense generosity, I am eternally grateful.

Thanks go to C4i Consultants Inc. for motivating the problem, assisting in research and providing the large terrain models, and to Mitacs Accelerate for sponsoring our collaboration. Additional support for this research was gleaned from Dr. Samavati's NSERC and GRAND NCE funds.

The C implementation of Cláudio Silva's greedy cuts algorithm, GcTin, was downloaded

from his user page at [VisTrails, 2013]. The 120×120 height map terrain models used for some comparisons were obtained from the GcTin package.

Images of the terrain models were generated using Blender and Global Mapper. Several figures throughout the paper were created using Inkscape.

Table of Contents

List of Tables

List of Figures and Illustrations

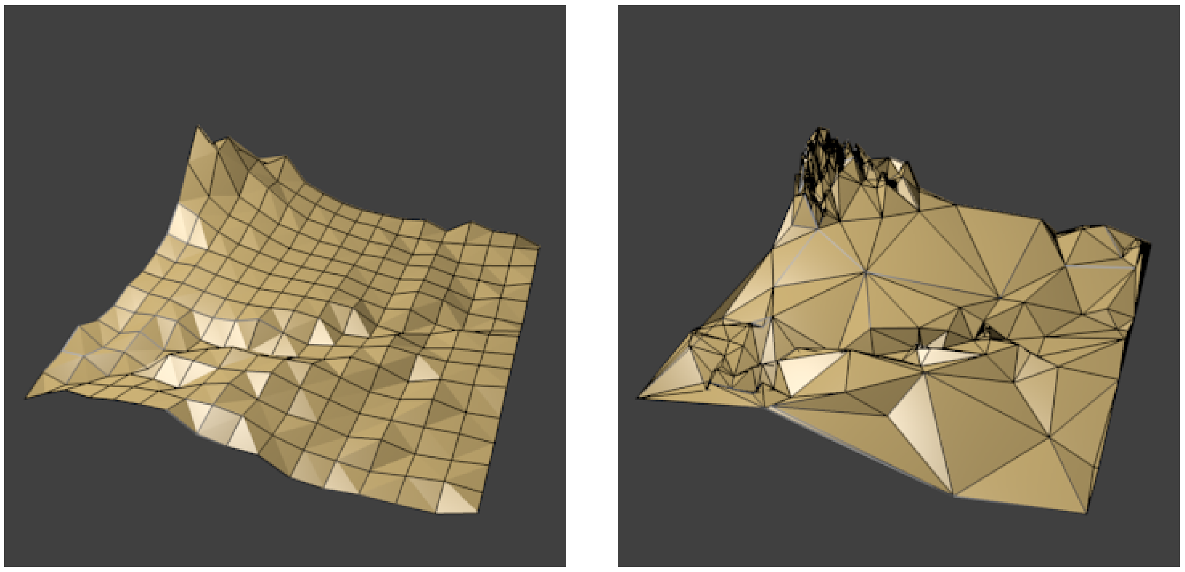
List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
LoS	Line-of-Sight
DEM	Digital Elevation Model (a.k.a. height map)
TIN	Triangulated Irregular Network
QEC	Quadric error metric-based Edge Collapse
LLSRFS	Local Least Squares Reverse Faber Subdivision
GLSRFS	Global Least Squares Reverse Faber Subdivision

Chapter 1

INTRODUCTION

Any application modeling a virtual world with a ground deals with some notion of a terrain model. Given modern three-dimensional graphics technologies, representing and processing three-dimensional terrain models have become especially important. These models are often found in two different data formats: digital elevation models (DEMs) and triangulated irregular networks (TINs) [Duvenhage, 2009]. Digital elevation models, also known as height maps, are quadrilateral (i.e. valence 4) grids of terrain elevation values, and are a well-accepted format for storing terrain data (see Figure 1.1(a)). Data from NASA's Shuttle Radar Topography Mission (SRTM) and the U.S. Geological Survey (USGS), for instance, can be found in height map format. Triangulated irregular networks are polygonal meshes composed of triangles, and are irregular in general (see Figure 1.1(b)).



(a) DEMs are regular grids of elevation values. (b) TINs are polygonal meshes composed of triangles.

Figure 1.1: An illustration of the two data formats for terrain models.

An important quality of DEMs is the quality of *regularity*. A terrain model is said to be

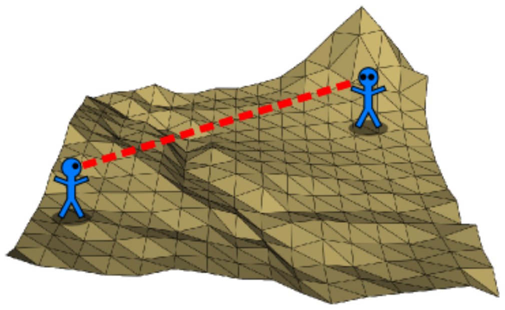


Figure 1.2: A line-of-sight query tests if two entities can see each other over a terrain.

regular if it has a grid-like structure, i.e. its interior vertices all have the same valence, and the connectivity of the boundary vertices is consistent with the connectivity of the interior vertices.

A data structure specifically designed for regular terrains offers several advantages over those for general (regular and irregular) terrains. Firstly, a regular terrain's connectivity is implicit, and need not be stored in memory. Secondly, in cases where the terrain's elevation values are evenly spaced along latitudes and longitudes, the lat/long coordinates of each elevation point can be calculated and would also not need to be stored. Finally, and quite importantly, a terrain with regular structure lends itself well to two-dimensional array indexing, allowing for faster data access than on a comparable irregular terrain.

Among computer graphics applications that make use of terrain models (such as video games and GIS), real-time simulations have high performance requirements. All computations and rendering must be done in real-time, and a high degree of accuracy is expected. *Line-of-sight (LoS)* queries compute whether or not a sight line between two entities intersects and is thus obstructed by the terrain (see Figure 1.2),[1] and they can slow down such a simulation to unacceptable levels. The problem is exacerbated when given a high number of testing entities and/or a large, detailed terrain model. These situations respectively introduce additional LoS queries to compute and additional faces/elevation values over which

[1]Although we treat entities as points, it is possible to compute visibility between 3D objects by casting multiple lines-of-sight from the observer position (e.g. an eye) to the viewed object, as in viewshed computation and supersample anti-aliasing.

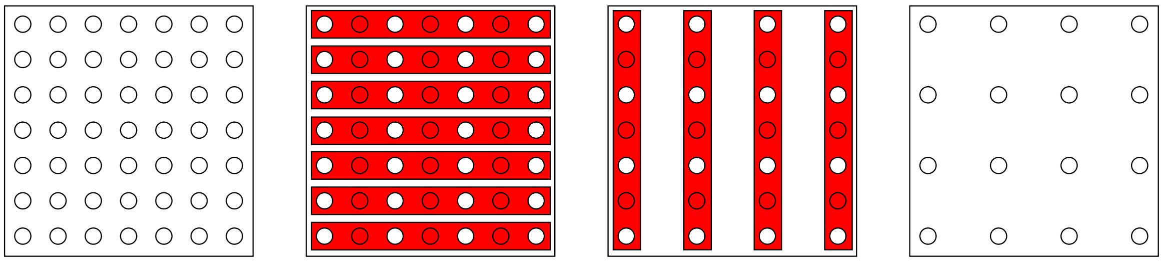
LoS queries must check for obstruction.

While LoS queries can raise an application's run time, they are a vital computation in certain applications and can not simply be discarded. In a military battlefield simulation, for example, the entities' (soldiers, tanks, planes, etc) decision-making process often relies on whether or not their target is visible. In such applications, ensuring the LoS queries can be performed in real-time is a pressing issue.

One approach that has been taken to speed up the LoS queries is terrain simplification [Ben-Moshe et al., 2002]. As noted in [Heckbert and Garland, 1997], simplifying a terrain model has many practical uses. Reduced model complexity lowers the memory requirements, rendering time, and network transmission time of a terrain. In addition, a simpler terrain results in gains in speed and efficiency for computations involving shape information; such as finite element analysis, collision detection, and shape recognition. Particularly important to our research is the gains in speed and efficiency for LoS queries resulting from terrain simplification. Simplifying the terrain model produces a speed-up in LoS query performance by a constant factor, but also introduces inaccuracies into the computation, and is particularly useful if the terrain is more detailed than is needed by the application or is simply too large to fit comfortably into main memory.

One of the questions that arises in this situation is which terrain simplification method should be used. In most applications, terrain simplification is expected to produce a combinatorially simpler model that minimizes the vertical distance error or is visually similar to the original model [Ben-Moshe et al., 2002]. Within this problem domain, however, the simplification is expected to maximize visibility test accuracy. The resulting model can be transformed and deformed in any way so long as accuracy is preserved.

Due to the fast data access for regular terrains (i.e. DEMs), LoS algorithms over DEMs have superior run times to those over irregular terrains [Seixas et al., 1999]. Hence, as regular terrains feature both low memory usage and fast visibility algorithms, it is particularly



(a) A regular grid of data points.

(b) The rows of the grid are simplified using a curve scheme.

(c) The columns are simplified using a curve scheme

(d) The resulting simplification.

Figure 1.3: An illustration of regularity preserving simplification, achieved by applying curve simplification (represented by the red rectangles) on the rows and columns of the regular terrain. The white dots represent data points/elevation values. The outlines of data points removed during simplification are shown on the rectangles.

desirable to preserve regularity in the simplification process.

An easy way to achieve regularity preservation in a terrain simplification scheme is to begin with a curve simplifying scheme and apply it to the rows and columns (u-curves and v-curves) of the regular terrain (see Figure 1.3). This is the approach we have taken. A suitable curve simplification scheme would need to ensure that curves with equal numbers of points before simplification will still have equal numbers of points after simplification, otherwise it will be impossible to connect the curve points in a regular manner. Additionally, DEM data points are often equally spaced along two dimensions, allowing the terrain to be specified almost entirely using only the data for the third dimension (i.e. the elevation values). Though not required, the curve simplification scheme should maintain or approximate this equal spacing between the data points, if it exists, for the memory savings and to avoid dealing with any spacing differences within the LoS algorithms.

A simplification scheme that satisfies these requirements is reverse subdivision. *Forward subdivision* introduces new points into a curve or surface in a deterministic manner. *Reverse subdivision* simplifies a curve or surface as an approximate inverse of this process. These two techniques can be combined into a multiresolution framework that provides a multiscale representation of a curve/surface, allowing an application to transition between several levels of resolution without loss of data.

Although simplification methods attempt to preserve the shape of the terrain, changes are inevitable. Therefore, it is also important to preserve the accuracy of the LoS queries. The accuracy of a LoS computation is the ratio of queries computed correctly to the total number of queries (see Section 2.1). After simplification, the entities should be mapped to new positions to account for changes in the terrain geometry. Standard practice is to project entities vertically onto the terrain, which makes sense for simplified terrains that minimize vertical distance to the original terrain. However, for terrain simplifications that maximize visibility test accuracy instead, projection may not be the wisest choice. Because the shape of the terrain and the entity positions together determine the LoS query results, it would seem that different simplification methods require different point relocation schemes.

1.1 Motivation

The primary motivation of our work is the use of LoS queries in real-time simulation applications, particularly military battlefield simulations. Early on, we were approached by C4i Consultants, our industrial collaborator on this work, who have been developing a constructive simulation for the purposes of training leaders and command staff.

In battlefield simulations, numerous mobile combatants (or entities) are spread over a terrain. Whether a combatant can see its target or not is vital to its decision making process, requiring many different LoS queries to be performed from each combatant to its enemies. Additionally, as combatants move the visibility information will change from moment to moment.

Hence, LoS query computations must be

- fast on a local scale, as combatants will tend to converge on key locations/battlefields (see Figure 1.4) and may have limitations on effective sight-line distance (to simulate visual acuity limitations);

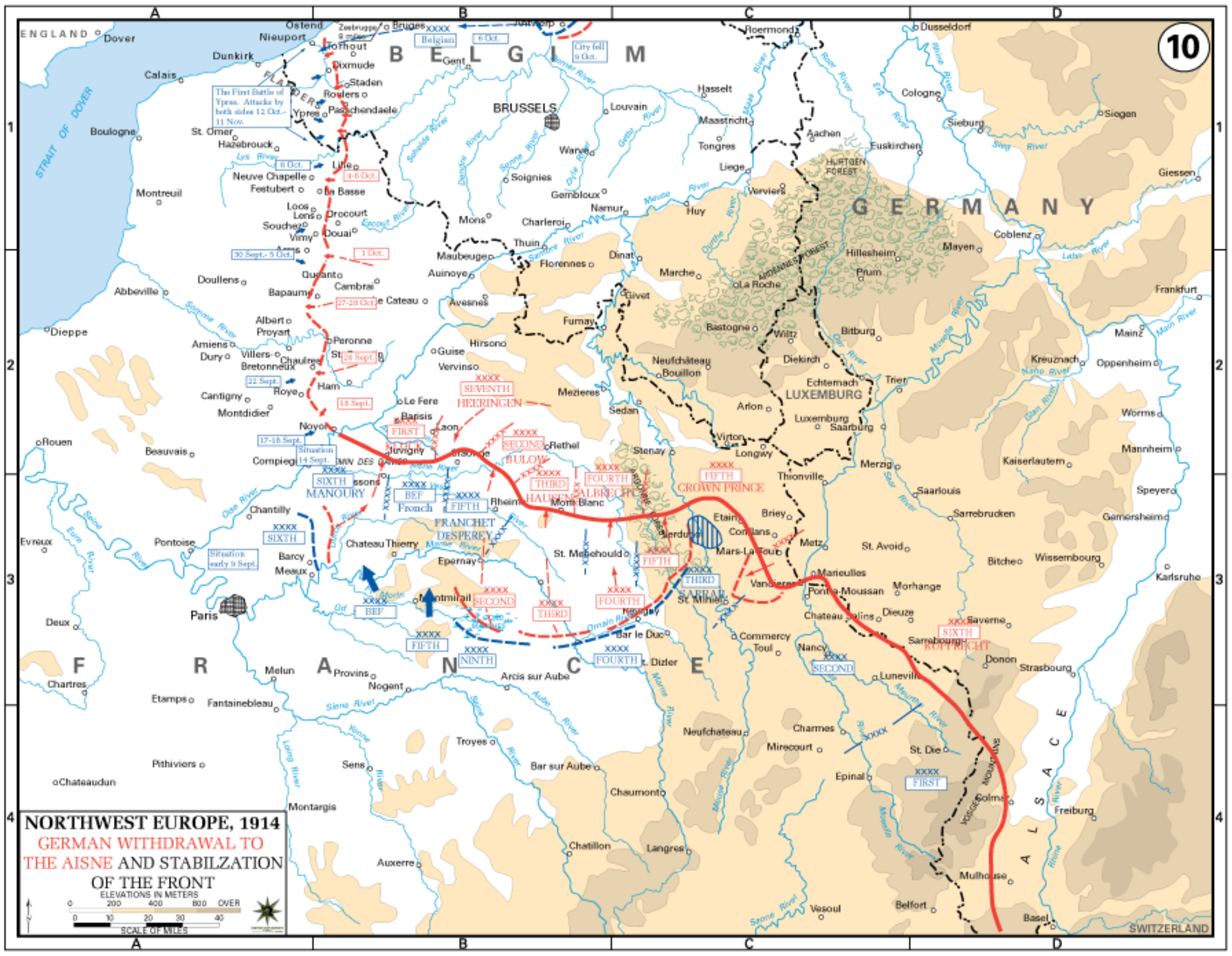


Figure 1.4: A map of the Western Front, 1914. Notice how the majority of combatants are locally concentrated along the front. This image was downloaded from [Wikipedia, 2013b] and is in the public domain.

- fast on a global scale, for when combatants are spread out or do not have such limitations (e.g. when visual acuity is enhanced by binoculars, telescopes, etc); and
- accurate, so that the query results are consistent with the simulated world.

Existing LoS query approaches were not meeting the needs of our industrial collaborator, as they were prohibitively slow. Moreover, the initial algorithm used by C4i did not attain an acceptable rate of accuracy. The need to meet these requirements was a strong motivator of our research into this area.

1.2 Problems and Challenges

Due to the presence of fast visibility algorithms over DEM terrain models, and the ubiquity of such models, we have chosen to focus our work on regular terrains and LoS query algorithms over them. An important challenge arose from this choice; namely, the problem of preserving a DEM's regularity during terrain simplification. Ensuring that a terrain remains regular after simplification places an enormous restriction on the scope of simplification methods that may be considered and any modifications that may be applied to them. At the outset of this research, it was unclear if the restriction to simplification methods that preserve regularity would have a significantly adverse effect on LoS query accuracy in comparison to more general simplification methods.

The nature of our LoS queries presented additional challenges. Unlike video games and flight simulators, in which LoS queries are usually conducted on a one-to-many basis (from the user-controlled entity to non-user entities), military simulations (the focus of our research) feature many-to-many LoS queries. Given n entities, the task of such a simulation is to compute $O(n^2)$ LoS queries in real time with a high degree of accuracy. The main challenge of our work has been to meet these requirements without placing any restrictions on the number of entities n.

The many-to-many nature of the problem reduces the utility of techniques that compute visibility with respect to a single entity (e.g. view-frustum culling and occlusion culling). Furthermore, many methods related to visibility computation operate on static scenes, and cannot efficiently handle the dynamic environment of a military simulation wherein entities are constantly changing position in an open-space environment. Pre-computing the visibility information would require the storage, for each vertex of the terrain, a viewshed map for the region of the terrain within viewing distance of the vertex (in the case of no sight line distance limitations, this would be the entire terrain). For particularly large terrains, this method is impractical.

Speeding up LoS queries using terrain simplification has some weaknesses, which presented an additional challenge during our research. A simplification offers a constant increase in speed in LoS queries; further speed increases require further simplification, and therefore further loss in accuracy. This accuracy drop affects even localized LoS queries, which may be fast enough even without terrain simplification.

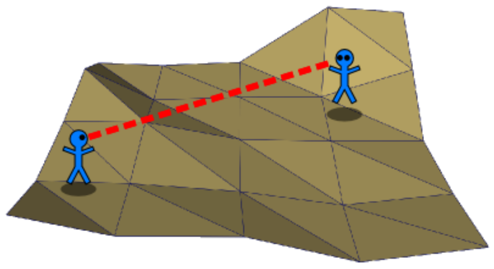
1.3 Methodology

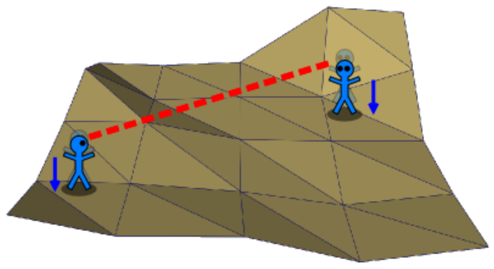
Methods studied in the field of terrain simplification have been applied to the problem of LoS query speed optimization in the work of [Ben-Moshe et al., 2002]. We have used this methodology in our work and applied *regularity-preserving* terrain simplification in the context of LoS query optimization.

We introduce reverse subdivision (from the field of curve and surface modeling; see [Samavati and Bartels, 1999]) to the visibility problem domain (see Figure 1.5(a)). In this work, we study several reverse subdivision schemes to achieve regularity-preserving terrain simplification. Taking inspiration from the ridge and valley preservation technique in [Ben-Moshe et al., 2002]'s novel simplification method, a novel reverse subdivision algorithm is developed that uses least squares error minimization to preserve the ridges and valleys of the terrain after regularity-preserving simplification.

Point relocation, as we refer to it, is the mapping of entities to new positions to improve visibility test accuracy after applying terrain simplification (Figure 1.5(b)). To relocate entities after simplification, we consider several point relocation methods and their impact on LoS query accuracy.

We first consider an optimization framework for point relocation to show that room for improvement exists over the standard projection method. Using iterative methods inspired by iterative optimizations, we estimate the optimal positions for entities, with emphasis on a theoretical perspective rather than a practical real-time implementation. Afterwards, we


(a) Regularity-preserving terrain simplification can be used to speed up queries.


(b) Point relocation adjusts the positions of the entities in the simplified space.

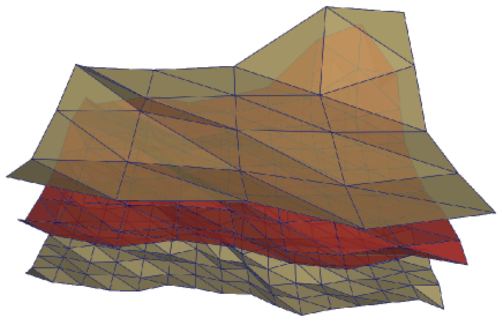

(c) Based on the expected run-time of a query, a simplified terrain can be chosen from a hierarchy such that the query will be both fast and reasonably accurate over the terrain.

Figure 1.5: We approach line-of-sight query optimization via three techniques: regularity-preserving terrain simplification, point relocation, and a hierarchical algorithm.

shift our focus from general theory to improving accuracy over simplified regular terrains in real-time. Specifically, we develop two practical relocation methods for regular terrains. The first is a simple hybrid approach of projection and no relocation. The second, which generalizes projection, uses residual vectors as relocation vectors to map entities to the simplified terrain. A pre-processing step based on our iterative estimation methods can be used to improve the methods' overall accuracy.

Unfortunately, terrain simplification and point relocation — even when paired with fast LoS query algorithms — can only offer a constant speed up. Further gains in speed are offset by drops in accuracy for all LoS queries. We identified a solution to this issue: only those LoS queries that require speed optimization would be expedited and affected by a drop in accuracy.

We formalize our solution to this problem in a LoS query algorithm that combines existing techniques to compute local visibility information quickly with a low cost to accuracy. Using terrain simplification, we generate a hierarchy of progressively simpler versions of the original terrain (Figure 1.5(c)). Each LoS query between two entities is computed over one of the terrain variants in this hierarchy based on the expected run time of the query.

1.4 Contributions

Our work introduces reverse subdivision to the context of speeding up LoS queries using terrain simplification and examines the impact on LoS query accuracy. Due to the regularity-preserving simplification behaviour of reverse subdivision, fast LoS algorithms that operate exclusively on regular terrains may still be used on terrains simplified in this way.

Additionally, we contribute a novel reverse subdivision method, *feature aware reverse subdivision*, specifically designed to improve LoS query accuracy. The method attempts to preserve features identified as critical to visibility (ridges and valleys).

Secondly, our work defines and explores the concept of point relocation. Using our iterative estimation methods, we show that projection may not always be the best choice for relocating entities onto a simplified terrain. We introduce practical relocation methods for reverse subdivided terrains that preserve LoS query accuracy well, and apply our iterative estimatation methods in a pre-processing step to improve the resulting accuracy.

Finally, we have developed a hierarchical LoS query algorithm that addresses the needs of military battlefield simulations. The algorithm combines two existing algorithms, using the strengths of each to overcome the weaknesses of the other. A hierarchy of reverse subdivided terrains allows LoS queries to be computed with a level-of-detail approach. Queries expected to be fast are computed on higher-resolution versions of the terrain, resulting in higher overall accuracy.

These contributions have been published in two separate venues. In our conference paper, *Reverse Subdivision for Optimizing Visibility Tests* [Alderson and Samavati, 2012], we present our feature aware reverse subdivision method and explore the concept of point relocation using the projection, identity, and half projection methods. Our journal paper, *Optimizing Line-of-sight Using Simplified Regular Terrains* [Alderson and Samavati, 2014], introduces our iterative estimation methods, residuals relocation method, and hierarchical LoS algorithm.

1.5 Thesis Overview

We begin in Chapter 2 with background and related work. Chapter 3 follows with an overview of the terrain simplification methods used throughout the work, including our novel reverse subdivision algorithm designed to maximize visibility test accuracy. In Chapter 4, we describe point relocation — what it is, why we have chosen to study it, and our approaches to it. A description of Bresenham's line algorithm and the min/max quad tree algorithm for computing LoS queries, and the hierarchical LoS algorithm we developed around them, may be found in Chapter 5. Finally, Chapter 6 contains our comparison results and discussion, followed by Chapter 7 with our conclusions and directions for future work.

Chapter 2

BACKGROUND AND RELATED WORK

There exists a wealth of literature on the subjects of line-of-sight computation, terrain simplification, and subdivision methods. However, there appear to be few works that combine all three. Here we provide necessary background on these three topics, including related work and a brief overview of subdivision and multiresolution. First, however, we present the various mathematical definitions used throughout the paper.

2.1 Definitions

As in the work of [De Floriani and Magillo, 1993], a terrain model T is defined as a function $T : \mathbb{R}^2 \to \mathbb{R}$ that takes points (x, y) in a rectangular region $D \subset \mathbb{R}^2$ and maps them to elevation values $T(x, y)$.

Consider a terrain model T and a set of 3D points $P = \{p_1, p_2, \ldots, p_n\}$ on T (see Figure 1.2, for example). Let $O(p)$ be the observer position for any $p \in P$, which is offset by a certain height from p. We define a line-of-sight query over terrain T to be a function, $V_T : \mathbb{R}^3 \times \mathbb{R}^3 \to \{0, 1\}$, such that

$$V_T(p_1, p_2) = \begin{cases} 1 & \text{if the sight line between } O(p_1) \text{ and } O(p_2) \\ & \text{is unobstructed by } T, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $V_T(p_1, p_2) = V_T(p_2, p_1)$. For the purposes of this work, we ignore all factors other than the terrain (such as weather, eyesight impairments and limitations, the direction in which an entity is facing, etc) that can impact visibility for either entity.

