The Visual Computer manuscript No.
(will be inserted by the editor)

Optimizing Line-of-sight Using Simplified Regular Terrains

Troy Alderson · Faramarz Samavati

Received: date / Accepted: date

Abstract In this work, we explore a set of techniques for speeding up line-of-sight queries whilst attempting to maintain accuracy. Line-of-sight queries, which test if two entities can see each other over a 3D terrain model, are an important operation in several applications. Given enough entities and a large enough terrain, computing these queries can be expensive. We apply reverse subdivision methods to simplify the terrain model and speed up the queries, including a novel feature-aware reverse subdivision scheme. To counteract the loss of accuracy due to simplification, we also examine the problem of where entities should be placed after terrain simplification to increase accuracy. Using iterative methods that attempt to maximize accuracy, we show that room for improvement exists over the standard projection method. Then, using residual multiresolution vectors, we develop a relocation method designed to maximize accuracy over simplified terrain models. Finally, we present a fast line-of-sight algorithm that combines these techniques with pre-existing algorithms.

Keywords Line-of-sight · Terrain Simplification · Multiresolution · Subdivision · Reverse Subdivision

Mathematics Subject Classification (2000) MSC 68U05 Computer graphics; computational geometry

1 Introduction

Terrains, and the operations that can be applied to them, comprise an important area of study in computer graphics and GIS. Line-of-sight (LoS) queries, in particular, can be an important operation in applications such as flight or military battlefield simulations. Given two entities on a terrain, a LoS query tests if the entities can see each other by determining whether or not the sight line between those two entities intersects the terrain (see Figure 1(a)). This involves traversing each face of the terrain that lies along the path of the sight line, and testing each for intersection with the sight line.

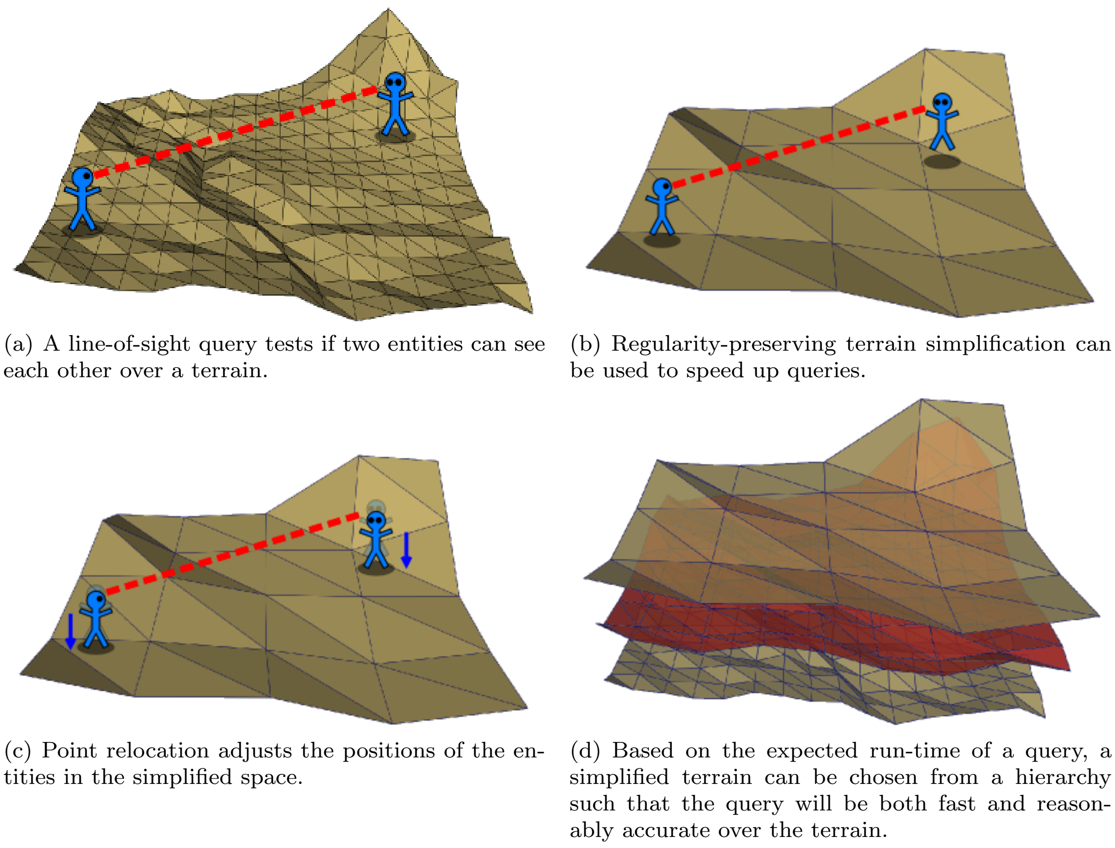
Although we treat entities as points, it is possible to compute visibility between 3D objects by casting multiple lines-of-sight from the observer position (e.g. an eye) to the viewed object, as in viewshed computation and supersample anti-aliasing.