We define a LoS query algorithm to be a function A such that

$$A(T, p_1, p_2) = \begin{cases} 1 & \text{if the sight line between } O(p_1) \text{ and } O(p_2) \\ & \text{is thought to be unobstructed by } T, \\ 0 & \text{otherwise.} \end{cases}$$

Note that this definition allows for some inaccuracies in the computation of V_T using algorithm A. We define the accuracy of algorithm A acting on T and P, as in [Ben-Moshe et al., 2002], to be the ratio of the number of queries computed correctly to the total number of queries. More formally,

$$Acc(A, T, P) = 1 - \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |V_T(p_i, p_j) - A(T, p_i, p_j)|}{\binom{n}{2}}.$$

Accuracy can additionally be given in terms of the rate of true positives (resp. true negatives). This is the ratio of the number of queries correctly computed as unobstructed (resp. obstructed) to the total number of unobstructed (resp. obstructed) queries.

A terrain simplification method is a function S that accepts a terrain model T and outputs a simplified version $S(T)$. A relocation method is a function R that accepts a point $p \in \mathbb{R}^3$ and outputs a relocated point $R(p) \in \mathbb{R}^3$.

We can approximate V_T by computing $V_{S(T)}$ with point set $R(P)$. Given a LoS query algorithm A, the computation of such an approximation is implicitly performed using a new algorithm $A_{S,R}$, defined such that

$$A_{S,R}(T, p_1, p_2) = A(S(T), R(p_1), R(p_2)).$$

2.2 Line of Sight

[De Floriani and Magillo, 1993] identified three variants of the visibility problem and presented some algorithms for solving them. Given an observer position p, the three problems are given as follows:

- *Point visibility*, which determines from a set of candidate points Q the subset $Q' \subseteq Q$ visible from p (called the *discrete visible region* of p).

- *Line visibility*, which computes the horizon of the terrain with respect to p. That is, we determine the function $\phi = h(\theta)$ such that, for each azimuthal angle $\theta \in [0, 2\pi)$, ϕ is the maximum polar angle such that the ray defined by spherical coordinates (r, θ, ϕ) for all $r \in \mathbb{R}^+$ does not intersect the terrain.

- *Region visibility*, also known as *viewshed analysis*, which finds the regions of the terrain that are visible from p.

Our research concerns point visibility in the many-to-many case. Here, we consider a set of positions P and, for each $p \in P$, wish to find the subset $Q_p \subseteq P$ visible to p.

Several fast algorithms have been developed for point-to-point LoS computations over DEMs and TINs. The main challenge that any LoS algorithm must address is the question of how to efficiently identify and traverse the region(s) of the terrain that lie along the path of the sight line. Higher resolution terrains feature more faces that must be culled from the computation or traversed, increasing algorithm run-time.

Bresenham's line algorithm [Bresenham, 1965] is a well-known algorithm developed to plot a line segment on a raster grid (see Figure 2.1). The algorithm traverses and colours the raster cells along the path of the line segment by using the line's slope to increment/decrement the raster indices of the current cell. It can be optimized to use only integer operations by considering the difference in x and y coordinates between the two endpoints (which are used to calculate the slope).

The algorithm can be adapted for DEMs to traverse the path of a sight line and, rather than colour the cells, compare elevation values along that path against the sight line (as described in [Seixas et al., 1999], see Figure 2.2). The algorithm is linear in the Manhattan length of the sight line. An efficient implementation of Bresenham's line algorithm can be found in Michael Abrash's book on graphics programming [Abrash, 1997].

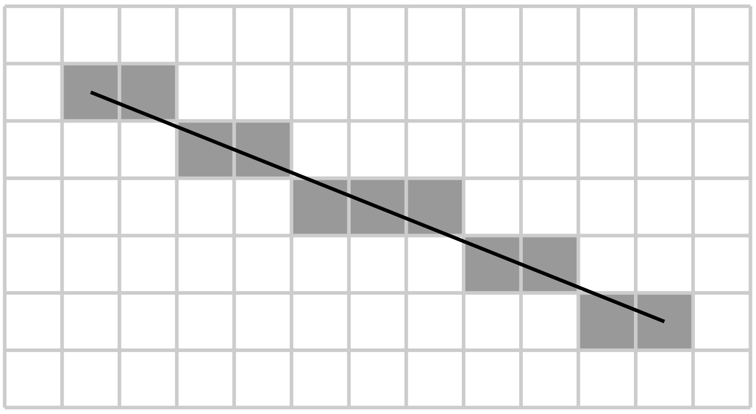
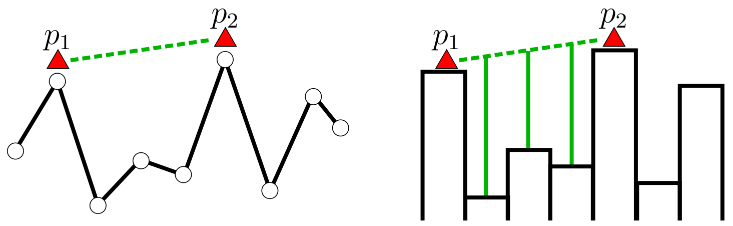


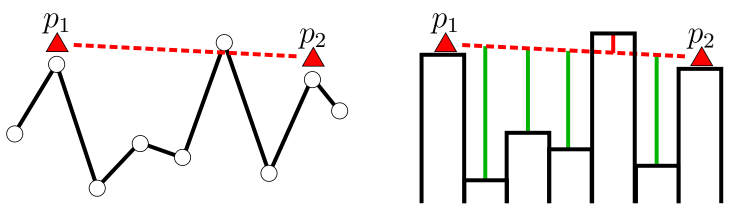
Figure 2.1: An illustration of Bresenham's line algorithm. This image was downloaded from [Wikipedia, 2014] and is in the public domain.

Spatial subdivision can be used to produce an algorithm that is asymptotically faster than Bresenham's algorithm, on average. [Duvenhage, 2009] uses a min/max quad tree to quickly cull regions of the terrain that lie completely under or over the sight line (see Figure 2.3). Each leaf node of the quad tree holds an elevation value. At each non-leaf node, the minimum and maximum elevation values of the node's descendants are stored. If the sight line over a node lies above its maximum elevation, then it does not intersect the terrain over that node and its children need not be traversed. If part of the sight line over the node lies under its minimum elevation, then the sight line definitely intersects and the result may be returned. The algorithm is logarithmic (in terms of the Manhattan length of the sight line) on average. The quad tree used by the algorithm can also prove useful for other spatial queries.

An R-Tree [Guttman, 1984], which tracks spatial data objects in a multi-dimensional space, can be used to implement an efficient LoS algorithm on TINs. Each node of an R-Tree contains a set of data pairs, each containing a pointer to either a spatial data object (in this case, the triangle faces of the terrain) or a child node paired with a bounding rectangle that encapsulates the object or child. The LoS algorithm using R-Trees described in [Seixas et al., 1999] proceeds in two steps. First, the sight line is intersected with the bounding rectangles of the tree nodes. Then, the sight line is intersected with those terrain

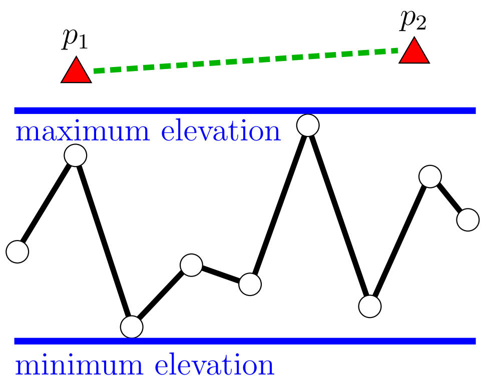


(a) LoS query returning *visible* under Bresenham's algorithm.

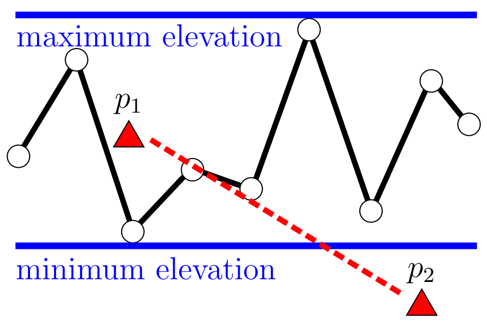


(b) LoS query returning *not visible* under Bresenham's algorithm.

Figure 2.2: An illustration of Bresenham's algorithm adapted to LoS queries. The line-of-sight is shown as a dashed line.



(a) The sight line lies completely above a region of the terrain covered by a quad tree node, hence the sight line is known to be unobstructed over this region.



(b) Part of the sight line lies below a region of the terrain covered by a quad tree node, hence the sight line is known to be obstructed.

Figure 2.3: An illustration of the min/max quad tree algorithm. The line-of-sight is shown as a dashed line.

faces whose bounding rectangles were intersected in the first step. If an intersection is found, the entities are not visible to each other.

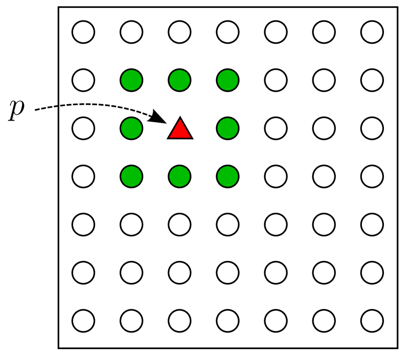
The implicit connectivity of regular models allows for data structures featuring efficient storage, addressing, and data access. This would suggest that visibility algorithms over DEMs should be computationally more efficient than visibility algorithms over TINs. In [Seixas et al., 1999], the authors compared the run times of two LoS algorithms which operate on similar principles — Bresenham's line algorithm for regular terrains and the R-tree algorithm for irregular terrains — and found Bresenham's algorithm to be more efficient in both run time and memory usage.

[Franklin and Ray, 1994] present a fast algorithm for computing viewshed analysis on regular terrains, known as Xdraw. Given an observer position p, the algorithm grows a square ring out from the observer. For each elevation point e along the perimeter of the ring, a LoS r is cast from p to e. If the slope of r is less than the greatest LoS slope encountered in the same direction as r, then e is not visible from p, else e is visible from p and the greatest slope is updated to the slope of r. See Figure 2.4 for an illustration of the fundamental idea behind the algorithm.

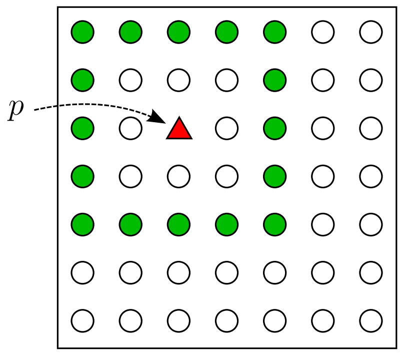
In [Andrade et al., 2011], the authors adapt Franklin and Ray's algorithm to perform viewshed queries on terrain in external memory. I/O operations on external memory form the bottleneck for the algorithm, and so were minimized to keep the algorithm fast. In general, because of the slow speed of I/O operations on external memory it is preferable for the terrain to reside completely in main memory. For particularly large terrain data sets, terrain simplification can prove useful.

2.2.1 Occlusion Culling

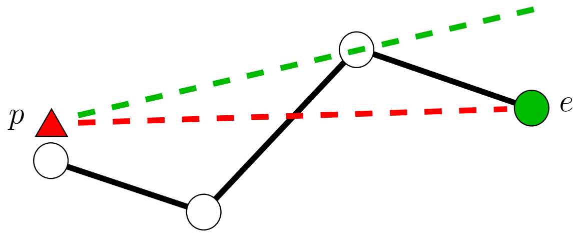
A related field of work is occlusion culling and hidden surface removal in rendering, which determines what objects are visible from a viewpoint so that time is not wasted rendering occluded objects. See [Cohen-Or et al., 2003] for a survey of occlusion culling for walkthrough



(a) A square ring of elevation values (highlighted in green) is considered about p.



(b) The square ring grows with each iteration.



(c) Point e is not visible to p because the LoS between them has a smaller slope than the greatest LoS slope yet encountered.

Figure 2.4: An illustration of Franklin and Ray's Xdraw algorithm.

applications. Notably, many of these techniques are designed to operate on static scenes with many occlusions (e.g. urban environments or building interiors), and are not suited to the dynamic and wide-open environment of simulations with many entities on a terrain model. Furthermore, several techniques are designed for use from a single viewpoint and do not scale well to LoS queries from multiple entities.

For instance, [Wonka, 2001] presents three algorithms for computing occlusion culling in an urban environment for real-time walkthroughs: the first an online algorithm that uses occluder shadows, the second a precalculation of visibility between view cells that discretize the scene, and the third an algorithm that runs parallel to the rendering pipeline which uses occluder shrinking to compute a visible set that is valid for several frames. [Funkhouser et al., 1992] use spatial subdivision, visibility analysis and a display database containing objects at several levels-of-detail to implement real-time walkthroughs through a high-resolution building interior.

In [Greene et al., 1993], the authors use a hierarchical z-buffer that combines an object-space octree with an image-space z-pyramid (i.e. z-buffer quad tree) to perform occlusion culling. This technique is useful for rendering from a single viewpoint, but is ineffective at performing LoS queries between many entities.

[Sudarsky and Gotsman, 1999] present a technique to efficiently handle mobile entities within data structures designed for static scenes (e.g. the object-space octree of Greene et al.). When an object/entity becomes hidden from view, it is replaced within the data structure by a temporal bounding volume (TBV) — a volume guaranteed to contain the entity for some length of time. Until that length of time expires or the TBV enters the view, the hidden entity is ignored, significantly reducing the number of updates to the data structure. As with other methods, this technique is useful for a single viewpoint, but the presence of multiple entities conducting LoS queries is likely to invalidate TBVs fairly rapidly.

2.3 Terrain Simplification

The study of terrain model simplification has existed for decades, and research into this area has produced a number of unique simplification algorithms. See [Heckbert and Garland, 1997] for a survey. Simplification methods are usually categorized into refinement methods, which continually refine an initial coarse approximation of the terrain, and decimation methods, which continually remove elements of the original terrain model.

Many simplification methods introduce irregularities in the general case. A sampling of several such methods are described here.

2.3.1 Refinement Methods

[Garland and Heckbert, 1995]'s *greedy insertion algorithm* is a generalization to 3D polygonal surfaces of [Douglas and Peucker, 1973]'s algorithm for approximating curves. The algorithm starts with a coarse approximation of the terrain model and iteratively refines the mesh by adding to the approximated terrain the mesh vertex that is vertically furthest from it.

Often, this process will result in long, thin triangles. Hence, to ensure mesh quality, the mesh is restricted to a Delaunay triangulation. After the addition of each new vertex, the surrounding faces are evaluated for the Delaunay condition in 2D (ignoring elevation). If the condition is not satisfied, the faces are re-triangulated by flipping edges.

Of note is that the greedy insertion algorithm will prioritize approximating high energy areas of the terrain, as vertices in these areas will generally be furthest from the approximation. Thus, low energy areas of high energy terrains are approximated relatively poorly.

The *greedy cuts algorithm* of [Silva et al., 1995, Silva and Mitchell, 1998] incrementally removes regions (or, in Silva et al.'s terminology, takes "bites") out of the yet-to-be-triangulated original terrain using three basic operations: ear cutting, greedy biting, and edge splitting. To form the simplified terrain model, each bite region is approximated with a triangle with some user-specified error tolerance ϵ. A triangle t is said to be *feasible* with respect to ϵ (and thus meets the error tolerance) if, for every vertex v of the regular terrain T that lies in the projection of t onto T, the vertical distance from v to t is at most ϵ.

The algorithm maintains a list of untriangulated simple polygons P. If there exists a triangle uniquely defined between two boundary edges of P that is feasible, then that triangle is removed from P and added to the simplification. This operation is known as *ear cutting.*

If no feasible "ear" can be found, *greedy biting* traverses the boundary edges of P and searches for a point v within P such that the triangle formed between v and the edge is feasible. This operation may divide P into two disjoint polygons.

Edge splitting is a last resort operation for cases in which ear cutting and greedy biting fail. Here, the algorithm searches for an edge of P to split, starting with the longest edge, such that a feasible triangle results. This operation is noted to produce skinny triangles, which are undesirable.

2.3.2 Decimation Methods

Iterative vertex contraction, or edge collapsing, is a mesh decimation paradigm in which edges deemed to be unimportant via some importance metric have their endpoints merged into a single point [Garland, 1999].

Edge collapse schemes differ from each other primarily in the metrics used to select edges for contraction. A shape preserving edge collapse scheme based on Garland and Heckbert's quadric error metric [Garland and Heckbert, 1997, Garland, 1999] is one such method, which we refer to as quadric error metric-based edge collapse (or QEC, for short).

In QEC, a set of planes F_v is associated with each vertex v of the terrain model, obtained by extending the faces incident to the vertex to infinity. For a given edge e, let p be the point that e will collapse to. The error resulting from collapsing the edge to p is computed as the sum of the squared distances from p to each of the planes in F_{e_1} and F_{e_2}, where e_1 and e_2 are the endpoints of e. This squared distance sum can be efficiently computed using matrix multiplication. Ideally, p should be chosen as a point that minimizes the error.

QEC carries no guarantees for boundary preservation. The effects of boundary degeneration can be reduced by applying some boundary constraints, as described in [Garland, 1999]. This is accomplished by associating an additional plane with F_v for vertices v along the terrain boundary.

For each boundary edge e, an additional plane passing through e and perpendicular to the boundary face incident to e is added to F_{e_i}, for each endpoint e_i of e. Note that this is a soft constraint, and does not eliminate boundary degradation. While boundary preservation is important to visual quality, its effect on LoS accuracy preservation is uncertain.

2.4 Subdivision, Reverse Subdivision and Multiresolution

The processes of subdivision, reverse subdivision, and multiresolution are highly important to our work. Here we provide an overview of work done in the field, followed by a brief description of these processes.

Forward and reverse subdivision have gained prominence in recent years as an important geometric modeling technique, the latter of which can be used for simplification. Together these schemes form the basis of multiresolution frameworks [Samavati et al., 2007]. Multiresolution provides a multi-scale representation of a model and has several applications, the most obvious of which is multi-scale editing.

Multiresolution has additional use in synthesis applications. In [Brosz et al., 2008], the authors use multiresolution details to synthesize terrains by example. They apply a decomposition process on a detailed terrain and then apply the resulting multiresolution details onto a target terrain, producing a terrain with the shape of the latter but the details of the former. The work of [Wecker et al., 2010] uses a similar process to synthesize new iris images.

For the sake of simplicity, we limit our discussions of subdivision and multiresolution to curve schemes. It is easy to generalize a curve simplification scheme to a regularity-preserving terrain simplification scheme, as one need only apply the curve scheme to the rows and columns (i.e. the u-curves and v-curves) of the regular terrain.

2.4.1 Subdivision

Subdivision is a family of methods that introduce new points into an existing curve, with a continuity constraint. The resulting curve can be subdivided as well, ad infinitum.

Several subdivision schemes are derived from knot insertion into B-Spline curves. Applying such a subdivision scheme to a curve's control points will return the set of control points for a curve with identical shape but additional knots at the midpoints of the original knot values. The limit curve of such schemes when repeatedly applied will be a discretized B-Spline curve. The subdivision schemes that converge to first-order, second-order, third-order, and fourth-order B-Spline curves, respectively, are Haar, Faber, Chaikin, and Cubic subdivision.

Curve subdivision schemes are simple to understand and can be easily represented in matrix notation. Given a vector of curve points c, a subdivision scheme is a linear transformation P that produces a refined vector of curve points $f = Pc$. An alternative Lindenmayer system (L-system) notation also exists, presented by [Prusinkiewicz et al., 2003], which better reflects the local nature of many subdivision schemes.

Consider, for instance, Faber subdivision. This scheme, which is named after Georg Faber [Samavati and Bartels, 1999], forms the basis for our reverse subdivision schemes, for reasons outlined in Section 2.4.4. It replicates the behaviour of knot insertion into a second-order B-Spline curve. Put simply, when applied to a curve the scheme introduces midpoints between the existing vertices, producing a new curve with C^0 continuity (see Figure 2.5).

If c is a vector containing the points of the starting curve, and f is a vector containing the points of the resulting curve, then Faber subdivision is a linear transformation P with matrix form:

$$P = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.1}$$

and $f = Pc$. Note that P is rectangular, so that P applied to c will yield a greater number

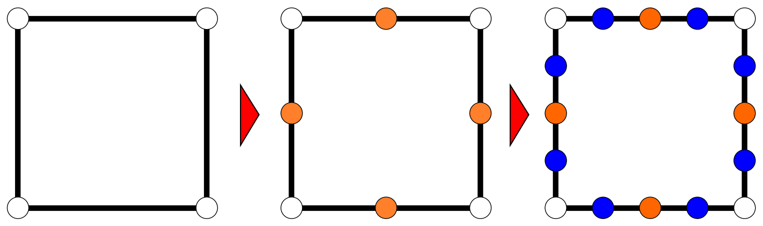


Figure 2.5: Faber subdivision applied to a curve. The shape of the curve remains unchanged; only the number of vertices is affected. Orange vertices are introduced after one application of subdivision, blue vertices after two.

of points. Note also that in this case P is sparse and banded, with a repeating local pattern. Many subdivision schemes share this behaviour and can be implemented in linear time, and can additionally be represented compactly by an ordered list of *filter values* which capture the repeating pattern of the columns of P. For Faber subdivision, these filter values are

$$\{\tfrac{1}{2}, 1, \tfrac{1}{2}\}.$$

Better known curve subdivision schemes include [Chaikin, 1974]'s corner cutting scheme (see Figure 2.6) and the interpolatory scheme of [Dyn et al., 1987]. The limit curves of these schemes (a B-Spline curve in the case of Chaikin) both have C^1 continuity. Their filter values are

$$\{\tfrac{1}{4}, \tfrac{3}{4}, \tfrac{3}{4}, \tfrac{1}{4}\}$$

for Chaikin subdivision, and

$$\{-\tfrac{1}{16}, 0, \tfrac{9}{16}, 1, \tfrac{9}{16}, 0, -\tfrac{1}{16}\}$$

for Dyn-Levin subdivision.

Subdivision schemes are not limited to curves, and can be generalized to arbitary topology surfaces. Of particular note is [Catmull and Clark, 1978]'s scheme; a generalization of cubic B-spline subdivision to surfaces. The generalization of Chaikin's corner-cutting curve subdivision scheme to surfaces was described in [Doo and Sabin, 1978]. Dyn-Levin subdivision

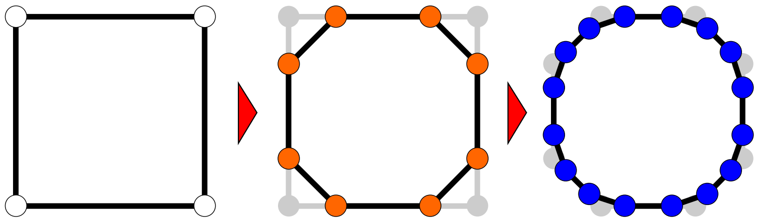


Figure 2.6: Chaikin's corner cutting scheme applied to a curve.

was extended to surfaces by [Dyn et al., 1990]. See citeCashman2012 for a survey.

2.4.2 Reverse Subdivision

Reverse subdivision attempts to reverse the subdivision process. These schemes simplify a curve or surface (i.e. remove points) such that the simplified version, once subdivided by the corresponding subdivision scheme, will yield an approximation of the original curve/surface. However, as with any simplification, high resolution details of the original are lost in the process. Hence, reversing subdivision rules on an arbitrary (i.e. not subdivided) curve/surface is not unique, and several different approaches to reverse a given subdivision scheme exist.

One of the methods used to reverse subdivision, developed by [Samavati and Bartels, 1999], is global least squares optimization. The least squares error, $||Pc-f||_2$, can be minimized by solving the overdetermined linear system $Pc = f$ for c. The optimization solution is a reverse subdivision matrix, A, such that $c = Af$. However, the matrix A must be recomputed for each size of f, which is both slow and does not reflect the local nature of most subdivision schemes.

An alternative exists: local least squares reverse subdivision [Bartels and Samavati, 2000]. In this method, the local behaviour of a subdivision scheme is captured within a local subdivision matrix P_L that acts on fixed-size c and f. Using local least squares minimization, we can obtain a local reverse subdivision matrix A_L. The repeating row pattern of A_L can be extracted to a list of *mask values* and used to construct reverse subdivision matrices A for any size of f.

For our local least squares reverse Faber subdivision (LSSRFS) scheme, we use the coefficients resulting from reversing a 3×5 local Faber subdivision matrix (refer to Equation 2.1 for the non-local matrix), which is the smallest affine local subdivision matrix that can be formed for Faber subdivision. The general form of the reverse subdivision matrix is

$$A = \begin{bmatrix} \ddots & & & & & & & & \\ & -\frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{1}{3} & -\frac{1}{6} & 0 & 0 & \\ & 0 & 0 & -\frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{1}{3} & -\frac{1}{6} & \\ & & & & & & & & \ddots \end{bmatrix}$$

and has mask values

$$\{-\tfrac{1}{6}, \tfrac{1}{3}, \tfrac{2}{3}, \tfrac{1}{3}, -\tfrac{1}{6}\}.$$

Both the global least squares and local least squares schemes apply reverse subdivision in linear time.

Reversing a subdivision scheme often results in shape exaggeration to offset any smoothing introduced by the forward scheme. After several levels of reverse subdivision, it is possible for a simplified model to bear little visual resemblance to the high-resolution model, which makes it difficult to perform tasks that benefit from a visual resemblance between the high-resolution and low-resolution data, such as multiscale editing. To address this, [Sadeghi and Samavati, 2009] introduces energy minimization to the reverse subdivision process in order to achieve *smooth reverse subdivision*. The models resulting from smooth reverse subdivision are smoother and, visually, better resemble the original models.

2.4.3 Multiresolution

While details necessary to reconstruct the original curve or surface are lost after reverse subdivision, a means of saving these details allows the subdivision process to be made to be perfectly invertible. Matrices Q and B can be found such that

$$\begin{bmatrix} A \\ B \end{bmatrix} \begin{bmatrix} P \; Q \end{bmatrix} = I$$

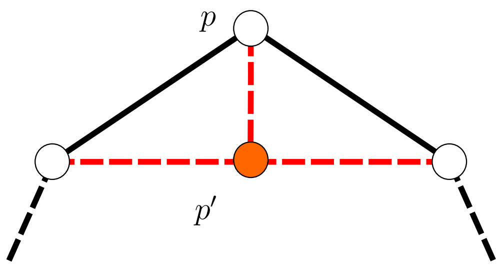
(see [Samavati et al., 2007] for a full explanation of the derivation process). Then, $c = Af$, $d = Bf$ (where d is a collection of *detail vectors*), and $f = Pc + Qd$. The process of reverse subdividing a curve/surface and determining the details is called *decomposition*; the process of subdividing a curve/surface and restoring lost details is called *reconstruction*. Together these operations form the basis of a *multiresolution framework*.

For the local Faber scheme we employ, the multiresolution matrices B and Q have the general form

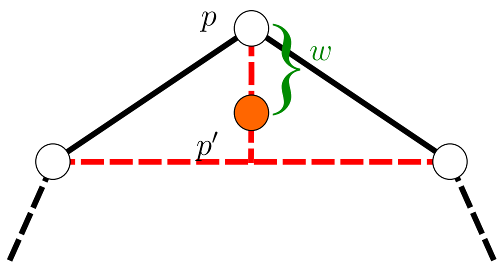
$$B = \begin{bmatrix} \ddots & & & & & & \\ & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & \\ & 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & \\ & & & & & & \ddots \end{bmatrix}$$

and

$$Q = \begin{bmatrix} \ddots & & & \\ & -\frac{1}{6} & 0 & \\ & -\frac{1}{3} & 0 & \\ & \frac{2}{3} & -\frac{1}{6} & \\ & -\frac{1}{3} & -\frac{1}{3} & \\ & -\frac{1}{6} & \frac{2}{3} & \\ & 0 & -\frac{1}{3} & \\ & 0 & -\frac{1}{6} & \\ & & & \ddots \end{bmatrix}.$$



(a) Under the Laplacian smoothing operation, a vertex p is smoothed to its new position p'.



(b) Under weighted Laplacian smoothing, a vertex p is smoothed to its new position p', according to a weight parameter w. p's neighbours remain fixed.

Figure 2.7: The Laplacian and weighted Laplacian smoothing operations.

In [Sadeghi and Samavati, 2013, Sadeghi, 2013], the authors incorporated smooth reverse subdivision into the multiresolution framework through the introduction of local fairing matrices with local inverses. Here, multiresolution matrices $\hat{A} = SA$, $\hat{B} = TB$, $\hat{P} = PS^{-1}$, and $\hat{Q} = QT^{-1}$ are employed, where S and T are local fairing matrices with banded inverses and A, B, P, and Q are the multiresolution matrices for a given multiresolution scheme.

The most promising result of this work is a weighted Laplacian smoothing operation with a local inverse. Regular Laplacian smoothing moves a point p_i on a curve to the midpoint of its neighbours p_{i-1} and p_{i+1} (i.e. $p'_i = \frac{1}{2}p_{i-1} + \frac{1}{2}p_{i+1}$). See Figure 2.7(a) for an illustration. The matrix that represents this operation is given by

$$S_L = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \cdots & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & \cdots & 0 \\ 0 & \frac{1}{2} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \frac{1}{2} & 0 & \cdots & \frac{1}{2} & 0 \end{bmatrix}.$$

S_L is singular and, thus, not invertible. By fixing one of the points, the matrix can be made invertible, but the inverse will be a full matrix.

However, it is possible to produce a fairing operation with banded inverse by fixing every other point and moving p_i towards the midpoint of its neighbours, but not all the way (i.e. $p'_i = (1-w)p_i + w(\frac{1}{2}p_{i-1} + \frac{1}{2}p_{i+1})$, where $0 \leq w < 1$ is a weighting parameter that controls the smoothness of the final curve). This operation is given by the matrix

$$F_1 = \begin{bmatrix} 1-w & \frac{1}{2}w & 0 & 0 & \cdots & \frac{1}{2}w \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{2}w & 1-w & \frac{1}{2}w & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

See Figure 2.7(b) for an illustration. When combined with a corresponding fairing operation (say, F_2), that fixes the points moved and moves the points fixed by F_1 (F_2 is a shifted version of F_1), a smoothing operation $S = F_1F_2$ is obtained that operates on all points in the curve and has a banded inverse.

2.4.4 Why Reverse Faber Subdivision?

Different subdivision and multiresolution schemes satisfy the needs of different applications. For our work, we focus specifically on reverse Faber subdivision.

Smooth subdivision schemes alter the shape of the curve/surface to which they are applied. To compensate, reverse subdivision schemes that minimize least squares error (i.e. the ones we consider) exaggerate the shape of the curve/surface. Intuitively, for best LoS accuracy preservation, we want to keep the shape of the simplified and original terrains as similar as possible. While energy minimization may be used to smooth out the result, as in [Sadeghi and Samavati, 2009], we suspect that this would negatively impact the LoS query accuracy by smoothing important features (e.g. ridges and valleys) of the terrain. Hence, we instead consider only non-smooth subdivision schemes.

Faber subdivision, which converges to a second-order B-Spline with C^0 continuity (i.e. a polyline), is one such scheme. It does not affect the shape of the curve/surface, so its least squares minimizing reverse schemes are expected to not have much of an effect on the curve's/surface's shape.

Preliminary tests comparing local least squares reverse Faber subdivision against local least squares reverse Chaikin and Cubic subdivisions indicated that, while simplifying with the Faber scheme results in lower total query accuracy, in general it features a more equitable ratio of true positives to true negatives. Local least squares reverse Dyn-Levin-Gregory subdivision is more equitable still but has even lower total accuracy. We do not assert that reverse Faber subdivision is the best choice for all situations, but use it rather to give a general sense of the performance of reverse subdivision in the context of LoS optimization.

As with other reverse subdivision methods, reverse Faber subdivided terrains do not suffer from a shrinking boundary and feature an intuitive correspondence between the rows and columns of the simplified and original terrains.

2.5 Related Work

The application of terrain simplification to optimize LoS queries has been studied previously in [Ben-Moshe et al., 2002]. In their paper, Ben-Moshe et al. present a measure of visibility similarity between a terrain and its simplifications and describe a novel method designed to maximize the measure. Put simply, this measure is defined as the ratio between the number of LoS queries computed correctly over the simplified terrain (that is, the results are the same as LoS queries over the original terrain) against the total number of queries. We use the same visibility measure in our work to assess the results of our research.

Ben-Moshe et al.'s algorithm is a modified version of the greedy insertion algorithm of [Garland and Heckbert, 1995]. The algorithm identifies a network of features important to visibility (the "ridge network"), approximates this network, and then uses greedy insertion

— with the ridge network edges as constraints — to simplify the terrain. They found their algorithm preserved LoS query accuracy better than three other tested simplification methods: the greedy insertion algorithm on which their algorithm is based, the QEC scheme from [Garland and Heckbert, 1997, Garland, 1999], and the greedy cuts method described in [Silva et al., 1995, Silva and Mitchell, 1998]. Note that both their algorithm and the comparison algorithms introduce irregularities into the terrain model in the general case.

Several previous works have achieved regularity-preserving terrain simplification through the use of image compression algorithms. In these works, the regular terrain is first converted into a grayscale image, then simplified using one of these algorithms. In the work of [Rane and Sapiro, 2001], the authors simplified terrains using lossless JPEG-LS compression. [Owen and Grigg, 2004], similarly, applied state-of-the-art JPEG-2000 compression on their terrains. [Ben-Moshe et al., 2007, Serruya, 2011] introduce a presetting stage to determine the compression algorithm parameters based on the geometric nature of the terrain prior to simplifying using the DCT (discrete cosine transform).

In this presetting stage, global and local properties of the terrain's geometry are computed. Such properties include the minimum/maximum elevation, the average elevation difference (with standard deviation) between a pixel and its local neighbourhood, and a rough approximation of the watershed. Based on these properties, the terrain is classified as either flat, dune-filled, hilly, mountainous, natural with artifacts, natural without a water flow, or artificial. The parameters of the compression algorithm are set based on this information.

In [Losasso and Hoppe, 2004], the authors propose the *geometry clipmap*, used for level-of-detail control in rendering a height map. A geometry clipmap is a hierarchical set of $n \times n$ regular terrain regions (*clip regions*) centered about a viewer position p. A clip region at level l of the clipmap represents a section of the original terrain at resolution l centered about p, and is nested within the clip region at level $l - 1$. The data for these clip regions comes either from the original terrain (for coarser levels) or synthesized using Dyn-Levin subdivision (for finer levels). Areas of the terrain closest to the viewer are rendered using the finest clip region in the clipmap, neighbouring areas are rendered using the next finest level, and so on. As the viewer position p changes, the clip regions are updated accordingly to remain centered on p.

Our hierarchical algorithm bears some similarity to geometry clipmaps; however, rather than rendering, our work is concerned with the application of a reverse subdivided hierarchy to LoS queries. In a sense, our hierarchical algorithm takes a level-of-detail approach to LoS queries. We use reverse Faber subdivision to generate coarse versions of the terrain to try and minimize shape exaggeration by reverse subdivision and preserve LoS query accuracy better.

Chapter 3

SIMPLIFICATION METHODS

As technology advances and terrains can be mapped at higher and higher resolutions, it is often necessary to simplify large terrain data sets for application use [Ben-Moshe et al., 2002]. To improve the speed and/or memory usage of LoS algorithms, it is possible to simplify the terrain and reduce the amount of data that must be traversed between pairs of points.

Simplification methods can alter the characteristics of a terrain model, and not all simplifications preserve regularity. For LoS algorithms that operate on regular terrains, it is important that the terrain simplification method be regularity-preserving. A regularity-preserving terrain simplification scheme is any terrain simplification scheme which, given an input regular terrain, outputs a simplified regular terrain. Subsampling, for instance, or several reverse subdivision methods could be used as regularity-preserving terrain simplification methods.

In this chapter, we briefly describe our approaches to simplifying terrains in the context of LoS query optimization. Several simplification methods used for comparison are described, followed by the reverse subdivision methods we have used to speed up LoS queries.

3.1 Irregular Simplification Methods

For comparison purposes, we have tested our reverse subdivision methods against several simplification schemes that do not preserve regularity. For the sake of ease, since such methods introduce irregularities to the terrain model (in general) we refer to them as *irregular simplification methods.*

The irregular simplification methods we have looked at are the same as those used for comparison in [Ben-Moshe et al., 2002]. These are

1. The greedy insertion algorithm [Garland and Heckbert, 1995]. This algorithm iteratively refines a coarse terrain approximation by adding the vertex of the original terrain that is furthest from the approximation to the simplified terrain.

2. The greedy cuts algorithm [Silva et al., 1995, Silva and Mitchell, 1998]. Each iteration of this algorithm sees a triangular "bite" region removed from the original terrain, approximated with a single triangle, and added to the terrain approximation.

3. The QEC algorithm [Garland and Heckbert, 1997, Garland, 1999], both with and without soft boundary constraints. The quadric error metric is used to assess edges' importance to terrain shape and, in each iteration of the algorithm, the least important edge is collapsed to a single point. Note that in our implementation of the algorithm, edges are collapsed to their midpoints.

Hence, our work may be compared with theirs.

3.2 Reverse Subdivision Methods

Preserving regularity through the simplification process offers several benefits. Regular terrains (height maps) are used extensively throughout the field of GIS, and so are widely supported. The implicit connectivity of regular structures allows the storage of terrain data in array-like data structures, which are easily accessed and have low memory usage beyond that needed for the raw data. Fast data access results in fast LoS query algorithms for regular terrains.

As terrain models become larger and more detailed with advances in terrain mapping technology, the run time of LoS queries deteriorates. Simplifying these models while preserving their regularity allows for continued use of fast LoS algorithms that operate on regular terrains, but with a faster run time.

Reversing a subdivision scheme allows a curve or surface to be simplified in a manner that preserves regularity. There are several ways to reverse a subdivision scheme. We have tested four variants of reverse Faber subdivision for their impact on LoS query accuracy, and compared these results against those for several irregular simplification methods.

3.2.1 Subsampling

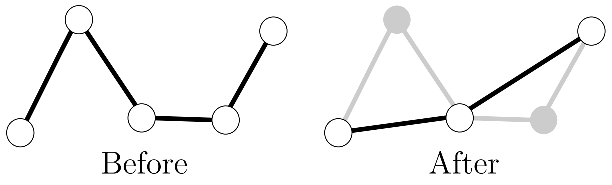


Figure 3.1: Reverse subdivision via subsampling. This scheme is not sensitive to the shape of the curve/surface.

Subsampling can be considered to be a reverse subdivision scheme. Since Faber subdivision introduces midpoints between vertices, the simplest scheme to reverse it assumes every other curve point is a midpoint and discards it. That is, we can reverse Faber subdivision by performing a downsampling of the curve points by a factor of 2.

3.2.2 Global Least Squares Reverse Faber Subdivision (GLSRFS)

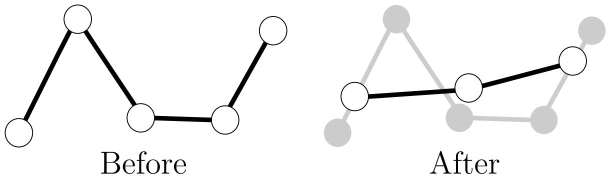


Figure 3.2: GLSRFS minimizes the least squares error between the original and the reverse subdivided curve/surface.

To better approximate the original curve, the global least squares scheme (GLSRFS) adjusts the positions of the vertices to minimize the least squares error, $||Pc - f||_2$, between the simplified and original curves. This entails solving the overdetermined linear system $Pc = f$ for c.

$$\begin{aligned} Pc &= f \\ P^T Pc &= P^T f \\ c &= (P^T P)^{-1} P^T f \end{aligned}$$

However, while the reverse subdivision matrix $A = (P^T P)^{-1} P^T$ gives a valid result, since P is banded it is often faster to solve for c directly from the linear system $Pc = f$.

3.2.3 Local Least Squares Reverse Faber Subdivision (LLSRFS)

The third Faber scheme used for this work minimizes the local least squares error between the simplified and original curves [Bartels and Samavati, 2000].

The subdivision matrix P is different for different lengths of the vectors c and f, despite the uniform structure of the underlying subdivision scheme. Analysis of subdivision schemes independent of the size of c and f is facilitated by the use of a *local* subdivision matrix, P_L, which has the same structure as P but operates on fixed-size c and f. $A_L = (P_L^T P_L)^{-1} P_L^T$ is the local reverse subdivision matrix that minimizes $||P_L c_L - f_L||_2$, where $c_L = A_L f_L$ and f_L is a vector containing a local neighbourhood of points from f. Using the (precomputed) coefficients of A_L, the reverse subdivision operation can be done efficiently in linear time.

Consider a vector, c, of n coarse points and a vector, f, of $2n$ fine points (where n is a positive integer). Let c_i and f_j ($0 \leq i \leq n-1$, $0 \leq j \leq 2n-1$) be the indexed points of c and f, respectively. The coarse points c resulting from local least squares on neighbourhoods of five fine points is given by

$$c_i = -\frac{1}{6} f_{2i-2} + \frac{1}{3} f_{2i-1} + \frac{2}{3} f_{2i} + \frac{1}{3} f_{2i+1} - \frac{1}{6} f_{2i+2}.$$

for $i = 1, \ldots, n-2$. Coarse points at the boundaries (i.e. c_0 and c_{n-1}) require special treatment. To handle these boundary cases we use the symmetric extension technique presented

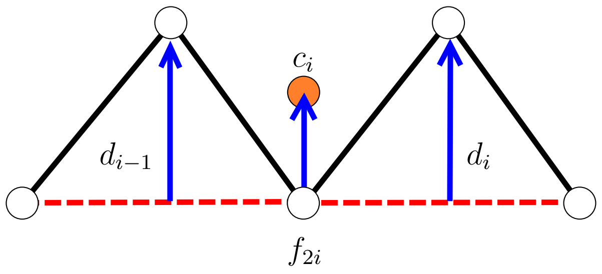


Figure 3.3: Computation of c_i under LLSRFS.

by [Hasan, 2013], based on the work of [Kiya et al., 1994]. The technique considerably simplifies the reconstruction and decomposition of boundary points and the computation of multiresolution details.

Notice that, given multiresolution detail vectors

$$d_i = -\frac{1}{2} f_{2i} + f_{2i+1} - \frac{1}{2} f_{2i+2},$$

c_i can be equivalently computed as

$$\begin{aligned} c_i &= -\tfrac{1}{6} f_{2i-2} + \tfrac{1}{3} f_{2i-1} + \tfrac{2}{3} f_{2i} + \tfrac{1}{3} f_{2i+1} - \tfrac{1}{6} f_{2i+2} \\ &= \tfrac{1}{3}(-\tfrac{1}{2} f_{2i-2} + f_{2i-1} - \tfrac{1}{2} f_{2i}) + f_{2i} + \tfrac{1}{3}(-\tfrac{1}{2} f_{2i} + f_{2i+1} - \tfrac{1}{2} f_{2i+2}) \\ &= \tfrac{1}{3} d_{i-1} + f_{2i} + \tfrac{1}{3} d_i. \end{aligned}$$

See Figure 3.3 for an illustration.

3.2.4 Feature Aware Reverse Faber Subdivision

Our novel reverse subdivision algorithm attempts to preserve terrain features using global least squares error minimization. Based on the observation that peaks and valleys in a terrain are important features that affect visibility, our algorithm identifies the critical points that define these features and uses least squares error minimization to preserve the spatial relationships between them.[1]

[1]Our definition for critical points is only one of many possibilities. So long as the critical points are taken to be a subset of f, the formulation of the augmented linear system $Pc = f$ remains unchanged.

Identification of the "critical points" is closely related to the ridge network computation that lies at the heart of the novel simplification algorithm in [Ben-Moshe et al., 2002], which is inspired by drainage network computation. In their work, an edge is said to be a ridge or a valley if the faces incident to the edge have opposite slopes. Similarly, we consider a point on a curve to be critical if its incident edges have opposite slopes (i.e. if it is a local extremum). We additionally consider the endpoints of the curve to be critical. See Figure 3.4 for an illustration. These local extrema are found with a simple direction test: if the slopes of the edges incident to a point have opposite signs, then the point is a critical point. That is, consider point f_i and its neighbours f_{i-1} and f_{i+1}, and let up be a vector in the up direction. Then f_i is a local extremum if $sgn((f_i - f_{i-1}) \cdot up) \neq sgn((f_{i+1} - f_i) \cdot up)$.

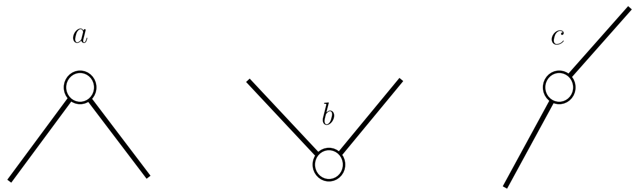


Figure 3.4: Identification of critical points. Points a and b are critical, since their incident edges have opposite slopes, whereas point c is not critical.

Let $\hat{f}_i \in f$ denote the critical points. Then, the vectors between the critical points (say $\vec{v}_i = \hat{f}_{i+1} - \hat{f}_i$), which define the spatial relationships between them, are calculated (see Figure 3.5).

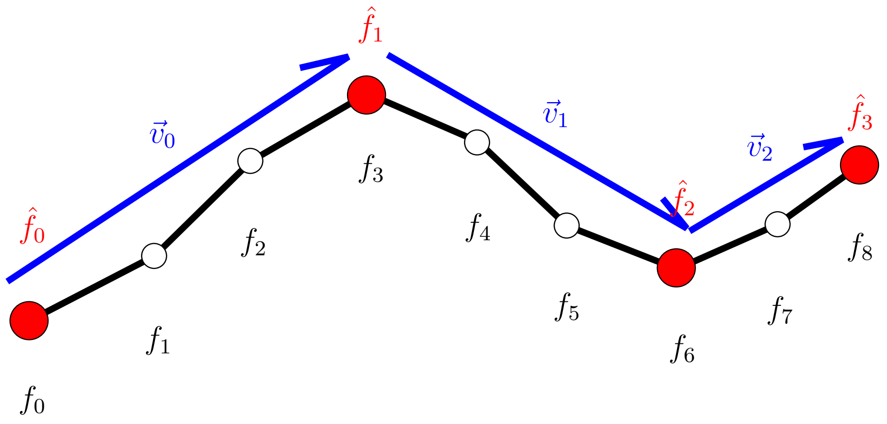


Figure 3.5: Critical points (shown in red) and the vectors between them (shown in blue).

To preserve these spatial relationships, we augment the linear system $Pc = f$ with additional constraints. For example, the linear system for the curve shown in Figure 3.5 would be

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ \vec{v}_0 \\ \vec{v}_1 \\ \vec{v}_2 \end{bmatrix}.$$

The vectors $\vec{v}_i$ are appended to the end of f and additional rows (one for each of the $\vec{v}_i$) are appended to the matrix P. These rows have exactly two non-zero entries, -1 and $+1$, for the points in coarse space that correspond to the critical points used to calculate the $\vec{v}_i$. That is, for $\vec{v}_i = f_j - f_k$, the coarse point $c_{\lfloor j/2 \rfloor}$ receives entry $+1$ in the matrix row and coarse point $c_{\lceil k/2 \rceil}$ receives entry -1, so that equations $\vec{v}_i = c_{\lfloor j/2 \rfloor} - c_{\lceil k/2 \rceil}$ are added to the linear system. Critical point collisions (i.e. when $\lfloor j/2 \rfloor = \lceil k/2 \rceil$) are discarded, resulting in the removal of rows from the augmented system.

By solving the augmented linear system, a coarse point vector c can be obtained that minimizes the error between the original and simplified curves and preserves the spatial relationships between the critical points.

Additionally, a weighting parameter can be used to control the impact of the feature-preservation. Let $w \in \mathbb{R}$ where $w > 0$ be a "feature weight" parameter. The weighted linear system for the curve shown in Figure 3.5 becomes

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 1 \\ -w & w & 0 & 0 & 0 \\ 0 & 0 & -w & w & 0 \\ 0 & 0 & 0 & -w & w \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ w \cdot \vec{v}_0 \\ w \cdot \vec{v}_1 \\ w \cdot \vec{v}_2 \end{bmatrix}.$$

In this case, we augment the linear system $Pc = f$ by appending the vectors $w \cdot \vec{v}_i$ to the end of f and append additional rows to P such that, for $\vec{v}_i = f_j - f_k$, the coarse point $c_{\lfloor j/2 \rfloor}$ receives entry $+w$ and coarse point $c_{\lceil k/2 \rceil}$ receives entry $-w$.

3.2.5 Smooth Reverse Faber Subdivision

During our research, we wondered what the impact of smooth reverse subdivision would be on LoS query accuracy. While smoothing can be important to preserving visual similarity between an original model and its simplification (as noted in [Sadeghi, 2013]), we suspect that its effect on LoS query accuracy would be negative. As LoS query results depend heavily on the features of terrain, it stands to reason that smoothing these features would negatively impact query accuracy.

The smooth reverse subdivision methodology we apply is a modified version of the weighted Laplacian smoothing proposed in [Sadeghi and Samavati, 2013, Sadeghi, 2013]. As we are not concerned with invertibility, we do not fix points. After each application of reverse subdivision, all points p_i are concurrently moved to their smoothed position $p'_i = (1-w)p_i + w(\frac{1}{2}p_{i-1} + \frac{1}{2}p_{i+1})$, where $0 \leq w < 1$.

That is, we use a reverse subdivision matrix $\hat{A} = SA$, where A is the reverse subdivision matrix of one of the above Faber schemes, and

$$S = \begin{bmatrix} \ddots & & & & & & \\ & \frac{1}{2}w & 1-w & \frac{1}{2}w & 0 & 0 & \\ & 0 & \frac{1}{2}w & 1-w & \frac{1}{2}w & 0 & \\ & 0 & 0 & \frac{1}{2}w & 1-w & \frac{1}{2}w & \\ & & & & & & \ddots \end{bmatrix}.$$

Chapter 4

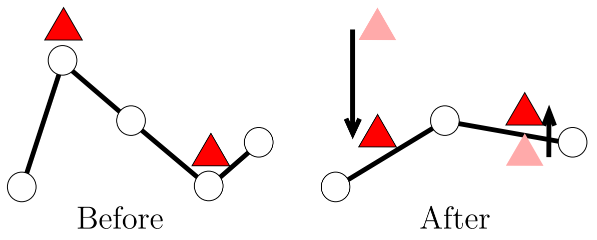
POINT RELOCATION

After terrain simplification, the shape of the terrain (which is a key factor in LoS query results) is changed. This change in the terrain requires a corresponding change in the entity positions to ensure accurate LoS queries, specified by some "point relocation" scheme.

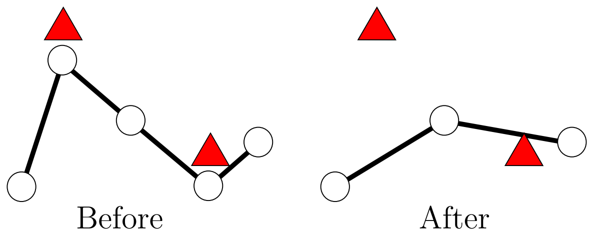
Vertically projecting entities onto the simplified terrain is an obvious choice, and well-supported by mathematical definitions of terrain models. In [Ben-Moshe et al., 2002], for instance, the projection relocation scheme is built directly into the mathematical notation. Let T be a terrain model defined over a rectangular domain $D \subset \mathbb{R}^2$ and T' be a simplification of T also defined over D. Given a point $p = (x, y) \in D$, let $p_T = (x, y, T(x, y)) \in \mathbb{R}^3$ (respectively $p_{T'} = (x, y, T'(x, y))$ be the point obtained by "lifting" p onto T (respectively T'). Two points p and $q \in D$ are said to be visible with respect to T (respectively, T') if the sight line between points p_T and q_T (respectively $p_{T'}$ and $q_{T'}$) does not intersect T (respectively, T'). What the authors refer to as lifting, we refer to as projection relocation.

We consider point relocation to be a perturbation of the entity positions to match perturbations in the terrain shape due to simplification, or alternatively as a mapping from entity positions in the original terrain space to the simplified terrain space. We suspect that the performance of a point relocation method depends a great deal on the choice of simplification method. A move to simplifications that maximize LoS query accuracy rather than minimize vertical distance error suggests a move away from projection relocation.

In this chapter, we present our approaches to studying point relocation. We first consider point relocation from a theoretical standpoint to establish that there exists room for improvement over projection relocation. We then present a simple relocation method that provides a middle ground between projection relocation and no relocation (i.e. identity).



(a) Illustration of the projection function.



(b) Illustration of the identity function.

Figure 4.1: Illustration of the behaviour of our comparison relocation methods, applied after terrain simplification. Entities are shown as triangles before relocation (in pink) and after (in red).

Afterwards, we generalize projection to a relocation method designed for use with regular simplified terrains and introduce a modification to the scheme in order to improve accuracy further. Our final discussion describes a pre-processing step that attempts to optimize the relocation vectors.

4.1 Comparison Methods

For comparison, our relocation methods are tested against the following two standard point relocation methods.

The **projection function** is the commonly accepted relocation method projecting all entities vertically onto the simplified terrain. This method makes sense particularly for simplification methods that minimize vertical distance error, and is well supported by mathematical definitions of terrain models.

The **identity function** is a "do nothing" relocation scheme that leaves the entities in their original locations. We use it to establish an accuracy-preservation baseline for the relocation methods.

It remains an open question whether or not it is possible to preserve LoS query accuracy better than the already accepted method of projection. Hence, we have pursued a path based on optimization in order to determine whether or not room for improvements in accuracy exist over these two methods.

4.2 Point Relocation by Iteration

To maximize the visibility test accuracy after simplification, a key component in the test results, i.e. the entity positions, may be optimized. We wish to find a relocation function R such that the accuracy of a LoS algorithm A on relocated points $R(P)$ over a simplified terrain $S(T)$ (i.e. $Acc(A_{S,R}, T, P)$) is maximized. That is, we wish to solve an optimization problem:

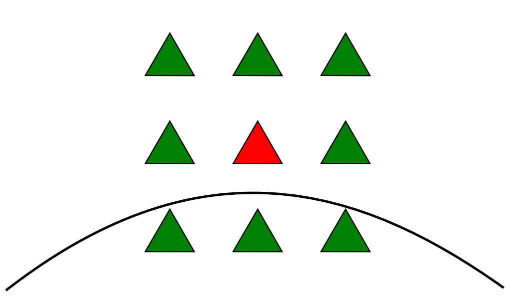
$$\min_{R} \sum_{i \neq j} |V_T(p_i, p_j) - A(S(T), R(p_i), R(p_j))|. \tag{4.1}$$

Note that the presence of V_T in Equation 4.1 implies the visibility test results are already known, meaning that direct use of such an optimization is impractical for most purposes. Additionally, a closed form solution to the problem does not appear to exist.

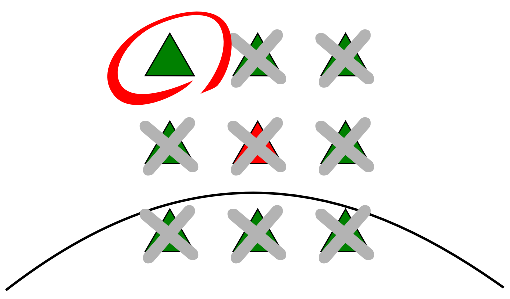
However, the optimal solution can be estimated using discrete, iterative methods. Performing this estimation is quite slow and far below real-time levels. We describe here our approaches to estimating the optimal solution.

4.2.1 Estimation Methods

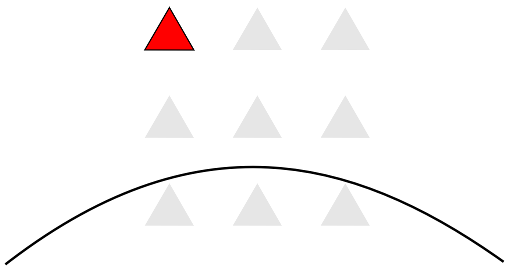
Our discrete, iterative approach to estimating the optimal entity positions is inspired by iterative numerical optimization. In iterative optimization, an initial guess for the optimal solution is provided. Then, the guess is refined iteratively until it converges to the optimal solution. Similarly, we begin with a guess for the optimal entity positions and iteratively



(a) The entity's local neighbourhood of candidate positions (shown in green) is discretized.



(b) The position that produces the highest accuracy is determined.



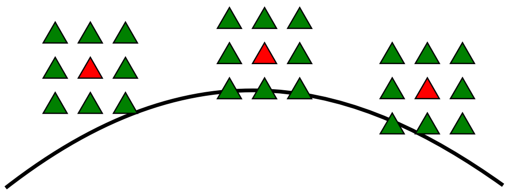
(c) The entity is relocated.

Figure 4.2: Illustration of our iterative method in 2D. The entity's current position is highlighted in red. The terrain is represented by the curved line.

adjust positions to maximize accuracy. See Figure 4.2 for an illustration.

In our approach, the entities' original positions are used as an initial guess for the optimal entity positions. In each iteration, each entity's local neighbourhood is discretized into a set of candidate positions, evenly spaced in a grid pattern. From each entity's candidate positions, sight lines are cast to the other entities' current position and the visibility test accuracy is calculated. The entities are then moved to the candidate position that produces the highest accuracy, with preference given to those candidate positions that lie closest to the original position. This process is repeated iteratively until no further accuracy improvement can be found, or an iteration threshold is passed.

As we do not conduct an exhaustive search of the entire space, the optimal solution is not guaranteed to be found, and there can be considerable difference between the results of different implementations. We have tested four implementations of this general method to determine which implementation will produce the best results.


(a) Local neighbourhoods discretized with a constant distance.

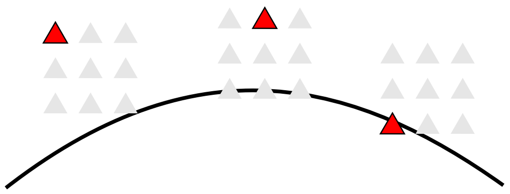
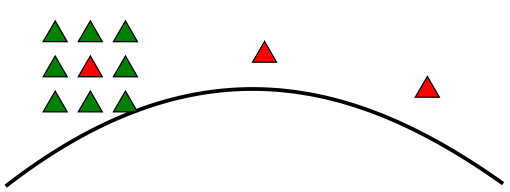

(b) Entities relocated all at once.

Figure 4.3: Illustration of the Jacobi approach in 2D.


(a) Local neighbourhoods discretized with a constant distance.

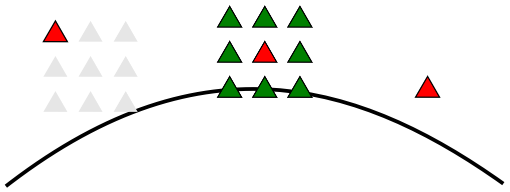
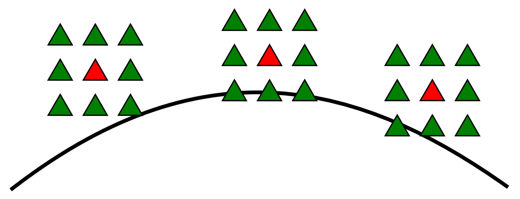

(b) Entities relocated one at a time.

Figure 4.4: Illustration of the Gauss-Seidel approach in 2D.

Jacobi Iterative Estimation: Our first implementation of the general method is inspired by Jacobi's iterative method for solving linear systems of equations. The local neighbourhood of each entity is discretized using a constant distance. Entities in this method are relocated only after the best candidate position has been determined for every entity. See Figure 4.3 for an illustration.

Gauss-Seidel Estimation: Our second implementation takes inspiration from the Guass-Seidel iterative method for solving linear systems of equations. As with the above implementation, candidate positions are calculated using a constant distance. In this method, however, each entity is relocated immediately after its best candidate position is identified, and this new entity position is used in subsequent calculations to determine test accuracy. See Figure 4.4 for an illustration.

Decreasing Distance Estimation: Our decreasing distance implementation is inspired by the process of simulated annealing. After each iteration, the distance used to calculate candidate positions is decreased by a constant factor. Entity relocation occurs as in the Gauss-Seidel method. See Figure 4.5 for an illustration.


(a) Local neighbourhoods discretized with a constant distance.

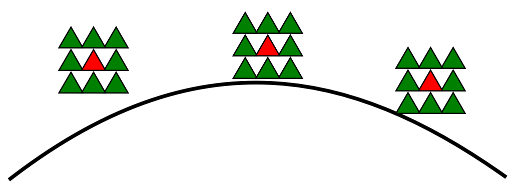
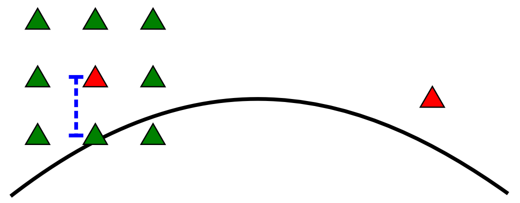

(b) Distance descreased with each iteration.

Figure 4.5: Illustration of the descreasing distance approach in 2D.


(a) Local neighbourhoods discretized with the distance to the terrain.

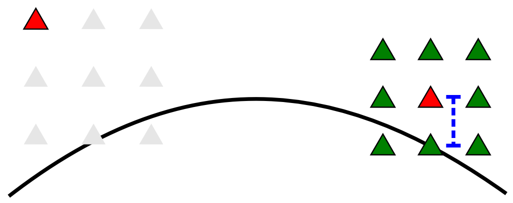

(b) Entities relocated one at a time.

Figure 4.6: Illustration of the projection distance approach in 2D.

Projection Distance Estimation: Our final implementation of the general method is based on the commonly-accepted method of projecting the entity positions vertically onto the simplified terrain. Here, the distance used to discretize the local neighbourhood is taken to be the vertical distance between the original entity position and the simplified terrain (i.e. the distance the entity would be moved were it to be projected). Entity relocation occurs, again, as in the Gauss-Seidel method. See Figure 4.6 for an illustration.

See Section 6.1.4 for accuracy results for our estimation and comparison methods paired with several terrain simplification schemes. Our results indicate that there does exist room for accuracy improvement over the comparison relocation methods.

However, as previously noted these methods cannot be used directly as relocation schemes in a real-time application, although it is possible to use the methods in a pre-processing step. We present two practical point relocation schemes we have developed in the following two sections; first, one that combines projection and identity, and a second that generalizes projection. Then, we describe the use of our iterative pseudo-optimization in a pre-processing step that attempts to optimize the relocation vectors.

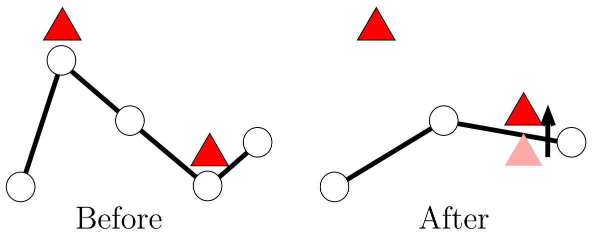


Figure 4.7: Illustration of the half projection relocation function.

4.3 Point Relocation by Half Projection

Our first practical relocation scheme that attempts to improve LoS accuracy over the projection method is a hybrid of identity and projection we refer to as *half projection*, so called because the entities that lie in the half-space beneath the simplified terrain are projected up onto the terrain under this relocation, whereas entities above the simplified terrain are left in place. See Figure 4.7 for an illustration.

The method is intended to address perceived flaws with the identity and projection methods. The unexpectedly strong accuracy performance of the identity function, particularly on LLSRFS terrains, lends support to its use for point relocation. However, the allowance of entities beneath the terrain is potentially problematic, as these entities are rendered effectively invisible to all entities above the terrain.

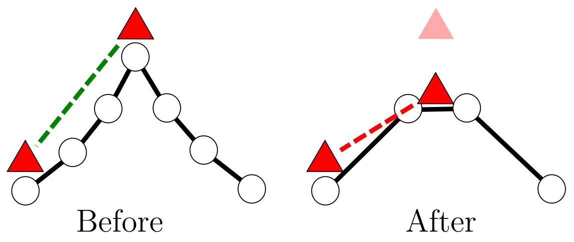


Figure 4.8: The flattened sharp feature problem. An entity from atop a sharp feature cannot see a formerly visible entity after being projected to the flattened sharp feature.

For entities above the simplified terrain, the problem of flattened sharp features emerges (see Figure 4.8). If an entity atop a sharp feature (which can see many entities below it) is projected to a flat face resulting from the simplification of the sharp feature, then that face can obstruct the entity's view of lower entities.

Taken together, these observations motivate us to leave entities above the simplified terrain in their original positions and project entities that lie below the simplified terrain. The result is a relocation scheme that favours false positives more strongly than either projection or identity, which may prove desirable for some applications.

In a sense, the identity, projection, and half projection relocation methods impose constraints on the set of entity positions P. Whereas identity is unconstrained, projection relocation imposes the initial condition constraint that "all entities must lie on the terrain". Our proposed half projection method satisfies the constraint that "no entity may lie beneath the terrain".

Naturally, these relocation methods are not the only ones that satisfy these constraints. In the next section we present a generalization of projection relocation that additionally satisfies the "all entities must lie on the terrain" constraint.

4.4 Point Relocation by Residual Vectors

Consider a curve of fine points f after a single application of reverse subdivision, resulting in a curve of coarse points c. Notice that since forward Faber subdivision does not change the shape of any curve to which it is applied, all changes in geometry between the original curve and the simplified curve are encapsulated by the residual vectors $r = Pc - f$. To compute these residuals directly from the given curves, it is trivial to note that

$$r_i = \begin{cases} c_{\frac{i}{2}} - f_i & \text{if } i \text{ is even} \\ \frac{1}{2}c_{\lfloor \frac{i}{2} \rfloor} + \frac{1}{2}c_{\lceil \frac{i}{2} \rceil} - f_i & \text{if } i \text{ is odd.} \end{cases}$$

Hence, by using the residual vectors as translation vectors for relocation, we can project entities from the original curve onto the simplified curve. (Notice that $f+r = f+(Pc-f) = Pc$, which are points on the simplified curve.) That is, rather than a blind vertical translation, we can use these residual vectors to better project entities onto the simplified curve in a geometrically-informed manner based on the simplification.

Our word choice here of “project” is quite deliberate, as this relocation scheme is a generalization of vertical projection. Residual vector relocation responds to changes in (x, y, z) coordinates between f and Pc, whereas projection relocation does not. If the x and y coordinates for the points in f and Pc are the same, then residual vector relocation acts as a vertical projection.

Generalizing to n steps of subdivision is a simple matter. Let c^k (for $k = 0, \ldots, n-1$) be the vector of coarse points after $n-k$ applications of reverse subdivision on f. Then the residual vectors are given by $r = P^n c^0 - f$, or equivalently,

$$r_i = (1-w) \cdot c_{\lfloor \frac{i}{2^n} \rfloor} + w \cdot c_{\lceil \frac{i}{2^n} \rceil} - f_i$$

where $w = \frac{i \bmod 2^n}{2^n}$.

Further generalization to regular surfaces (e.g. height map terrains) results in the following residual vectors:

$$\begin{aligned} r_{i,j} &= (1-w_j)(1-w_i) \cdot c_{\lfloor \frac{i}{2^n} \rfloor, \lfloor \frac{j}{2^n} \rfloor} + (1-w_j) w_i \cdot c_{\lceil \frac{i}{2^n} \rceil, \lfloor \frac{j}{2^n} \rfloor} + \\ & \quad w_j (1-w_i) \cdot c_{\lfloor \frac{i}{2^n} \rfloor, \lceil \frac{j}{2^n} \rceil} + w_j w_i \cdot c_{\lceil \frac{i}{2^n} \rceil, \lceil \frac{j}{2^n} \rceil}) - f_i \end{aligned}$$

where $w_i = \frac{i \bmod 2^n}{2^n}$ and $w_j = \frac{j \bmod 2^n}{2^n}$.

Special consideration needs to be given to what should be done with entities that do not lie at one of the vertices of the original curve/surface (i.e. that are not in f). Let p be the position of such an entity, lying on a edge (in the curve case) or face (in the surface case) with vertices q_i (for some i). For the sake of affineness, we desire that $R(p)$ have the same barycentric coordinates with respect to the $R(q_i)$ as p has with the q_i. Hence, we take the relocation vector $R(p) - p$ to be a barycentric combination of the relocation vectors corresponding to the q_i.

Together with the residual relocation vectors, this defines a vector field over a curve/surface from which we can assign relocation vectors to every entity in real time.

4.4.1 Multiresolution Details and Residual Vectors

There is a very close relationship between the residual vectors and multiresolution details. In fact, the residual vectors can be computed directly from the details. See Figure 4.9 for a visual illustration. Hence, given a reverse subdivided curve/surface and associated details, we do not need to reconstruct the original curve/surface to compute the residuals.

The process to compute these residuals from the details is conceptually simple: using the given detail vectors, we apply the reconstruction process on a trivial (all zero) vector field and negate the results.

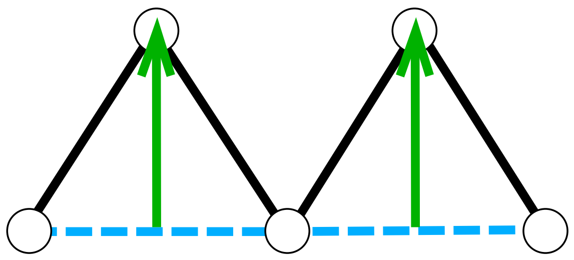
To see why this works mathematically, consider n steps of reverse subdivision on a curve. Let c^k and d^k be the vectors of coarse points and details at various resolutions, respectively, after $n-k$ reverse subdivisions. Then

$$\begin{aligned} f &= Pc^{n-1} + Qd^{n-1} \\ &= (P^2c^{n-2} + PQd^{n-2}) + Qd^{n-1} \\ &= ((P^3c^{n-3} + P^2Qd^{n-3}) + PQd^{n-2}) + Qd^{n-1} \\ &\vdots \\ &= P^nc^0 + P^{n-1}Qd^0 + \ldots + PQd^{n-2} + Qd^{n-1}. \end{aligned}$$

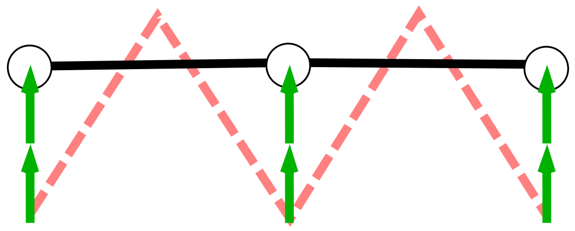
After some formula rearrangement, we see that the residuals vectors can be computed as

$$r = P^nc^0 - f = -(P^n\vec{0} + P^{n-1}Qd^0 + \ldots + PQd^{n-2} + Qd^{n-1}),$$

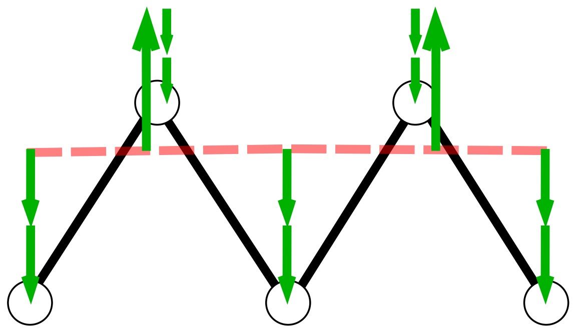
which are precisely the negations of the detail vectors applied to the reconstruction of a trivial terrain.



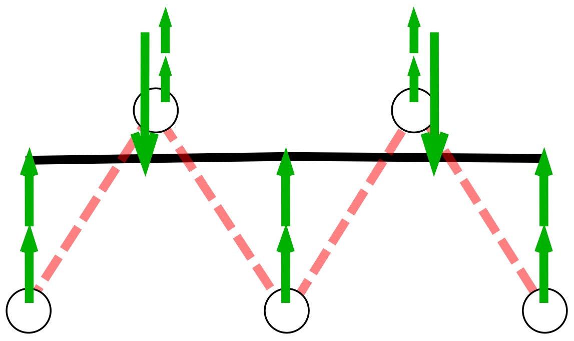
(a) Computation of the details (shown in green) from the original terrain (shown in black).



(b) The original terrain (red) is simplified to a new version (black) using linear combinations of the details (green).



(c) The original terrain (black) can be reconstructed from the simplified terrain (red) using linear combinations of the details (green).



(d) The reconstruction details (green) can be reversed to map points onto the simplified terrain (black).

Figure 4.9: The negation of the detail vectors used for reconstruction can be used to map entities from the original terrain to the simplified terrain.

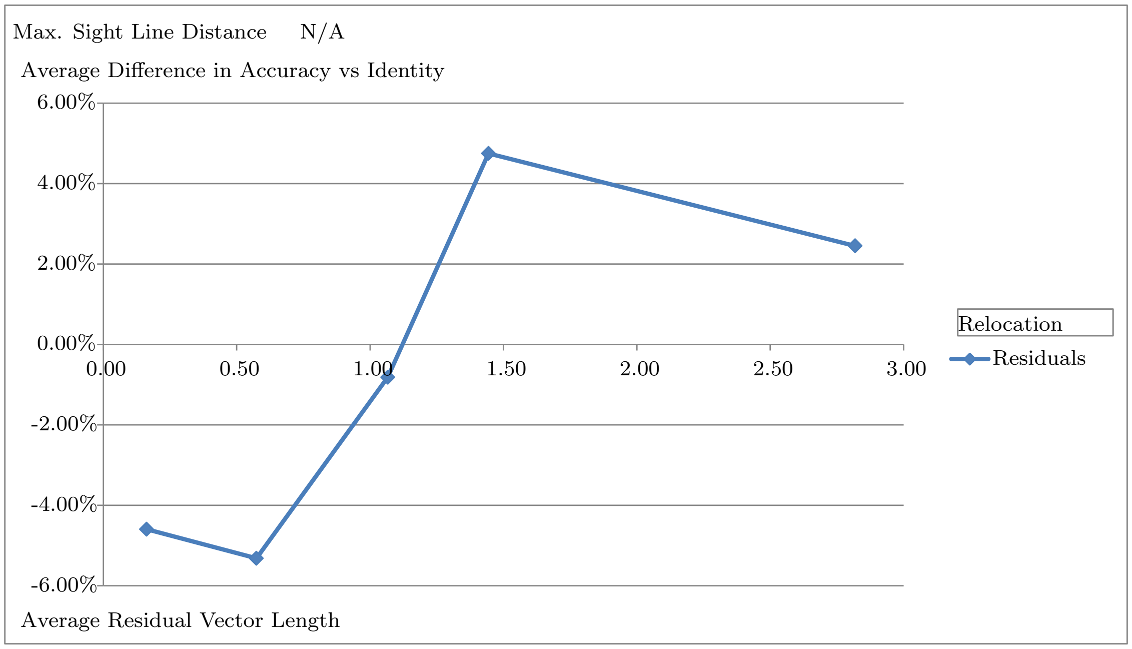


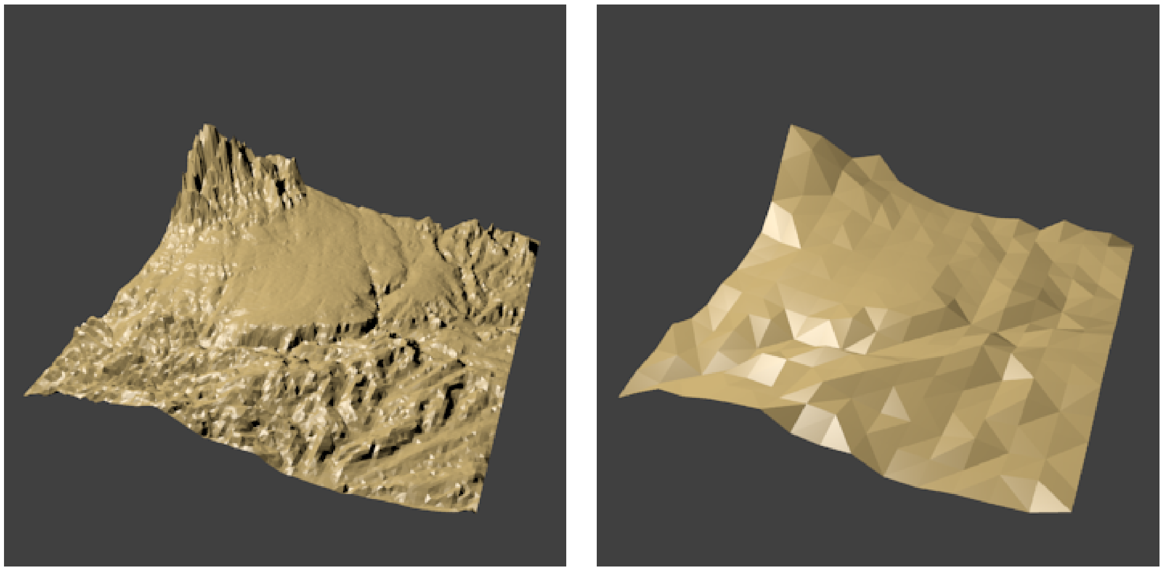
Figure 4.10: The average length of residual vectors plotted against the difference in accuracy (after simplification by LLSRFS) between the identity and residuals relocation functions. Each data point corresponds to a tested terrain model. Notice that as the average length of residual vectors increases, the accuracy of residuals relocation improves in comparison to identity.

4.4.2 Scaled Residual Vector Relocation

We realized that, under certain conditions, identity preserves LoS accuracy better than the proposed residuals relocation method. Our gathered results indicate that this occurs when the average length of the terrain's residual vectors is low (see Figure 4.10). This makes sense, as short residual vectors imply that geometric changes between the simplified and original terrains are small, therefore the simplified terrain is close enough to the original that identity performs well (see Figures 4.11 and 4.12 for example terrains).

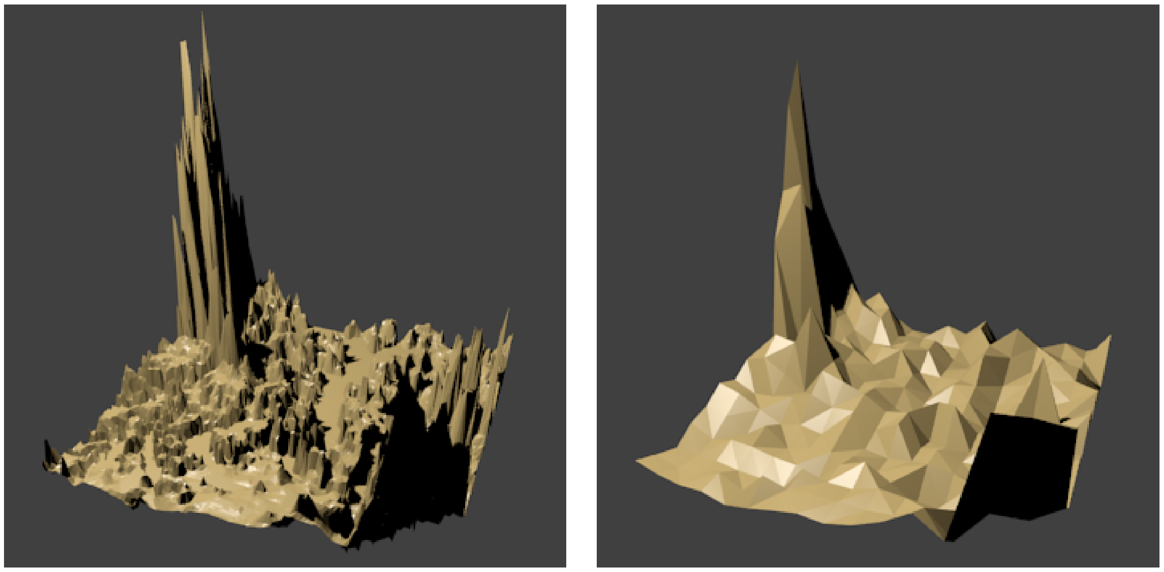
If identity is suspected to perform best over a section of the terrain, then we want the relocation over that section to behave more like identity. We sought to implement this behaviour on a face-by-face basis by scaling down relocation vectors if the vectors' average length falls below a certain threshold.

Hence, for each face of the simplified terrain we compute the average length of the residual



(a) Original terrain mesh. (b) Simplified terrain mesh.

Figure 4.11: The “Eagle Pass” terrain, which features a small average residual vector length. The simplified and original terrains are similar enough to each other that identity performs well.



(a) Original terrain mesh. (b) Simplified terrain mesh.

Figure 4.12: The “Seattle” terrain, which features a large average residual vector length. Geometry changes between the simplified and original terrain are quite significant, resulting in poorer performance for identity.

vectors associated with the entities corresponding to the face. Then, if the computed average falls below a given threshold, we scale the residual vectors over that face.

Consider a simplified terrain face F. Let $R(p)$ denote the unmodified residuals relocation function, $r(p) = R(p) - p$ denote the unscaled residual vector for a given point p, and $avg(F)$ denote the average length of unscaled relocation vectors over F. Then, the scaled residuals relocation function $R'(p)$ acting on p (the position of an entity associated with face F) is given by

$$R'(p) = \begin{cases} p + \frac{avg(F)}{threshold} r(p) & \text{if } avg(F) < threshold \\ p + r(p) & \text{otherwise} \end{cases}$$

where $threshold \in \mathbb{R}$.

Note that the average length of residual vectors is dependent on the scale of the terrain models, and so the value of *threshold* will vary between applications. The problem of determining a threshold value is non-trivial and would benefit from normalization of the lengths and/or an algorithmic approach. The threshold value used in our tests was determined from Figure 4.10 in an ad-hoc manner (see Section 6.1.5).

In the next section, we present an algorithmic approach that determines scaling factors for each relocation vector, rather than a length threshold. How to determine such a threshold value remains an open question.

4.5 Pseudo-Optimization Pre-Processing Step

To improve the accuracy preservation quality of our practical relocation methods, we propose using our estimation relocation technique in a pre-processing step after the initial relocation vectors have been computed. This modification attempts to optimize the relocation vectors, but it is not a true optimization. Hence, we refer to this pre-processing method as *pseudo-optimization*.

Our iterative estimation methods, as described, consider a set of particular entity positions and output a corresponding pseudo-optimized set of entity positions. We wish to generate a set of relocation vectors that can be used with any set of entity positions and output a corresponding set of pseudo-optimized entity positions. To accomplish this we can consider a set of entities, one for each relocation vector, and take the pseudo-optimized relocation vectors to be the differences between the original entity positions and the final entity positions after the estimation.

However, since the number of relocation vectors in the vector field is equal to the number of vertices of the original terrain, it is impractical to pseudo-optimize all of them. Hence, we pseudo-optimize only a subset of these–namely those that correspond to vertices of the *simplified* terrain. Starting from a vector field of relocation vectors as an initial solution, we use the results of the pseudo-optimization to perturb the relocation vectors to an approximation of the optimal solution.

Relocation vectors fluctuate noticeably in direction and size, hence it makes little sense to blend them with a perturbation vector computed from a different relocation vector. For this reason, we find pseudo-optimal scaling factors for each relocation vector rather than pseudo-optimal perturbation vectors. This restricts the space in which we discretize candidate positions for each entity to lie in the direction of its initial relocation vector. Hence, our pre-processing step operates similarly to line-search optimization.

Furthermore, to keep the run-time of the pre-processing step from spiralling out of control on larger terrains, we estimate the optimal scaling factor for each entity with respect to a local neighbourhood of entities. Because entities that are far apart are likelier to be unable to see other, this additionally helps avoid biasing the relocation too heavily towards false negatives.

We consider a set of entities e_i that exists solely for the purposes of the pre-processing step; one for each vertex of the simplified terrain, initially placed at each vertex's corresponding position on the original terrain (say, e_i^0). For each entity e_i, we estimate its optimal position along the line passing through e_i in the direction of its initial relocation vector r_i, with LoS accuracy computed relative to a local neighbourhood of entities e_j.

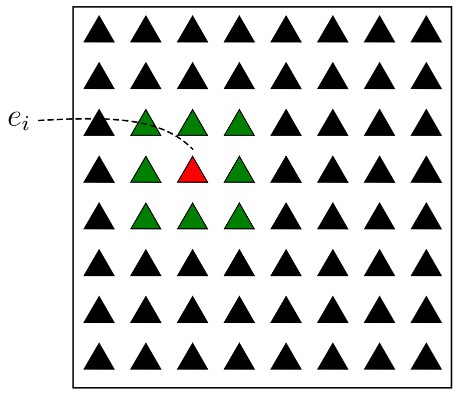
Given optimized positions for each entity e_i, we can determine optimized relocation vectors for the e_i. These relocation vectors have the form $s_i r_i$, where s_i is a scaling factor. Finally, we distribute scaling factors to all the relocation vectors in the vector field and scale them.

That is, at each iteration (see Figure 4.13), the following happens for each e_i

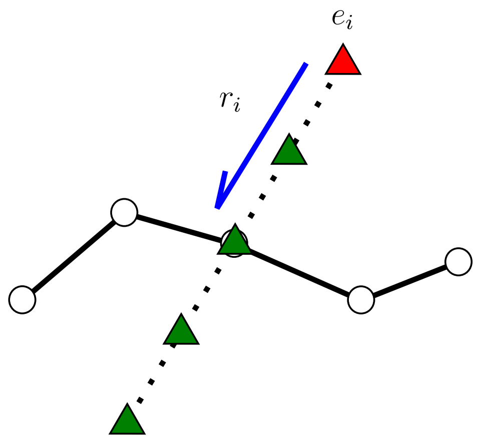
1. Consider a local neighbourhood of entities e_j around e_i.
2. Candidate positions for e_i are discretized along a line passing through e_i in the direction of r_i. (Note that if r_i is zero, all candidate positions will be located at the same position as e_i.)
3. The candidate position (say c) for e_i that produces the highest LoS accuracy with respect to the e_j is identified.
4. The scaling factor $s_i = \frac{(c-e_i^0)\cdot r_i}{r_i \cdot r_i}$, such that $e_i^0 + s_i r_i = c$, is calculated.
5. The position of e_i is updated to c.

The iterations continue until no further improvement can be found, or a maximum number of iterations is reached. Once the s_i have all been calculated, each relocation vector r_i is scaled by s_i (i.e. $r_i' = s_i \cdot r_i$).

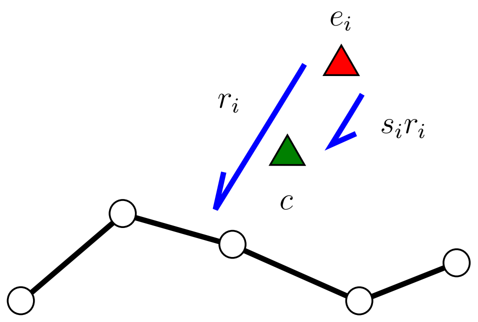
To scale the relocation vectors that do not correspond to one of the r_i, consider a face of the simplified terrain. Each of the face's four vertices has an associated entitiy e_j with a corresponding scaling factor s_j. The scaling factors for relocation vectors over the face (i.e. those vectors in the vector field which lie between the r_j) are bilinearly interpolated from these s_j.



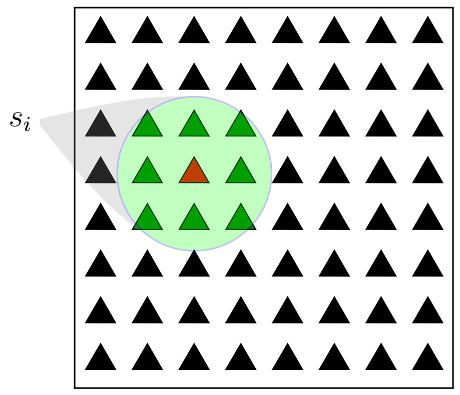
(a) Entity e_i is considered, along with a local neighbourhood of entities e_j.



(b) Candidate positions for e_i are determined along a line in the direction of e_i's relocation vector, r_i.



(c) The candidate position producing the highest accuracy, c, is identified. The scaling factor s_i is found.



(d) The scaling factor s_i is distributed to the relocation vectors in a local area about e_i.

Figure 4.13: Illustration of the pseudo-optimization pre-processing step.

This pre-processing step can be adapted for use with irregular terrains. From a relocation scheme suitable for irregular terrains (such as projection), a vector field over that terrain can be generated and the pre-processing applied. However, as our focus in this work lies with regular terrains, we have not explored this avenue of research.

Note that we have only tested this pre-processing step in conjunction with residual vector relocation.

Chapter 5

LINE-OF-SIGHT QUERY ALGORITHMS

While point relocation can be used to improve accuracy after simplification, there are inaccuracies introduced by terrain simplification for which point relocation cannot compensate. Simplification schemes reduce local regions of the terrain down to single faces. Hence, any entities originally in the region will be able to see each other over the simplification, regardless of their original LoS relationships.

Furthermore, although terrain simplification can be used to speed up individual LoS queries using a given algorithm, additional speed gains require further simplification. This results in lower accuracy for all LoS queries on the simplified terrain, which is undesirable.

In this chapter, we describe a hierarchical algorithm that combines existing techniques and uses simplified terrains for query acceleration only when acceleration is needed, thus improving the accuracy over a straight application of terrain simplification. Our algorithm is a combination of the Bresenham line algorithm adapted to LoS queries and the min/max quad tree algorithm. We take advantage of the strengths of each to reduce the impact of either of their weaknesses.

5.1 Bresenham's Line Algorithm

The *Bresenham line algorithm* is a well-known algorithm originally designed to plot a line segment on a raster image given its two endpoints (refer to Figure 2.1). It can be efficiently implemented using only integer operations [Bresenham, 1965]. Treating a regular terrain as a raster grid, the sight line between two entities can be traversed using the Bresenham algorithm and the height of the sight line compared against the terrain elevation values (see Figure 2.2).

Note that the original formulation of the Bresenham algorithm does not traverse *all* elevation points along the sight line's path, skipping elevation points that share a small intersection with the sight line, and so is slightly inaccurate. While it is possible to modify this behaviour and increase Bresenham's accuracy, our implementation uses the inaccurate version, since it is fast and accurate enough for our purposes.

Our implementation of Bresenham's line algorithm is based off of efficient pseudocode from Wikipedia's page on Bresenham's algorithm [Wikipedia, 2013a]. We reproduce a modified version of the pseudocode here for posterity (see Algorithm 5.1).

Algorithm 5.1 BresenhamVisibility(T, p_1, p_2) :

```
Determine grid indices $(i_1, j_1)$ for point $p_1$ on $T$
Determine grid indices $(i_2, j_2)$ for point $p_2$ on $T$
$di := abs(i_2 - i_1)$
$dj := abs(j_2 - j_1)$
$si := sgn(i_2 - i_1)$
$sj := sgn(j_2 - j_1)$
$err := di - dj$
while $i_1 \neq i_2$ AND $j_1 \neq j_2$ do
  $z :=$ Estimate of the sight line's elevation at cell $(i_1, j_1)$ of $T$
  if $z \leq T(i_1, j_1)$ then
    return 0 (Not Visible)
  end if
  $e2 := 2 * err$
  if $e2 > -dj$ then
    $err := err - dj$
    $i_1 := i_1 + si$
  end if
  if $e2 < di$ then
    $err := err + di$
    $j_1 := j_1 + sj$
  end if
end while
return 1 (Visible)
```

To estimate the height of the sight line at a given cell, our implementation computes the point on the sight line that is closest (with respect to the $x - y$ plane) to the center of the cell, and takes the z coordinate of that point to be the elevation of the sight line. That is, given sight line endpoints p_1 and p_2 in $\mathbb{R}^3$ with grid indices $q_1 = (i_1, j_1)$ and $q_2 = (i_2, j_2)$ respectively, then the elevation z of the sight line at the cell with index $q_{cell} = (i_{cell}, j_{cell})$ is $z = w * p_1.z + (1 - w) * p_2.z$ where $w = \frac{(q_2 - q_1)\cdot(q_2 - q_{cell})}{(q_2 - q_1)\cdot(q_2 - q_1)}$.

The Bresenham algorithm runs in $O(d)$ time, where d is the L_1 or Manhattan distance between the sight line's endpoints. Hence, the Bresenham algorithm runs fastest when the endpoints are close together on an elevation grid or, due to early algorithm termination when the sight line is occluded, when the terrain has many sharp features. Additionally, since the algorithm has no overhead, its memory footprint is equivalent to the memory footprint of the terrain it's executed on.

5.2 Min/Max Quad Tree Algorithm

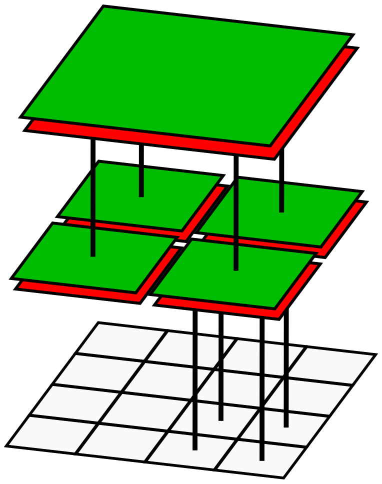
The *min/max quad tree algorithm*, by comparison, avoids visiting all elevation values along the sight line's path by visiting regions of the terrain first before recursing down into the finer data. A min/max quad tree is used to gather minimum and maximum elevation data for regions of the terrain (see Figure 5.1(a)).

Each leaf node of the quad tree corresponds to an elevation value in the original terrain. Each interior node records the minimum and maximum elevation values of its children nodes. Using this information, regions of the terrain that definitely lie below or above the sight line can be quickly culled from the computation [Duvenhage, 2009].

The algorithm is inherently recursive. Starting with the root node, the following occurs:

1. If the height of the sight line's endpoints is greater than the maximum elevation for a node, then the sight line lies completely above the region covered by the node, and so the algorithm may safely return a value of true for that node without traversing the children. See Figure 2.3(a).

2. If one or both of the sight line's endpoints lie below the node's minimum elevation, then the sight line is obstructed by the terrain in the region covered


(a) Quad tree structure.

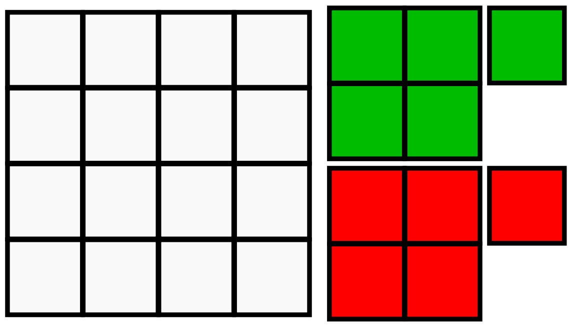

(b) Quad tree laid out as mip-maps. Notice that the quad tree and terrain take up less than twice the space of the terrain.

Figure 5.1: Illustration of the min/max quad tree structure. The terrain is shown in gray. Each node of the quad tree has a minimum elevation component (shown in red) and a maximum elevation component (shown in green).

by the node, and the algorithm may return a value of false for the entire computation. See Figure 2.3(b).

3. If neither of the above cases holds, then it is possible that the sight line is obstructed within one of the child nodes. Hence, each must be visited recursively. If the sight line passes through a given child node, determined using an intersection test between the line and the node, then the algorithm is recursively called on the child node with the sight line segment passing through it. At the leaf node level, the algorithm behaves similarly to the Bresenham line algorithm.

The min/max quad tree algorithm can be implemented efficiently using the technique described in [Langetepe and Zachmann, 2006] for intersecting a ray with the cells of a quad tree (see Figure 5.2). Consider a sight line between two points p_1 and p_2 extended to an infinite ray $r(t) = (1-t) * p_1 + t * p_2$, for $t \in \mathbb{R}$. Now consider the root node of a quad tree covering a rectangular domain $D = \{(x, y) : x_{left} \leq x \leq x_{right}, y_{bottom} \leq y \leq y_{top}\}$

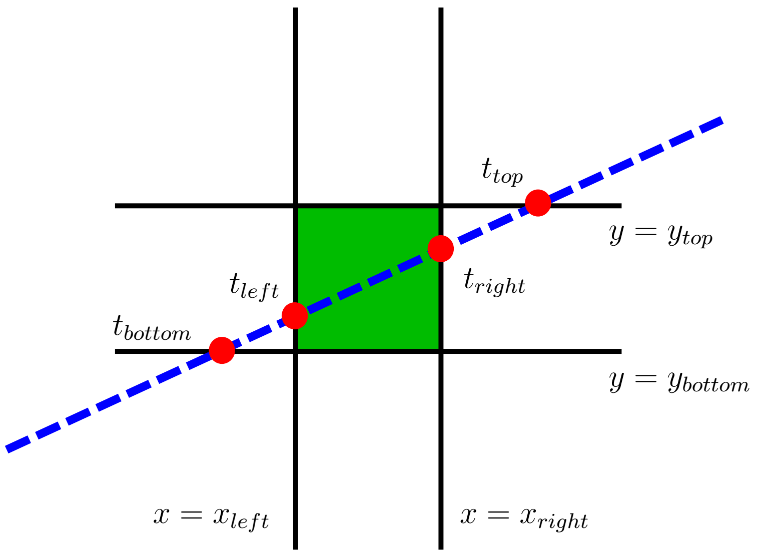


Figure 5.2: A quad tree node (shown in green) with bounding lines $x = x_{left}, x = x_{right}, y = y_{bottom}, y = y_{top}$ intersected with a sight line (shown in blue) using the technique from [Langetepe and Zachmann, 2006]. The sight line's intersections with the bounding lines are shown as red dots, with associated intersection parameter values $t_{left}, t_{right}, t_{bottom}, t_{top}$.

for some $x_{left}, x_{right}, y_{bottom}, y_{top} \in \mathbb{R}$. We can intersect $r(t)$ with the vertical and horizontal lines $x = x_{left}, x = x_{right}, y = y_{bottom}, y = y_{top}$ to obtain intersection parameter values $t_{left}, t_{right}, t_{bottom}, t_{top}$. The intersection parameters for child nodes of the quad tree can be easily found using these four values.

For each quad tree node, given its $t_{left}, t_{right}, t_{bottom}, t_{top}$ values, the intersection parameter values for the vertical and horizontal lines bisecting the node are $t_v = \frac{1}{2} \cdot t_{left} + \frac{1}{2} \cdot t_{right}$ and $t_h = \frac{1}{2} \cdot t_{bottom} + \frac{1}{2} \cdot t_{top}$. Then the given node's top left child has intersection parameters $t_{left}, t_v, t_h, t_{top}$; the top right child has values $t_v, t_{right}, t_h, t_{top}$; and so on.

This technique allows us to quickly and efficiently intersect the sight line with each child node and, hence, quickly and efficiently determine the height of the sight line at each node.

Furthermore, while the min/max quad tree algorithm is inherently recursive, it can be implemented efficiently using stacks. The pseudocode for our implementation can be found in Algorithm 5.2.

This algorithm runs in $O(\log(d))$ time on average [Duvenhage, 2009], however, the worst

Algorithm 5.2 QuadTreeVisibility(T, p_1, p_2) :

```
$stack$ := An empty stack
$r(t) := (1-t) * p_1 + t * p_2$, for $t \in \mathbb{R}$
$N$ := The min/max quad tree node at $T$
$t_{left}$ := Parameter for intersection between $r(t)$ and $x = N.x_{left}$
$t_{right}$ := Parameter for intersection between $r(t)$ and $x = N.x_{right}$
$t_{bottom}$ := Parameter for intersection between $r(t)$ and $y = N.y_{bottom}$
$t_{top}$ := Parameter for intersection between $r(t)$ and $y = N.y_{top}$
Push $N, t_{left}, t_{right}, t_{bottom}, t_{top}$ onto $stack$
while $stack$ is not empty do
  Pop $N, t_{left}, t_{right}, t_{bottom}, t_{top}$ off $stack$
  $t_{lower} := max(min(t_{left}, t_{right}), min(t_{bottom}, t_{top}))$
  $t_{higher} := min(max(t_{left}, t_{right}), max(t_{bottom}, t_{top}))$
  {Note: if $t_{lower} \geq t_{higher}$, then the node is not intersected}
  if $t_{lower} < t_{higher}$ then
    $q_1 := r(t_{lower})$
    $q_2 := r(t_{higher})$
    if $q_1.z \leq N.min$ OR $q_2.z \leq N.min$ then
      return 0 (Not Visible)
    end if
    if $q_1.z \leq N.max$ OR $q_2.z \leq N.max$ then
      $t_v := \frac{1}{2} \cdot t_{left} + \frac{1}{2} \cdot t_{right}$
      $t_h := \frac{1}{2} \cdot t_{bottom} + \frac{1}{2} \cdot t_{top}$
      Push $N$'s top left child, $t_{left}, t_v, t_h, t_{top}$ onto $stack$
      Push $N$'s top right child, $t_v, t_{right}, t_h, t_{top}$ onto $stack$
      Push $N$'s bottom right child, $t_v, t_{right}, t_{bottom}, t_h$ onto $stack$
      Push $N$'s bottom left child, $t_{left}, t_v, t_{bottom}, t_h$ onto $stack$
    end if
  end if
end while
return 1 (Visible)
```

case run time is $O(d\log(d))$. Due to overhead (including intersection computations), when d is small the quad tree algorithm tends to be slower than the Bresenham algorithm. Hence, this algorithm is preferred for use when the endpoints are far apart on an elevation grid or, due to region culling when the sight line lies above the maximum elevation, when the terrain is fairly flat. This algorithm features a larger memory footprint than that of the original terrain (less than double), as the maximum and minimum elevation mipmaps for each level of the quad tree must be stored (see Figure 5.1(b)).

5.3 Hierarchical LoS Algorithm

While both algorithms are fast under the right conditions, Bresenham's algorithm is not optimal when the terrain is flat and entities are far apart, whereas the min/max quad tree algorithm is not optimal when the terrain has many obstructing features and entities are close together. While terrain simplification can be used to speed each up, accuracy suffers as a result. Working with a static simplified terrain results in accuracy loss even for LoS queries that don't require speed-ups in the first place.

However, dynamically selecting a degree of simplification to match speed requirements for each query would address this problem. We present here an algorithm that makes use of this idea. The algorithm is a combination of Bresenham's line algorithm, the min/max quad tree algorithm, and regularity-preserving terrain simplification intended to use each of their respective strengths to develop a fast algorithm that is more accurate than either algorithm applied to a static simplified terrain (see Figure 5.3).

5.3.1 Algorithm Details

We wish to determine an algorithm A that is both fast and accurate when working on large terrain models T with large point sets $P \subset \mathbb{R}^3$.

The Bresenham algorithm works best when points are close together, whereas the min/max

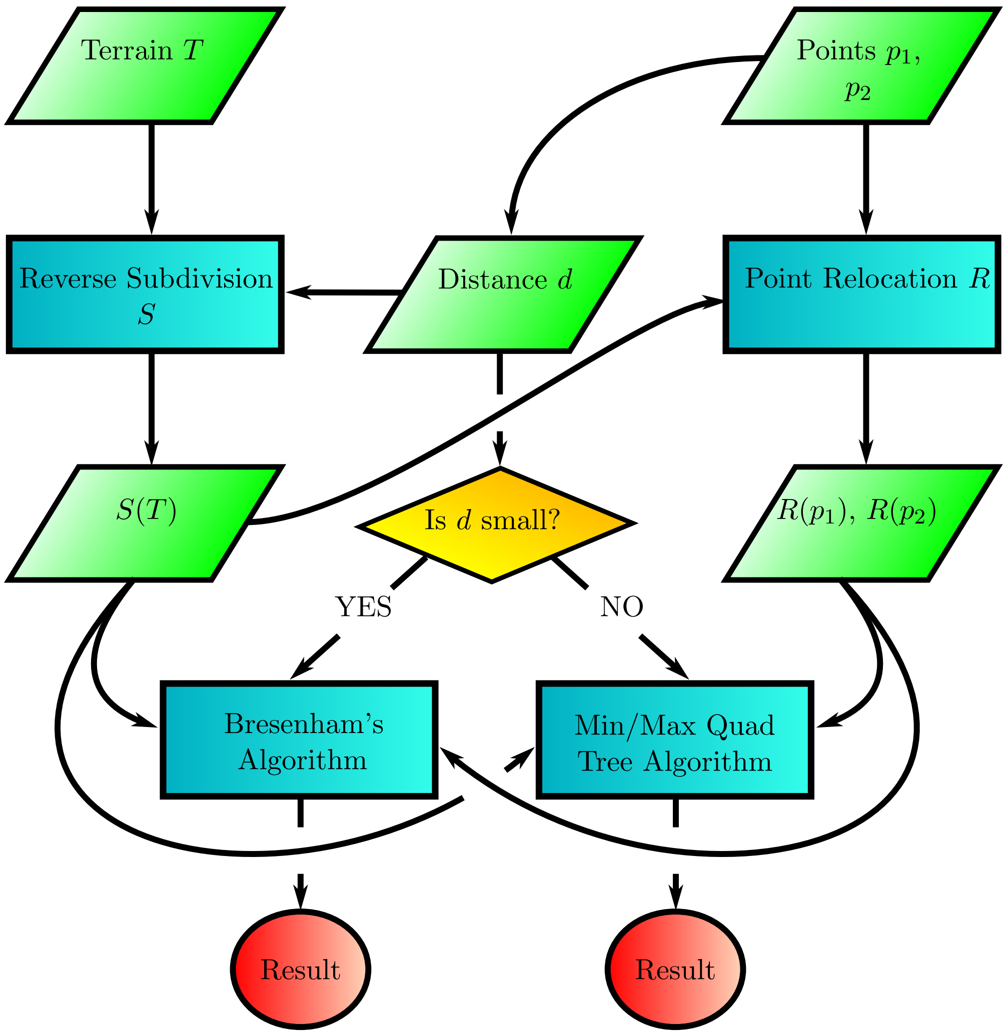


Figure 5.3: Simplified flowchart of our hierarchical algorithm for an LoS query.

quad tree algorithm works comparitively better when they are far apart. We expect that by combining the two we can create an algorithm that works well in either case.

Each simplification of the terrain reduces the accuracy of the LoS queries. For points that are close enough for the Bresenham algorithm to work fast we do not want to lose accuracy by using a simplified terrain. Hence, for points with a sufficiently small L_1 distance d between them, we use the Bresenham algorithm on the original terrain. When d is sufficiently large, we use a min/max quad tree. If d is not sufficiently small to use Bresenham's algorithm nor sufficiently large to use the quad tree, terrain simplification can be applied to repeatedly reduce d, bringing some far points close enough to be used with the Bresenham line algorithm.

Using terrain simplification in this way, we generate a hierarchy of simplified terrains (say $T_1, T_2, \ldots, T_{top}$ where $T_1 = T$) with a min/max quad tree on one of the levels (say T_{tree}). Each level of the hierarchy below the quad tree handles LoS queries using the Bresenham algorithm. The min/max quad tree handles LoS queries for points that are sufficiently far apart. Two distance thresholds, t_B and t_Q, respectively determine when points are sufficiently close together and sufficiently far apart.

Note that we build the quad tree based on a simplified terrain rather than the original. This is done both to reduce memory usage by the quad tree and to ensure consistent accuracy between the quad tree algorithm and Bresenham's algorithm running on the simplified terrain. The level index $tree$ may be computed as $\lceil \log_2(\frac{t_Q}{t_B}) \rceil + 1$.

The choice of terrain simplification method is an important one, as it has ramifications for algorithm accuracy and memory usage. Firstly, the method must be regularity-preserving. Secondly, it is ideal that a correspondence exist between the elevation values of the simplified and original terrains. Our work uses a 1 to 4 correspondence (i.e. every elevation value of a simplified terrain represents four elevation values from the terrain immediately below it in the hierarchy, similar to the structure of a quad tree), which means that with each application of simplification d is cut in half.

Subsampling by a factor of 2 is an obvious choice, and has the advantage that the simplified terrains need not be stored in memory. One could simply increase the step size of the Bresenham algorithm. Another possibility is to subsample by taking the maximum or minimum elevation values, which would respectively force all errors to be false negatives or false positives. The resulting hierarchy would have the same structure as a max or min quad tree, respectively, and could support quad tree queries.

For our tests, we have used local least-squares reverse Faber subdivision (LLSRFS) to generate the hierarchy of simplified terrains. This type of simplification can be applied in linear time (based on the size of the terrain model), and our results show that it preserves

LoS query accuracy better than subsampling. In our preliminary tests, both the identity and projection relocation functions produced low rates of true positives, hence we use the half projection hybrid approach — which produces high rates of true positives — to relocate entities onto the layers of the hierarchy.

Pseudocode for the algorithm is given in Algorithm 5.3.

Algorithm 5.3 Hierarchical(T, p_1, p_2) :

```
Determine grid indices $(i_1, j_1)$ for point $p_1$ on $T_1$
Determine grid indices $(i_2, j_2)$ for point $p_2$ on $T_1$
$k := 1$
$dist := sqrt((i_2 - i_1)^2 + (j_2 - j_1)^2)$
if $dist > t_Q$ then
  return QuadTreeVisibility($T_{tree}, p_1, p_2$)
else
  while $dist > t_B$ AND $k < top$ do
    $k := k + 1$
    $dist := dist/2$
  end while
  return BresenhamVisibility($T_k, p_1, p_2$)
end if.
```

To reduce the number of expensive square root calls in our pseudocode, a more efficient version of the algorithm compares squared distances (see Algorithm 5.4).

Algorithm 5.4 EfficientHierarchical(T, p_1, p_2) :

```
Determine grid indices $(i_1, j_1)$ for point $p_1$ on $T_1$
Determine grid indices $(i_2, j_2)$ for point $p_2$ on $T_1$
$k := 1$
$distSq := (i_2 - i_1)^2 + (j_2 - j_1)^2$
if $distSq > {t_Q}^2$ then
  return QuadTreeVisibility($T_{tree}, p_1, p_2$)
else
  while $distSq > {t_B}^2$ AND $k < top$ do
    $k := k + 1$
    $distSq := distSq/2^2$
  end while
  return BresenhamVisibility($T_k, p_1, p_2$)
end if.
```

Note also that it is possible to use the algorithm without the quad tree by setting $t_Q = \infty$.

5.3.2 Run Time Analysis

The average run time of our algorithm is given piecewise as

$$\begin{cases} O(t_B) & \text{when } d \leq t_Q \\ O(\log(d)) & \text{otherwise.} \end{cases}$$

The wost case run time is

$$\begin{cases} O(t_B) & \text{when } d \leq t_Q \\ O(d\log(d)) & \text{otherwise.} \end{cases}$$

Note that the asymptotic run time of the algorithm with $t_Q = \infty$ is simply $O(t_B)$. Notably, this run time is independent of both the terrain model size and entity positions. If this version of the algorithm is used with a constant t_B, the algorithm run time will be effectively constant.

We show the derivation for the run time of the algorithm without the quad tree (i.e. $t_Q = \infty$); the run time for the version with the quad tree follows from this and the run time of the min/max quad tree algorithm.

Let d_i denote the Manhattan distance between the sight line's endpoints over T_i. Notice that $d_1 = d$ and $d_i = (\frac{1}{2})^{i-1}d$ for all $i = 1, 2, \ldots, top$.

The bottleneck of the algorithm is the call to Bresenham's algorithm. Whenever $d_i \leq t_B$, we use the $O(d_i)$ time Bresenham line algorithm on T_i. Hence, the run time of our algorithm can be given piecewise as

$$\begin{cases} O(d_1) & \text{when } d_1 \leq t_B \\ O(d_2) & \text{when } \frac{1}{2}t_B < d_2 \leq t_B \\ \cdots & \\ O(d_{top}) & \text{when } \frac{1}{2}t_B < d_{top} \leq t_B. \end{cases} \tag{5.1}$$

Notice that $d_i \leq t_B$ in each case. Hence, we can replace the $O(d_i)$ by $O(t_B)$ in Equation 5.1 to yield

$$\begin{cases} O(t_B) & \text{when } d_1 \leq t_B \\ O(t_B) & \text{when } \frac{1}{2}t_B < d_2 \leq t_B \\ \cdots & \\ O(t_B) & \text{when } \frac{1}{2}t_B < d_{top} \leq t_B. \end{cases} \tag{5.2}$$

Equation 5.2 collapses down to

$$O(t_B), \tag{5.3}$$

which is what we wished to show.

It should be noted that, since the Bresenham line algorithm reaches its worst-case run time when the sight line is unobstructed, the performance of our algorithm will deteriorate quickly on terrains with large, flat regions. Indeed, in such cases it would be wise to use the min/max quad tree algorithm alone. Aside from these types of terrain, however, our algorithm is specially designed to perform well.

Chapter 6

RESULTS AND DISCUSSION

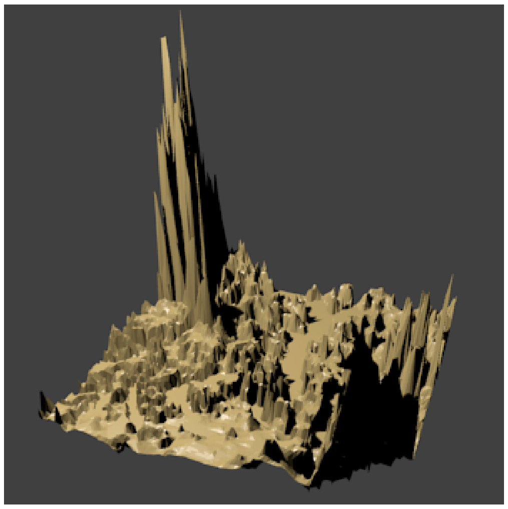
In this chapter, we present our LoS query accuracy comparison results and discussions of such for the various methods and algorithms described throughout this thesis. Results are divided between two sections: one for the simplification methods from Chapter 3 and the relocation methods from Chapter 4, and a second results section for the algorithms described in Chapter 5. Results tables and charts may be found at the end of the chapter.

In our tests, we have attempted to reflect the conditions of a real simulation by taking a small number of entities in comparison to the number of terrain vertices, partly too for speed concerns. As LoS accuracy is dependent on the underlying terrain, the number of entities, and their distribution, it is difficult to extrapolate with any certainty what the query accuracy for a given entity count and distribution would be given the query accuracy for another entity count and distribution (even one covering the entire terrain). Hence, our results serve better as a comparison between methods than as an indicator of expected LoS accuracy.

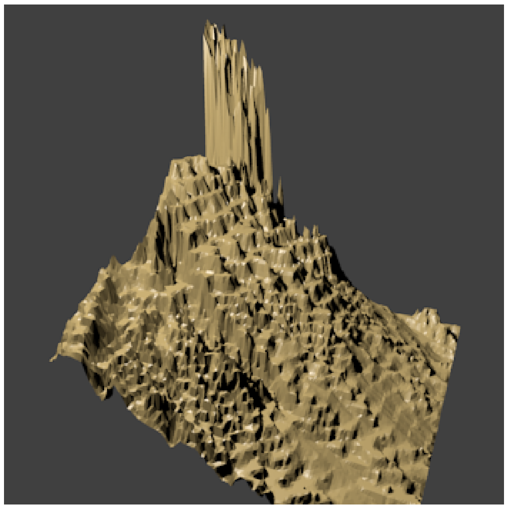
6.1 Simplification and Relocation Results

In this section we describe and discuss our results from comparing the LoS query accuracy of the various simplification methods and relocation schemes described throughout the paper.

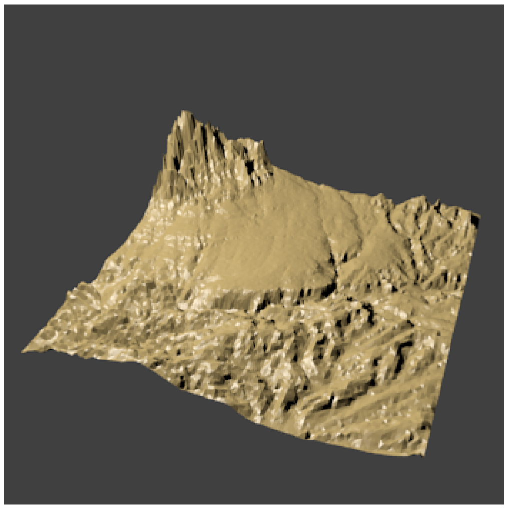
LoS queries were conducted using a ray casting approach on five different terrain models (120×120 height maps, shown in Figure 6.1) with varying levels of sharp features, each tested using six random distributions of fifty test points (1,225 sight lines). The terrain models were each tested at three levels of simplification: 25% of the original terrain size, 6% of the original terrain size, and 1.5% of the terrain size (which correspond to the terrain size



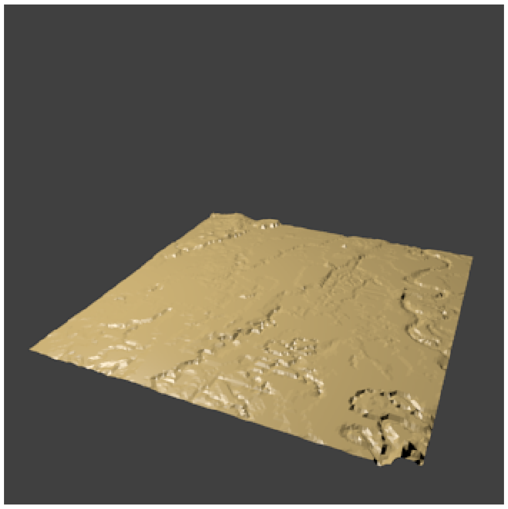
(a) The “Seattle” terrain



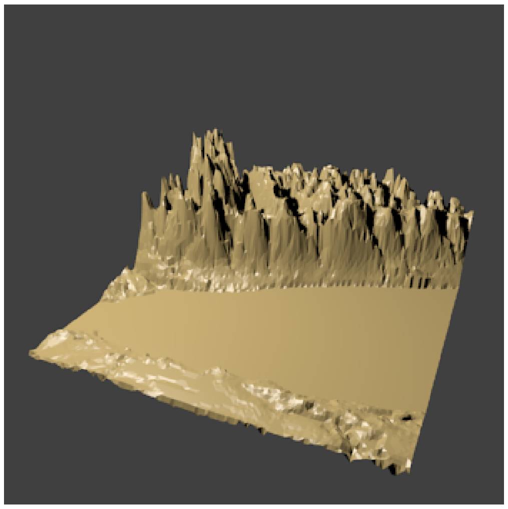
(b) The “Denver” terrain



(c) The “Eagle Pass” terrain



(d) The “Jackson” terrain



(e) The “Buffalo” terrain

Figure 6.1: The 120×120 height map terrains used in our tests.

after 1, 2, and 3 iterative applications of reverse subdivision, respectively).

Tests were conducted on a *global* scale, wherein no limits were placed on effective sight line distance. In several cases, additional tests at a *local* scale, wherein LoS queries had limits placed upon effective sight line distance, were also conducted. The distance limit used in our tests was 30 units for every entity (where 1 unit represents the horizontal spacing between elevation values); entities exceeding this distance from each other were automatically deemed to be invisible to each other. Hence, entities in our local tests can see up to one quarter of each 120×120 height map terrain.

6.1.1 Comparison of Simplifications

See Tables 6.1 through 6.3 for the LoS query accuracy results after simplification by each of the methods described in Chapter 3. Each simplification method was paired with the identity, projection, and half projection relocation methods. Rates of true positives and true negatives are shown in Table 6.4. Chart summaries of these results may be found in Figures 6.3 through 6.4.

Our results indicate that, while the reverse subdivision methods have a more restrictive nature than the irregular simplification methods, the restriction to regular surfaces does not significantly diminish the query accuracy. Indeed, these reverse subdivision methods appear to preserve LoS accuracy roughly on par with or, in some cases, superior to the irregular methods.

Of the regularity-preserving methods tested, the subsampling scheme performed unexpectedly well. The other three variants (GLSRFS, LLSRFS, and feature aware) have approximately equal rates of total accuracy, however, due to its superior run time the LLSRFS scheme emerges as the preferred reverse subdivision scheme. Unfortunately, our novel algorithm does not appear to have shown significant improvement in average accuracy over the other subdivision methods.

Interestingly, the identity relocation function also performed unexpectedly well, particularly when paired with LLSRFS. Overall, however, projection appears to preserve LoS query accuracy better than identity. Additionally, it has superior rates of true positives, albeit with slightly lower rates of true negatives. Both relocations, however, tend to favour false negatives over false positives. Ultimately, it seems that the false-positive-favouring behaviour of half projection tends to result in poorer LoS query accuracy results when compared against identity and projection. The reason for this, and likely the reason for identity's strong results, appears to be that the majority of LoS queries (especially without limitations on effective sight line distance) tend to return not visible.

6.1.2 Comparison of Feature Weights

To identify the best feature weight w to use when applying feature aware reverse subdivision, we conducted several tests to examine the impact of different weights on LoS query accuracy. See Figure 6.5 for LoS query accuracy results plotted against various feature weight values.

Our results indicate that LoS query accuracy over terrains reverse subdivided with feature aware reverse subdivision peaks in the vicinity of $w = 0.75$. Hence, for other results detailed in this chapter, those regarding feature aware reverse subdivision are computed using a feature weight of 0.75.

6.1.3 Comparison of Smoothing Weights

We tested the effect of the smoothing on LoS query accuracy using two smooth reverse subdivision schemes: smooth LLSRFS and smooth feature aware reverse subdivision with various smoothing weights w. See Figure 6.6 for a summary of our results.

Our results indicate that, at least when used with reverse Faber subdivision methods, the effects of smoothing are detrimental to LoS query accuracy results. As the smoothing effect increases with w, the LoS query accuracy decreases. This is consistent with our expectations regarding the effect of smoothing upon LoS query accuracy.

6.1.4 Comparison of Iterative Relocation Methods

For each ierative point relocation method, the local neighbourhood of each entity was discretized into a $5 \times 5 \times 5$ grid. A distance of 0.5 units was used for the constant distance methods. The decreasing distance method decreased the distance by a factor of 0.75 each iteration. The maximum number of iterations allowed was 10.

See Table 6.5 for the average LoS query accuracy and for the average number of iterations for each iterative relocation method, and Figure 6.7 for a chart summary. Due to speed concerns, the iterative relocation methods were only tested at simplification to 1.5% and at a global scale.

We have found that point relocation by iteration is very powerful, making it possible to achieve over 90% accuracy despite simplification to 1.5% of the original terrain size. Of the various implementations tested, the projection distance method performed the strongest, featuring the lowest average number of iterations, and generally higher accuracy averages. This would suggest that the distance of projection plays an important role in an optimal point relocation.

The results clearly show that there is room for improvement over the projection and identity methods. However, whether a practical point relocation method can achieve results similar to the iterative methods remains an open question.

6.1.5 Comparison of Residual Vector Relocation Methods

We tested the residuals relocation method (with and without modifications) against the identity and projection functions at a global and local scale.

Figure 4.10 indicates that, as the average length of residual vectors for a given terrain increases, the performance of residuals relocation on that terrain improves in comparison to the identity function. The relationship appears to be somewhat linear, with a maximum occuring when the average length of the residuals is around 1.5. Hence, we conducted our tests on scaled residual vector relocation taking $threshold = 1.5$.

Figure 6.8 compares the accuracy of the residuals relocation schemes, relative to identity.

Given a set of entities e_i, each corresponding to a vertex of the simplified terrain, our pre-processing step of the pseudo-optimized residuals scheme does the following. For each i, an entity e_j is considered to be in the local neighbourhood of e_i if at most three simplified terrain faces lie between the vertices corresponding to e_i and e_j. At each iteration, nineteen candidate positions are distributed randomly along the line segment from $e_i - 2(1 - s_i)r_i$ to $e_i + 2(1 - s_i)r_i$, where s_i is the current scaling factor and r_i is the residual vector for e_i. (Notice that $(1 - s_i)r_i$ is a vector in the direction of r_i that projects e_i onto the simplified terrain. Hence, this scheme behaves similiarly to projection distance relocation.) Our implementation uses up to a maximum of five iterations.

Note that, since the vertices of a subsampled terrain are equal to their corresponding vertices on the original terrain, the residual vectors at these vertices will equal the zero vector. Hence, by construction, our pseudo-optimized residuals scheme will behave equivalently to identity over a subsampled terrain.

See Tables 6.1 through 6.3 for the average LoS query accuracy results for the residual vector relocation methods and Figures 6.3 through 6.4 for chart summaries. Table 6.4 illustrates the rates of true positives/negatives under our residuals relocation methods. Our results indicate that our pseudo-optimized residuals relocation method improves the LoS query accuracy over the identity and projection functions on both a global and local scale, especially on LLSRFS terrains. The unscaled residuals relocation method offers slight improvement over the projection function, but on a global scale the identity performs stronger.

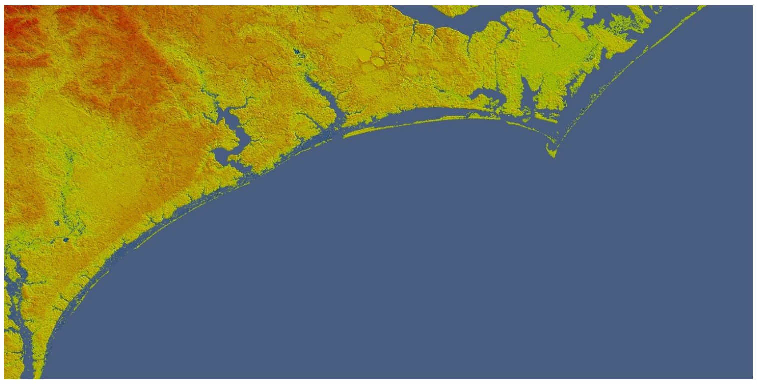
6.2 Hierarchical Algorithm Results

For our hierarchical algorithm tests, distance thresholds $t_B = 40$ and $t_Q = 320$ were used. Preliminary testing indicated that the performance of Bresenham's algorithm deteriorates against the run time of the min/max quad tree algorithm when $d > 40$. The quad tree threshold was chosen to allow four hierarchy levels on which to run Bresenham's algorithm, as accuracy results in Section 6.1.1 indicate that four levels can be used without too much loss in accuracy. All results were obtained on a computer with an Intel Core i7 CPU, 12 GB of RAM, and a 64-bit architecture running Windows 7.

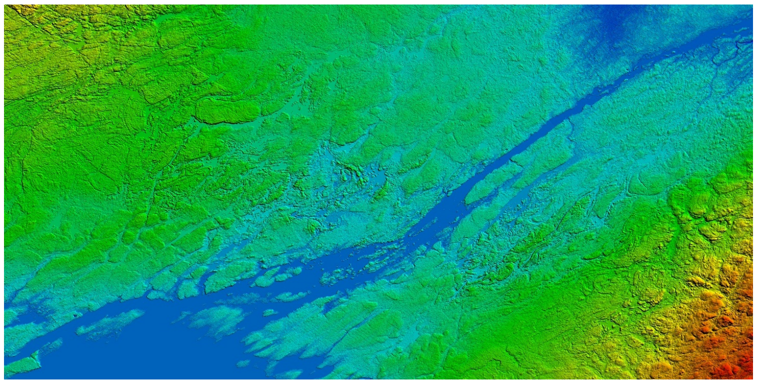
Our tests were run on three different large height map terrain models ranging from very flat to very mountainous. These are shown in Figure 6.2. Tests were run with random distributions of 300 test points (44,850 sight lines) on the Camp LeJeune and Kingston terrains, and 500 test points (124,750 sight lines) on the Rocky Mountains terrain. Tests were conducted on a global and local scale, with a maximum Manhattan distance of 320 allowed for sight lines in the local case.

Refer to Table 6.6 for the average query accuracy, rates of true positives and negatives, and average run time of each visibility algorithm. A graphical summary of the results can be found in Figure 6.9. Because of the thoroughness of the min/max quad tree approach, we consider the quad tree algorithm to be 100% correct, and compare the results of the other two algorithms against it.

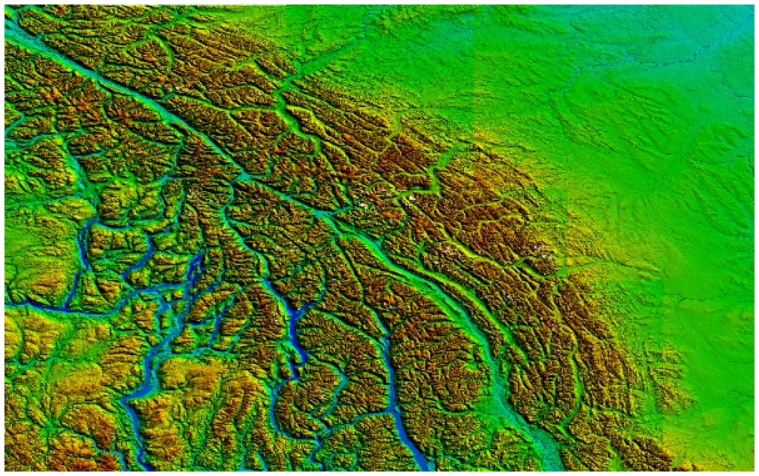
Our results indicate that our hierarchical approach can perform faster than Bresenham's algorithm and the min/max quad tree algorithm on average, with a fairly minimal impact on the accuracy. For our particular tests, visibility accuracy for the hierarchical approach never dropped below 95%. Since LoS queries between close points are performed on terrains that produce higher accuracy, our algorithm can clearly produce superior accuracy results than if all LoS queries were performed on a single terrain with a constant drop in accuracy (i.e. if reverse subdivision alone had been applied).



(a) The 7201x3601 "Camp LeJeune" terrain. Featuring both flat and rough regions, the Camp LeJeune terrain is suited primarily to the quad tree algorithm and secondarily to the Bresenham algorithm.

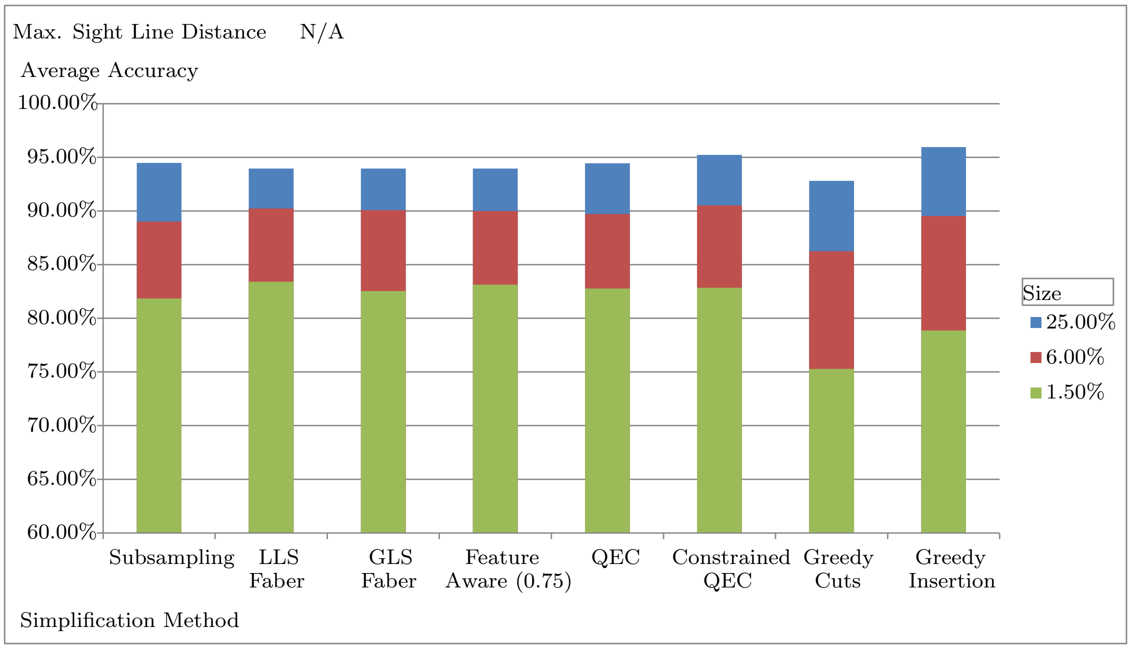


(b) The 7201x3601 "Kingston" terrain. As a terrain with many features, the Kingston terrain is well-suited to LoS queries using the Bresenham line algorithm.

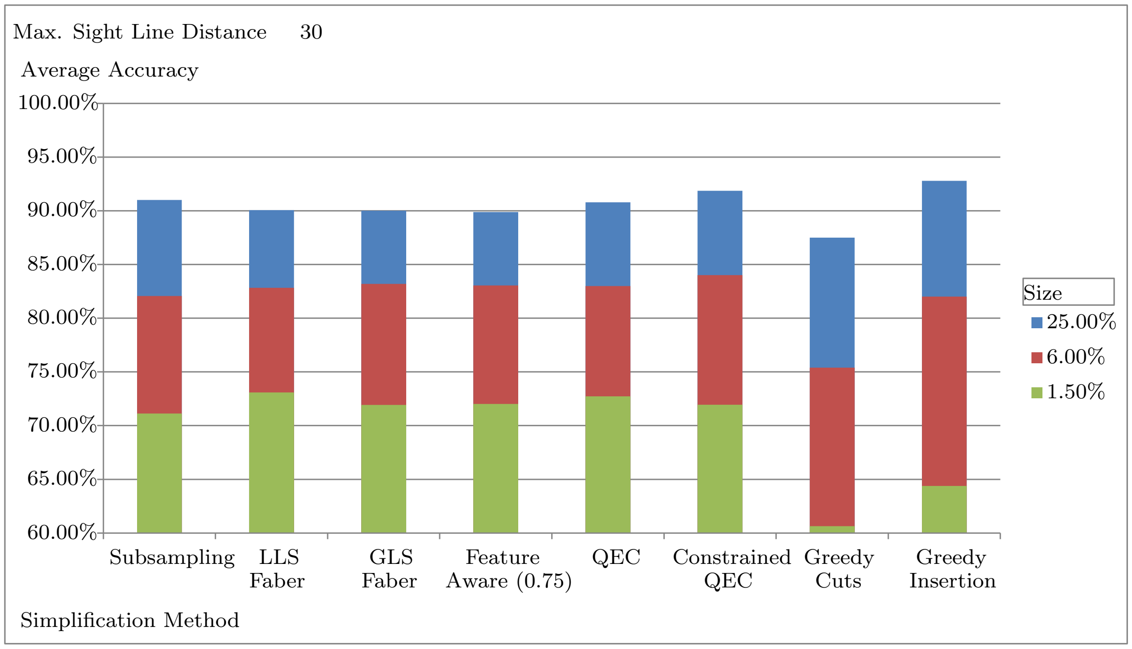


(c) The 9608x6005 "Rocky Mountains" terrain. As a very mountainous and feature-heavy terrain, despite its large size the Bresenham algorithm runs very fast on this terrain.

Figure 6.2: The three test terrain maps for the hierarchical algorithm.

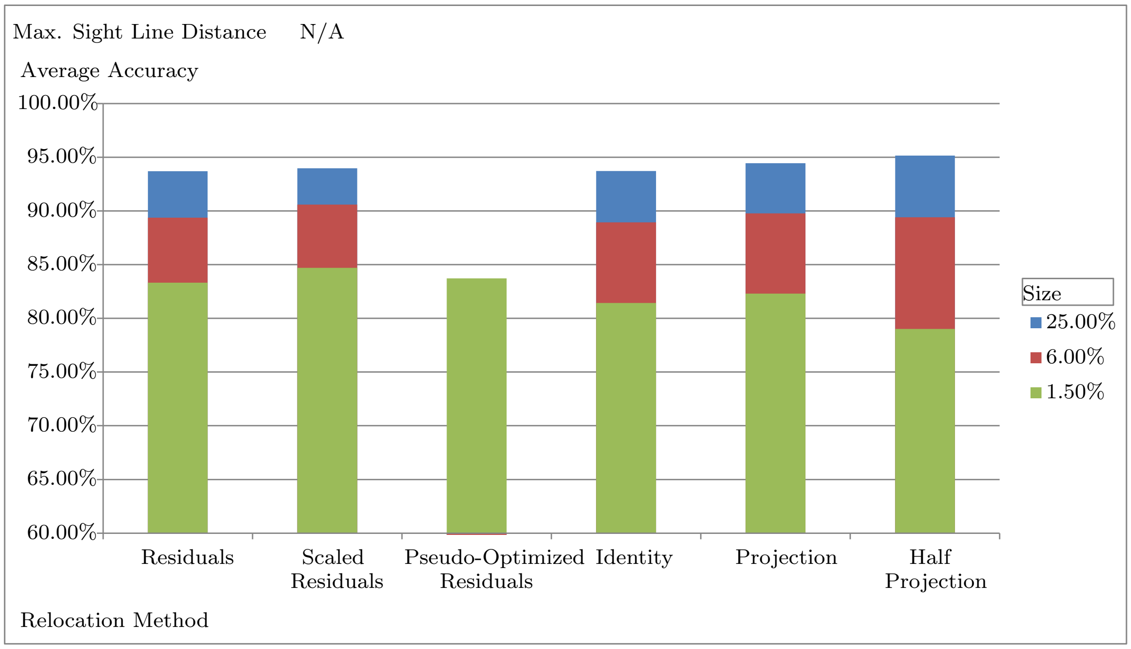


(a) Global query summary.

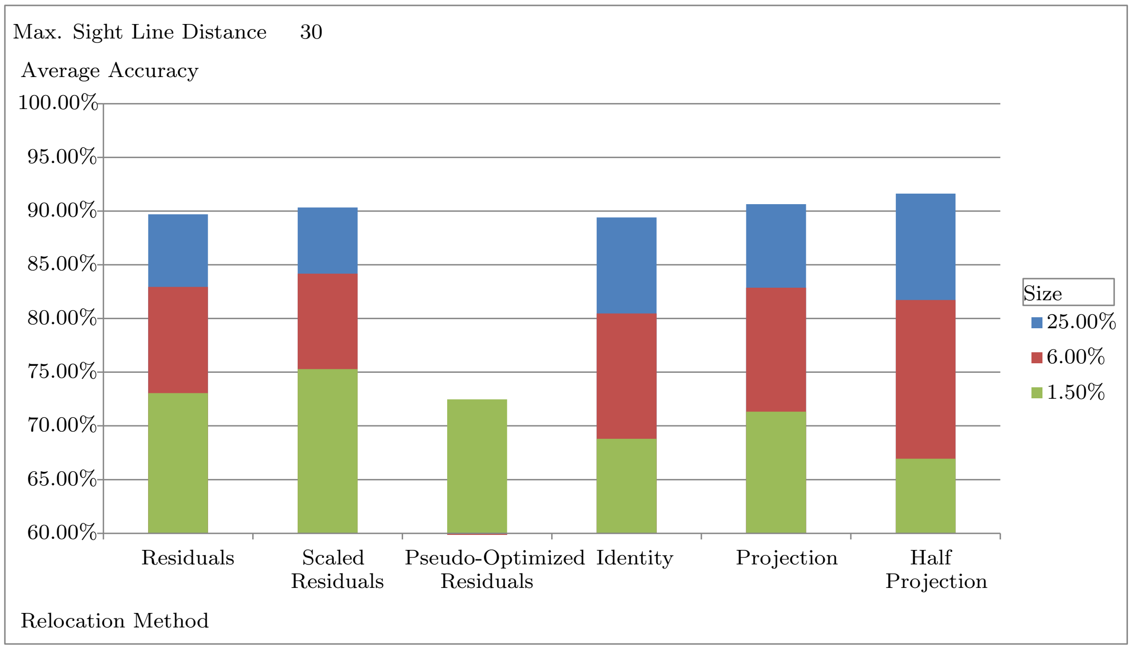


(b) Local query summary.

Figure 6.3: Summary of accuracy for the simplification methods (excepting smooth reverse subdivision) averaged over the identity, projection, half projection, residuals, and scaled residuals relocation schemes.



(a) Global query summary.



(b) Local query summary.

Figure 6.4: Summary of accuracy for the relocation methods averaged over the subsampling, LLSRFS, GLSRFS, feature aware, QEC, constrained QEC, greedy insertion, and greedy cuts simplification schemes.

(a) Global query results.

Max. Sight Line Distance	N/A

Average Accuracy	Identity	Projection	Half Projection	Residuals	Scaled Residuals
25.00%					
Subsampling	94.50%	94.37%	94.85%	93.99%	94.68%
LLSRFS	93.61%	93.95%	94.85%	93.71%	93.62%
GLSRFS	93.71%	94.00%	94.85%	93.59%	93.62%
Feature Aware (0.75)	93.85%	93.81%	94.67%	93.46%	93.96%
QEC	93.64%	94.47%	95.23%		
Constrained QEC	94.58%	95.18%	95.92%		
Greedy Cuts	90.59%	93.71%	94.13%		
Greedy Insertion	95.16%	96.00%	96.68%		
Grand Total	93.71%	94.44%	95.15%	93.69%	93.97%

(b) Local query results.

Max. Sight Line Distance	30

Average Accuracy	Identity	Projection	Half Projection	Residuals	Scaled Residuals
25.00%					
Subsampling	90.74%	90.79%	91.39%	90.56%	91.54%
LLSRFS	89.56%	90.01%	91.32%	89.50%	89.88%
GLSRFS	89.42%	90.12%	91.44%	89.48%	89.56%
Feature Aware (0.75)	89.36%	89.56%	90.91%	89.23%	90.36%
QEC	89.24%	91.09%	92.01%		
Constrained QEC	90.75%	92.04%	92.77%		
Greedy Cuts	84.62%	88.67%	89.18%		
Greedy Insertion	91.60%	92.78%	94.00%		
Grand Total	89.41%	90.63%	91.63%	89.69%	90.34%

Table 6.1: Average accuracy results after simplification to 25% of the original terrain size.

(a) Global query results.

Max. Sight Line Distance	N/A

Average Accuracy	Identity	Projection	Half Projection	Residuals	Scaled Residuals
6.00%					
Subsampling	88.48%	89.46%	88.41%	88.99%	89.73%
LLSRFS	90.84%	89.82%	90.01%	89.49%	90.99%
GLSRFS	90.36%	89.64%	89.93%	89.59%	90.91%
Feature Aware (0.75)	90.44%	89.60%	89.83%	89.38%	90.70%
QEC	88.88%	90.41%	89.86%		
Constrained QEC	89.74%	91.49%	90.30%		
Greedy Cuts	84.23%	87.93%	86.55%		
Greedy Insertion	88.45%	89.77%	90.37%		
Grand Total	88.93%	89.76%	89.41%	89.36%	90.58%

(b) Local query results.

Max. Sight Line Distance	30

Average Accuracy	Identity	Projection	Half Projection	Residuals	Scaled Residuals
6.00%					
Subsampling	80.20%	82.95%	80.92%	82.78%	83.49%
LLSRFS	82.26%	82.39%	82.25%	83.06%	84.21%
GLSRFS	81.99%	83.30%	82.87%	83.22%	84.54%
Feature Aware (0.75)	82.12%	83.05%	82.87%	82.71%	84.44%
QEC	81.36%	84.62%	82.98%		
Constrained QEC	82.27%	86.08%	83.71%		
Greedy Cuts	73.47%	77.68%	75.04%		
Greedy Insertion	80.00%	82.85%	83.16%		
Grand Total	80.46%	82.87%	81.72%	82.94%	84.17%

Table 6.2: Average accuracy results after simplification to 6% of the original terrain size.

(a) Global query results.

Max. Sight Line Distance	N/A

Average Accuracy	Identity	Projection	Half Projection	Residuals	Scaled Residuals	Pseudo-Optimized Residuals
1.50%						
Subsampling	80.53%	82.94%	78.51%	82.99%	84.24%	80.54%
LLSRFS	84.38%	82.88%	80.29%	83.67%	85.87%	86.23%
GLSRFS	82.62%	82.77%	79.93%	83.16%	84.23%	83.79%
Feature Aware (0.75)	82.91%	83.72%	81.15%	83.44%	84.44%	84.30%
QEC	81.93%	84.67%	81.72%			
Constrained QEC	82.40%	85.15%	81.01%			
Greedy Cuts	76.17%	77.31%	72.40%			
Greedy Insertion	80.50%	79.00%	77.10%			
Grand Total	81.43%	82.30%	79.01%	83.31%	84.70%	83.71%

(b) Local query results.

Max. Sight Line Distance	30

Average Accuracy	Identity	Projection	Half Projection	Residuals	Scaled Residuals	Pseudo-Optimized Residuals
1.50%						
Subsampling	68.16%	73.01%	67.13%	72.50%	74.81%	68.14%
LLSRFS	72.83%	72.98%	68.97%	73.85%	76.80%	75.43%
GLSRFS	70.07%	72.83%	68.90%	73.05%	74.79%	73.03%
Feature Aware (0.75)	70.36%	73.06%	69.10%	72.79%	74.75%	73.25%
QEC	71.41%	75.57%	71.20%			
Constrained QEC	70.31%	76.04%	69.51%			
Greedy Cuts	62.79%	61.80%	57.32%			
Greedy Insertion	64.43%	65.31%	63.43%			
Grand Total	68.79%	71.32%	66.94%	73.05%	75.29%	72.46%

Table 6.3: Average accuracy results after simplification to 1.5% of the original terrain size.

(a) Global query results.

Max. Sight Line Distance	N/A

	Identity	Projection	Half Projection	Residuals	Scaled Residuals	Pseudo-Optimized Residuals
1.50%						
Rate of True Positives	54.81%	69.34%	87.69%	50.53%	51.09%	53.59%
Rate of True Negatives	85.88%	83.59%	76.99%	88.36%	89.86%	88.79%
6.00%						
Rate of True Positives	59.90%	69.66%	87.87%	56.55%	59.03%	
Rate of True Negatives	93.39%	92.22%	89.11%	93.64%	94.64%	
25.00%						
Rate of True Positives	65.75%	75.52%	88.14%	69.94%	64.59%	
Rate of True Negatives	97.86%	97.03%	96.10%	96.73%	97.82%	

(b) Local query results.

Max. Sight Line Distance	30

	Identity	Projection	Half Projection	Residuals	Scaled Residuals	Pseudo-Optimized Residuals
1.50%						
Rate of True Positives	58.80%	74.19%	87.38%	55.80%	54.07%	54.97%
Rate of True Negatives	70.33%	67.00%	56.46%	76.14%	79.93%	76.72%
6.00%						
Rate of True Positives	61.11%	72.76%	86.90%	61.87%	60.84%	
Rate of True Negatives	84.64%	83.79%	77.39%	87.47%	89.15%	
25.00%						
Rate of True Positives	67.60%	77.44%	87.99%	71.06%	66.79%	
Rate of True Negatives	94.97%	93.81%	91.78%	94.17%	95.56%	

Table 6.4: Average rates of true positives and true negatives after simplification.

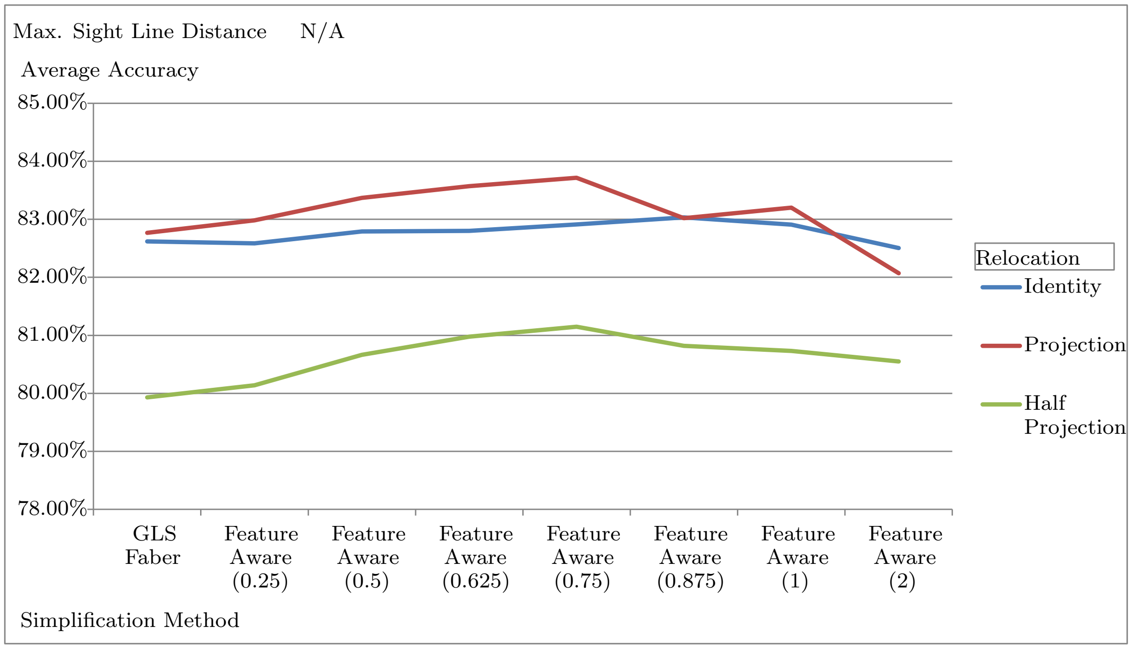
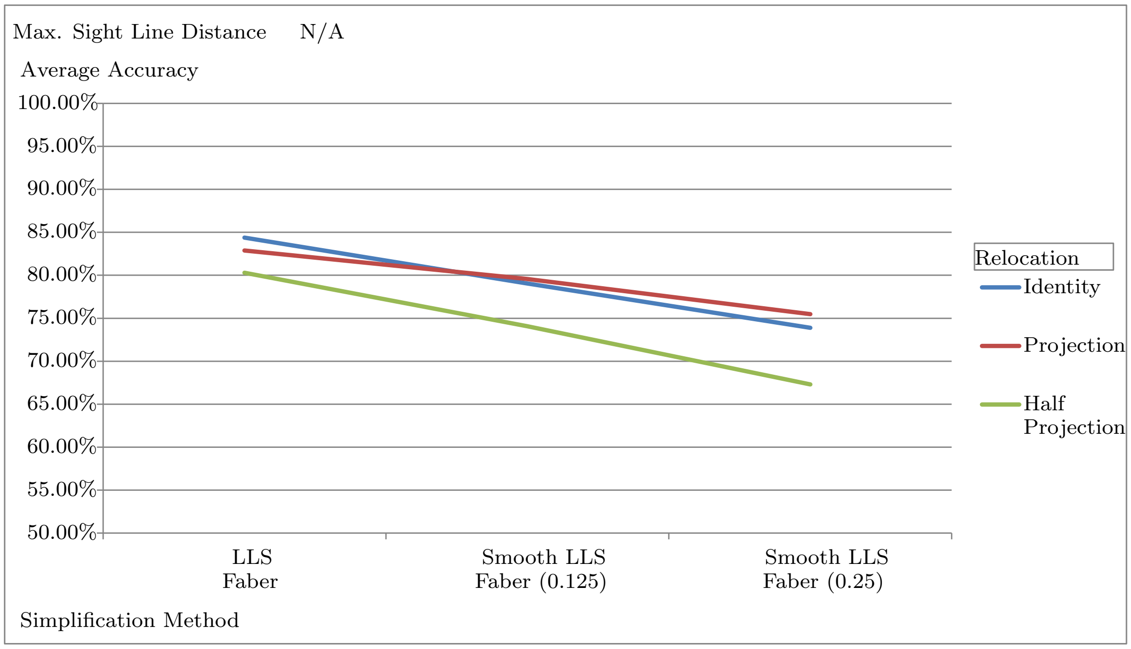
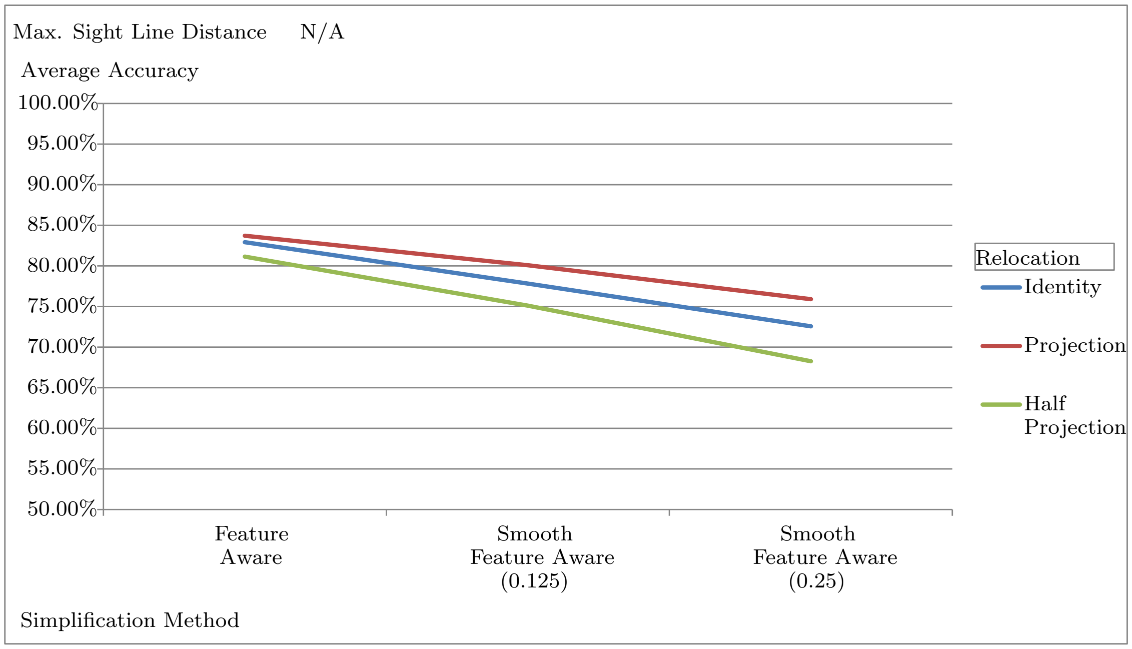


Figure 6.5: Average accuracy results after simplification to 1.5% using feature aware reverse subdivision with different feature weights.



(a) Smooth LLSRFS results.



(b) Smooth feature aware results.

Figure 6.6: Average accuracy results after simplification to 1.5% of the original terrain size using smooth reverse subdivision.

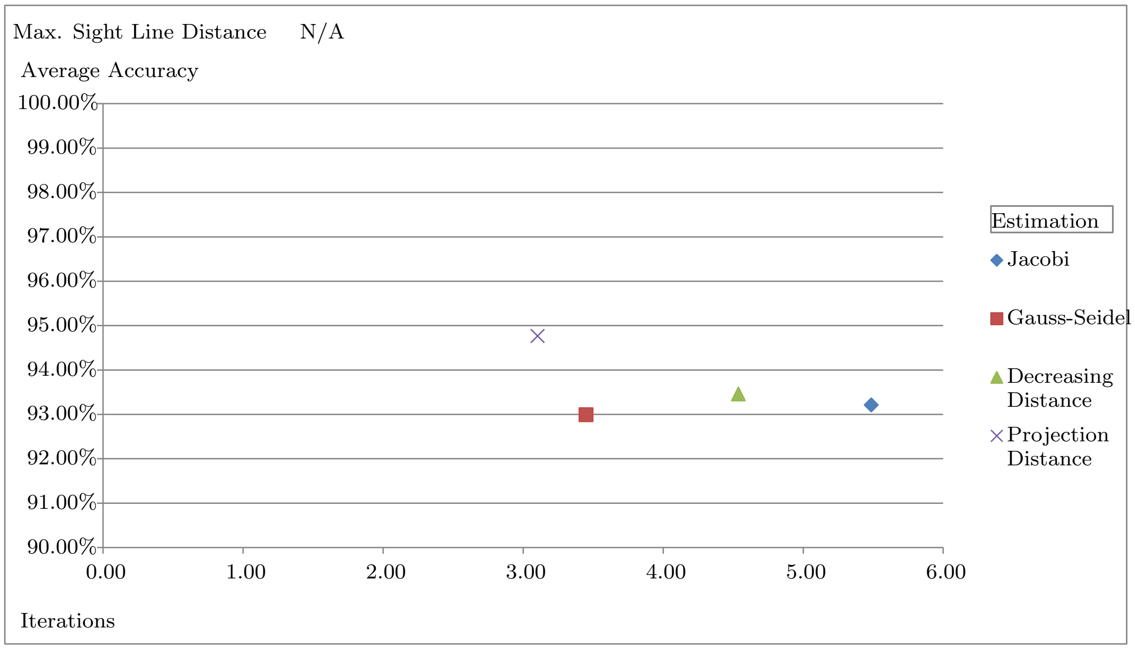


Figure 6.7: Summary of global accuracy vs iterations for the iterative estimation relocation methods averaged over the subsampling, LLSRFS, GLSRFS, feature aware, QEC, constrained QEC, greedy insertion, and greedy cuts simplification schemes.

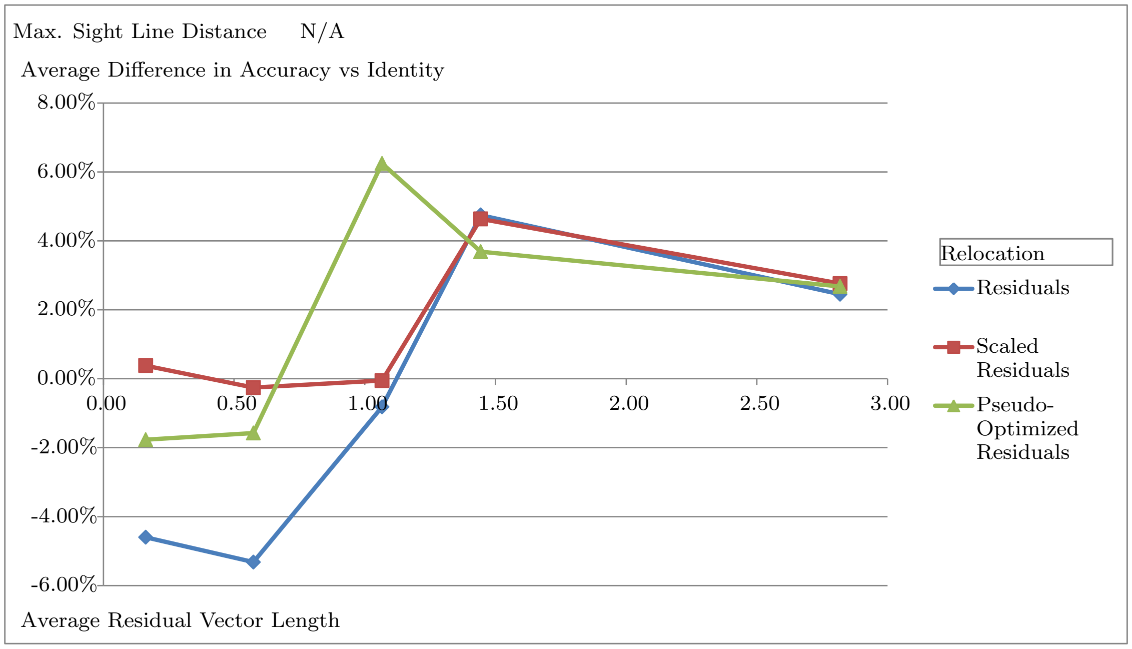


Figure 6.8: The average length of residual vectors plotted against the difference in accuracy (after simplification by LLSRFS) between the identity and each residual relocation function.

(a) Average accuracy results.

Max. Sight Line Distance	N/A

Average Accuracy	Jacobi	Gauss-Seidel	Decreasing Distance	Projection Distance
1.50%				
Subsampling	93.83%	93.39%	93.43%	95.05%
LLSRFS	94.61%	94.34%	94.79%	95.82%
GLSRFS	94.16%	93.90%	94.46%	95.31%
Feature Aware (0.75)	94.54%	94.04%	94.58%	95.58%
QEC	94.25%	93.94%	94.35%	94.41%
Constrained QEC	94.54%	94.53%	95.28%	95.67%
Greedy Cuts	88.51%	88.57%	89.50%	92.58%
Greedy Insertion	91.25%	91.24%	91.30%	93.68%
Grand Total	93.21%	92.99%	93.46%	94.76%

(b) Average rates of true positives and true negatives.

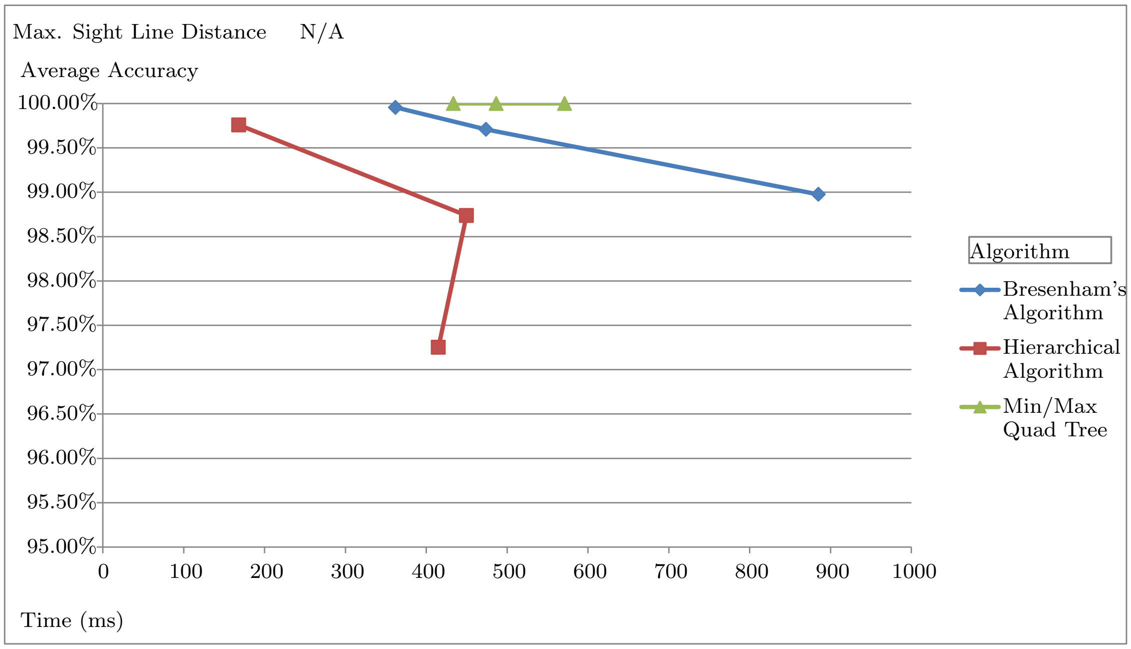
Max. Sight Line Distance	N/A

	Jacobi	Gauss-Seidel	Decreasing Distance	Projection Distance
1.50%				
Rate of True Positives	77.37%	76.83%	76.36%	76.96%
Rate of True Negatives	95.16%	94.97%	95.61%	96.93%

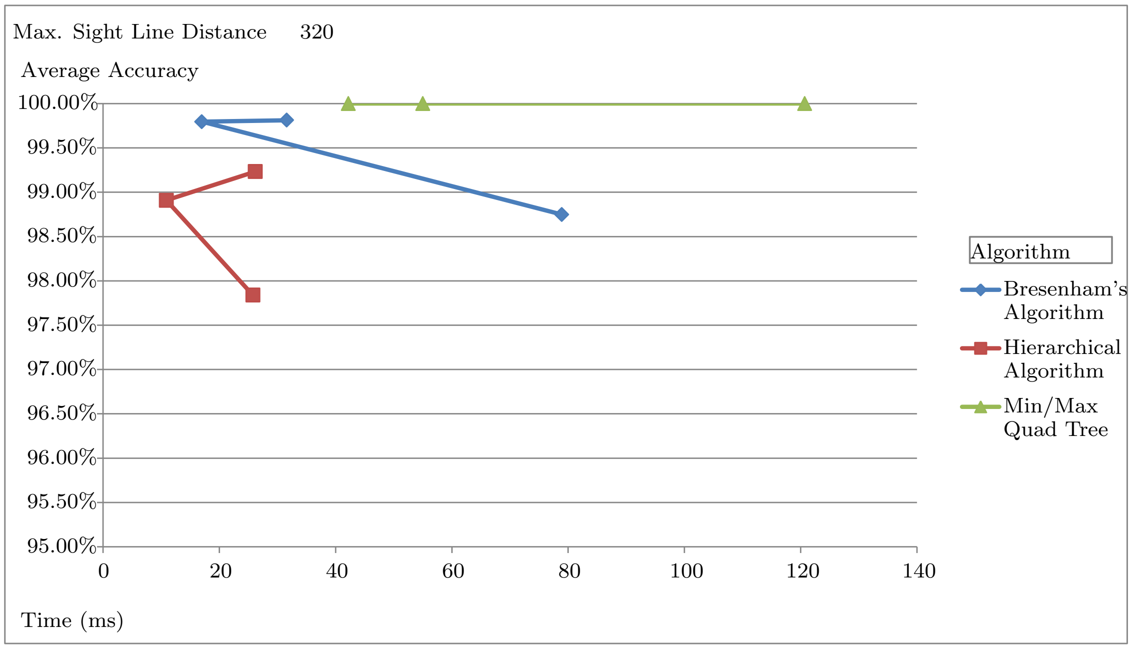
(c) Average number of iterations.

Average Iterations	Jacobi	Gauss-Seidel	Decreasing Distance	Projection Distance
1.50%				
Subsampling	5.37	4.10	5.17	3.57
LLSRFS	4.87	3.20	4.13	2.67
GLSRFS	5.47	3.47	4.63	2.73
Feature Aware (0.75)	5.13	3.37	4.47	3.07
QEC	6.23	3.83	4.90	3.87
Constrained QEC	5.20	3.13	4.27	3.07
Greedy Cuts	6.13	3.33	4.57	2.97
Greedy Insertion	5.50	3.17	4.17	2.90
Grand Total	5.49	3.45	4.54	3.10

Table 6.5: Average accuracy results and average number of iterations for the iterative relocation methods after simplification to 1.5%.



(a) Global query summary.



(b) Local query summary.

Figure 6.9: Summary of accuracy vs run-time for the LoS algorithms on each test terrain.

(a) Global algorithm results.

Max. Sight Line Distance	N/A

	Bresenham's Algorithm	Hierarchical Algorithm	Min/Max Quad Tree
Camp LeJeune			
Average Time (ms)	884.9	415	486.7
Rate of True Positives	97.35%	99.40%	100.00%
Rate of True Negatives	99.84%	96.09%	100.00%
Average Total Accuracy	98.98%	97.25%	100.00%
Kingston			
Average Time (ms)	473.9	449.8	571.1
Rate of True Positives	83.53%	80.06%	100.00%
Rate of True Negatives	99.90%	98.96%	100.00%
Average Total Accuracy	99.71%	98.74%	100.00%
Rocky Mountains			
Average Time (ms)	361.9	168	433.7
Rate of True Positives	81.79%	72.07%	100.00%
Rate of True Negatives	99.97%	99.79%	100.00%
Average Total Accuracy	99.96%	99.76%	100.00%

(b) Local algorithm results.

Max. Sight Line Distance	320

	Bresenham's Algorithm	Hierarchical Algorithm	Min/Max Quad Tree
Camp LeJeune			
Average Time (ms)	78.9	25.8	42.2
Rate of True Positives	97.30%	98.40%	100.00%
Rate of True Negatives	99.88%	97.39%	100.00%
Average Total Accuracy	98.75%	97.84%	100.00%
Kingston			
Average Time (ms)	17	10.9	55
Rate of True Positives	93.69%	91.44%	100.00%
Rate of True Negatives	99.89%	99.03%	100.00%
Average Total Accuracy	99.80%	98.91%	100.00%
Rocky Mountains			
Average Time (ms)	31.6	26.2	120.7
Rate of True Positives	90.77%	87.21%	100.00%
Rate of True Negatives	99.84%	99.28%	100.00%
Average Total Accuracy	99.81%	99.23%	100.00%

Table 6.6: Average accuracy and run-time results for the LoS algorithms. The min/max quad tree algorithm is considered to provide the correct result.

Chapter 7

CONCLUSIONS AND FUTURE WORK

Within this thesis, we have applied reverse subdivision methods as regularity-preserving terrain simplification to regular terrains in the interest of reducing the run time of LoS queries while preserving query accuracy. Using regularity-preserving simplification as opposed to irregularity-introducing simplification allows one to continue using the efficient data structures and algorithms associated with regular terrains.

Despite the additional constraint to preserve regularity that must be met by these schemes, our simplification methods have been shown to preserve LoS query accuracy comparably well to simplifications without such constraints. In particular, we have identified local least squares Faber reverse subdivision as a fast regularity-preserving simplification scheme that preserves LoS query accuracy well.

In the interest of improving upon these results, we introduced a novel reverse subdivision scheme intended to preserve features of the terrain that are important to visibility. Our feature aware reverse subdivision scheme adds additional constraints for maintaining the spatial relationships between the local minima and maxima of the terrain to the global least squares formulation of reverse subdivision. A feature weight parameter can be used to control the impact of these constraints on the final result.

LoS query accuracy is additionally impacted by the positions of the entities after simplification. We have explored the concept of point relocation and challenged the tacit use of projection relocation. Using an optimization framework for relocating entities to maximize LoS query accuracy after terrain simplification, we presented iterative estimation methods inspired by iterative numerical optimization techniques and showed that room for accuracy improvement exists over the standard projection function.

Achieving these improvements, however, remains a challenging task. To meet this challenge, we developed two practical relocation schemes. Half projection replaces the implicit constraint that “all entities must lie on the terrain” with the more permissive constraint that “no entity may lie beneath the terrain.” Residual vector relocation uses the residual vectors resulting from simplification as relocation vectors, generalizing projection. A pre-processing step that makes use of our estimation methods can be used to improve LoS accuracy further.

Unfortunately, each degree of simplification can only offer a constant improvement in run time while globally impacting LoS accuracy. To combat this, we have introduced a hierarchical approach to line-of-sight queries that combines these techniques with Bresenham’s algorithm and the min/max quad tree algorithm. The resulting algorithm has been shown to improve upon the speed of the min/max quad tree algorithm, with more consistent run times than the Bresenham algorithm, and a low drop in LoS query accuracy.

7.1 Future Work

Although we simplify all regions of the terrain uniformly in this work, the prospect of applying adaptive subdivision to allow different regions of the terrain to be approximated at different levels of fidelity is intriguing. While our hierarchical algorithm handles this in some respect, integrating level-of-detail directly into the simplification method is an avenue that we’ve yet to explore.

The concept of point relocation has proved to be a source of interesting research questions, and provides the bulk of our research contributions in this work. Further work into this area could be of potential interest, particularly in the area of developing new relocation methods for irregular terrains.

The distance threshold used in our formulation of the hierarchical algorithm has room for further exploration. For one, it could be interesting to consider the effect of different thresholds at each level of the hierarchy on the algorithm’s run time and accuracy, or a threshold based on something other than distance (e.g. the complexity of the terrain between two entities). Furthermore, while we keep the distance threshold constant throughout all our experiments, there exists the possibility to adjust the threshold on the fly to compensate for changes in an application's available resources. Using adaptively subdivided terrains within the hierarchy opens up further possibilities.

Although we place entities slightly above the terrain before simplification, the distance is kept constant and small. One wonders what the impact of our research would be in situations where entities are allowed to be airborne. Whether or not the same results will hold remains an open question.

The metric for LoS accuracy we have used is mathematically sound and features in existing literature, however it has some weaknesses. For instance, we have touched upon its strong bias towards negative results. While it goes beyond the scope of this work, it could be potentially interesting to examine the impact of different metrics for LoS accuracy.

Bibliography

[Abrash, 1997] Abrash, M. (1997). *Michael Abrash's Graphics Programming Black Book*, chapter 35. The Coriolis Group, Inc.

[Alderson and Samavati, 2012] Alderson, T. F. and Samavati, F. F. (2012). Reverse subdivision for optimizing visibility tests. In *Proceedings of the International Conference on Computer Graphics Theory and Applications*, GRAPP '12, pages 143–150.

[Alderson and Samavati, 2014] Alderson, T. F. and Samavati, F. F. (2014). Optimizing line-of-sight using simplified regular terrains. *The Visual Computer*. The final publication is available online at Springer via `http://dx.doi.org/10.1007/s00371-014-0936-3`.

[Andrade et al., 2011] Andrade, M. V. A., Magalhães, S. V. G., Magalhães, M. A., Franklin, W. R., and Cutler, B. M. (2011). Efficient viewshed computation on terrain in external memory. *Geoinformatica*, 15(2):381–397.

[Bartels and Samavati, 2000] Bartels, R. H. and Samavati, F. F. (2000). Reversing subdivision rules: Local linear conditions and observations on inner products. *Journal of Computational and Applied Mathematics*, 119(1-2):29–67.

[Ben-Moshe et al., 2002] Ben-Moshe, B., Mitchell, J. S. B., Katz, M. J., and Nir, Y. (2002). Visibility preserving terrain simplification: An experimental study. In *Proceedings of the 18th Annual Symposium on Computational Geometry*, SCG '02. ACM, New York, NY, USA.

[Ben-Moshe et al., 2007] Ben-Moshe, B., Serruya, L., and Shamir, A. (2007). Image compression terrain simplification. In *Proceedings of the 19th Canadian Conference on Computational Geometry*, CCCG '07, pages 125–128.

[Bresenham, 1965] Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30.

[Brosz et al., 2008] Brosz, J., Samavati, F. F., and Sousa, M. C. (2008). Terrain synthesis by-example. *Communications in Computer and Information Science: Advances in Computer Graphics and Computer Vision*, 4:58–77.

[Catmull and Clark, 1978] Catmull, E. and Clark, J. (1978). Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355.

[Chaikin, 1974] Chaikin, G. M. (1974). An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3(4):346–349.

[Cohen-Or et al., 2003] Cohen-Or, D., Chrysanthou, Y., Silva, C. T., and Durand, F. (2003). A survey of visibility for walkthough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431.

[De Floriani and Magillo, 1993] De Floriani, L. and Magillo, P. (1993). Algorithms for visibility computation on digital terrain models. In *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing: States of the Art and Practice*, SAC '93. ACM, New York, NY, USA.

[Doo and Sabin, 1978] Doo, D. and Sabin, M. (1978). Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360.

[Douglas and Peucker, 1973] Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122.

[Duvenhage, 2009] Duvenhage, B. (2009). Using an implicit min/max kd-tree for doing efficient terrain line of sight calculations. In *Proceedings of the 6th International Con-*

ference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa, AFRIGRAPH '09. ACM, New York, NY, USA.

[Dyn et al., 1987] Dyn, N., Levin, D., and Gregory, J. A. (1987). A 4-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4(4):257–268.

[Dyn et al., 1990] Dyn, N., Levine, D., and Gregory, J. A. (1990). A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169.

[Franklin and Ray, 1994] Franklin, W. R. and Ray, C. K. (1994). Higher isn't necessarily better: Visibility algorithms and experiments. In *Proceedings of the 6th International Symposium on Spatial Data Handling*.

[Funkhouser et al., 1992] Funkhouser, T. A., Séquin, C. H., and Teller, S. J. (1992). Management of large amounts of data in interactive building walkthroughs. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, I3D '92.

[Garland, 1999] Garland, M. (1999). *Quadric-Based Polygonal Surface Approximation*. PhD thesis, Carnegie Mellon University.

[Garland and Heckbert, 1995] Garland, M. and Heckbert, P. S. (1995). Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Carnegie Mellon University.

[Garland and Heckbert, 1997] Garland, M. and Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 1997*.

[Greene et al., 1993] Greene, N., Kass, M., and Miller, G. (1993). Hierarchical z-buffer visibility. In *Proceedings of SIGGRAPH 1993*.

[Guttman, 1984] Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference on Data Engineering*, pages 47–56.

[Hasan, 2013] Hasan, M. (2013). Balanced and feature-preserving multiresolution. Graphics Jungle Lab Presentation. University of Calgary. Calgary, AB, Canada.

[Heckbert and Garland, 1997] Heckbert, P. S. and Garland, M. (1997). Survey of polygonal surface simplification algorithms.

[Kiya et al., 1994] Kiya, H., Nishikawa, K., and Iwahashi, M. (1994). A development of symmetric extension method for subband image coding. *IEEE Transactions on Image Processing*, 3(1):78–81.

[Langetepe and Zachmann, 2006] Langetepe, E. and Zachmann, G. (2006). *Geometric Data Structure for Computer Graphics*, chapter 1. A K Peters, Ltd.

[Losasso and Hoppe, 2004] Losasso, F. and Hoppe, H. (2004). Geometry clipmaps: Terrain rendering using nested regular grids. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04. ACM, New York, NY, USA.

[Owen and Grigg, 2004] Owen, M. J. and Grigg, M. W. (2004). The compression of digital terrain elevation data (DTED) using JPEG2000. Technical Report DSTO-TR-1548, DSTO Information Sciences Laboratory, Australia.

[Prusinkiewicz et al., 2003] Prusinkiewicz, P., Samavati, F. F., Smith, C., and Karwowski, R. (2003). L-system description of subdivision curves. *International Journal of Shape Modeling*, 9(1):41–59.

[Rane and Sapiro, 2001] Rane, S. D. and Sapiro, G. (2001). Evaluation of JPEG-LS, the new lossless and controlled-lossy still image compression standard, for compression of high-resolution elevation data. *IEEE Transactions on Geoscience and Remote Sensing*, 39(10):2298–2306.

[Sadeghi, 2013] Sadeghi, J. (2013). *Smooth Reverse Subdivision.* PhD thesis, University of Calgary.

[Sadeghi and Samavati, 2009] Sadeghi, J. and Samavati, F. F. (2009). Smooth reverse subdivision. *Computers and Graphics*, 33(3):217–225.

[Sadeghi and Samavati, 2013] Sadeghi, J. and Samavati, F. F. (2013). Local fairing with local inverse. In *Proceedings of the 2013 Graphics Interface Conference*, GI '13.

[Samavati and Bartels, 1999] Samavati, F. F. and Bartels, R. H. (1999). Multiresolution curve and surface representation: Reversing subdivision rules by least-squares data fitting. *Computer Graphics Forum*, 18(2):97–120.

[Samavati et al., 2007] Samavati, F. F., Bartels, R. H., and Olsen, L. (2007). Local b-spline multiresolution with examples in iris synthesis and volumetric rendering. In *Image Pattern Recognition: Synthesis and Analysis in Biometrics*, volume 67 of *Series in Machine Perception and Artificial Intelligence*, pages 65–102. World Scientific Publishing.

[Seixas et al., 1999] Seixas, R. d. B., Mediano, M. R., and Gattass, M. (1999). Efficient line-of-sight algorithms for real terrain data. In *Proceedings of the Simpósio de Pesquisa Operacional y Logística da Marinha*, SPOLM '99.

[Serruya, 2011] Serruya, L. (2011). Image compression terrain simplification. Msc dissertation for applied project, IDC Herzliya.

[Silva and Mitchell, 1998] Silva, C. T. and Mitchell, J. S. B. (1998). Greedy cuts: An advancing front terrain triangulation algorithm. In *Proceedings of GIS 1998.*

[Silva et al., 1995] Silva, C. T., Mitchell, J. S. B., and Kaufman, A. E. (1995). Automatic generation of triangular irregular networks using greedy cuts. In *Proceedings of the IEEE Conference on Visualization 1995*, pages 201–208, 453.

[Sudarsky and Gotsman, 1999] Sudarsky, O. and Gotsman, C. (1999). Dynamic scene occlusion culling. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):13–29.

[VisTrails, 2013] VisTrails (2013). User:CSilva — VisTrailsWiki. `http://www.vistrails.org/index.php/User:Csilva`. [Online; accessed 25-April-2013].

[Wecker et al., 2010] Wecker, L., Samavati, F. F., and Gavrilova, M. (2010). A multiresolution approach to iris synthesis. *Computers and Graphics*, 34(3):468–478.

[Wikipedia, 2013a] Wikipedia (2013a). Bresenham's line algorithm — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/w/index.php?title=Bresenham%27s_line_algorithm&oldid=535828893`. [Online; accessed 31-January-2013].

[Wikipedia, 2013b] Wikipedia (2013b). File:Stabilization of Western Front WWI.PNG — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/File:Stabilization_of_Western_Front_WWI.PNG`. [Online; accessed 20-June-2013].

[Wikipedia, 2014] Wikipedia (2014). File: Bresenham.svg — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/wiki/File:Bresenham.svg`. [Online; accessed 03-February-2014].

[Wonka, 2001] Wonka, P. (2001). *Occlusion Culling for Real-Time Rendering of Urban Environments*. PhD thesis, Technischen Universität Wien.