Digital terrain models can be divided into two types. These are triangulated irregular networks and digital elevation models, often abbreviated to TINs and DEMs, respectively. DEMs, also known as height maps, are regular (valence 4) grids of equally spaced terrain elevation values, and are a well-accepted format for storing terrain data. Data from NASA's Shuttle Radar Topography Mission (SRTM), for instance, is stored in height map format. TINs are polygonal meshes, and are irregular in general.

The connectivity of height map terrains is implicit, allowing elevation values to be stored in simple array-like structures, whereas connectivity information for TINs must be explicitly stored. As a result, it is possible to traverse the elevation values of a DEM along a sight line's path analytically, whereas the faces of a TIN must be queried (usually from a separate spatial data structure) and then intersected. Hence, line-of-sight algorithms for DEMs tend to be faster than those for TINs [23]. (Note that a number of LoS algorithms for DEMs — for instance, those described in [12][23][10] — implicitly interpret DEMs as 3D bar charts with flat faces, as

T. Alderson
University of Calgary
E-mail: tfalders@ucalgary.ca

F. Samavati
University of Calgary



(a) A line-of-sight query tests if two entities can see each other over a terrain.

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

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

(d) 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.

Fig. 1 We approach line-of-sight query optimization via three techniques: regularity-preserving terrain simplification, point relocation, and a hierarchical algorithm.

opposed to continuous terrain meshes. They can be easily converted to 3D meshes by forming faces between the elevation values for use with LoS algorithms designed for TINs and for rendering.)

Even when using a fast algorithm, however, given a large enough terrain and sizeable number of entities the amount of time needed to compute the LoS queries between all entities will be substantial. This can become problematic for real time applications, particularly those with a large number of mobile entities whose visibility information changes from moment to moment (as in a military simulation). One approach to speeding up LoS queries is the use of terrain simplification. By reducing the number of faces in the terrain, we reduce the number of elevation values that must be checked within a LoS query (Figure 1(b)). This method produces gains in algorithm speed by a constant factor, at the cost of some accuracy.

In most applications, terrain simplification is expected to produce a simpler model that minimizes the vertical distance error with or is visually similar to the original model. Within this problem domain, however, the simplification is expected to maximize visibility test accuracy. While there appears to be some overlap, these differing goals suggest different simplifications.

Most simplification algorithms introduce irregularities into the terrain model. In the work of Ben-Moshe et al. [4], irregularity-producing simplification methods are used to speed up LoS queries. However, as noted above, LoS algorithms tend to be faster for completely regular terrains. In this and previous work [1], we consider the use of regularity-preserving terrain simplification in speeding up queries.

The regularity-preserving methods we consider are of the reverse subdivision paradigm, well-studied in the field of curve and surface modeling. Subdivision methods introduce additional vertices into a curve or surface with a constraint on geometric continuity. Reverse subdivision schemes are an approximate inverse of subdivision, and can be used as regularity-preserving simplification schemes. The two can be combined into a multiresolution framework, which allows one to transition between and synthesize various levels of resolution. We test the accuracy-preserving performance of several reverse subdivision schemes, including a novel scheme intended to maximize LoS query accuracy, against a selection of irregular simplification methods.

After simplification, entities are usually projected vertically onto the simplified terrain. This makes mathematical sense particularly for simplified terrains that minimize vertical distance error to the original terrain. However, for more general transformations and/or deformations of the terrain, as we are considering, a more robust, general method is needed.

Point relocation is the transformation of entity positions to maximize visibility test accuracy after terrain simplification (Figure 1(c)). We first consider an optimization framework for point relocation, with emphasis on a theoretical perspective rather than a practical real-time implementation to demonstrate that room for improvement exists over projection. Afterwards, we present a practical relocation method for regular terrains based on residual vectors.

Although simplification offers an increase in LoS query speed, this speed gain is constant based on the degree of simplification. Further speed increases require further simplification, and therefore further loss in accuracy. Drops in accuracy affect even those LoS queries that are fast without simplification (e.g. those between nearby entities). It would be ideal if only those LoS queries that require speed optimization would be sped up and affected by the accuracy drop. To this end, we introduce an LoS query algorithm that computes 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(d)). Each LoS query between two entities is computed over one of the terrain variants in this hierarchy, chosen based on the expected run-time of the query.

We present related work in Section 2. Definitions used throughout the paper, including what we mean by test accuracy, are given in Section 3. Descriptions of our terrain simplification methods (Section 4), point relocation methods (Section 5), and hierarchical LoS algorithm (Section 6) follow. Finally, Section 7 contains comparison results and discussions for our methods, followed by Section 8 with conclusions and future work.

2 Related Work

De Floriani and Magillo [8] identified three variants of the visibility problem and presented some algorithms for solving them. The three variants are point-to-point visibility, point-to-line visibility, and point-to-region visibility (also known as *viewshed analysis*). Our work is primarily concerned with the problem of point-to-point visibility.

The implicit connectivity and 2D array indexing for regular terrains suggest that algorithms on regular terrains are more efficient than those for irregular terrains. Seixas et al. [23] compared the run times of two visibility algorithms which operate on similar principles—Bresenham's line algorithm [5] for regular terrains and an R3-tree algorithm [16] for irregular terrains—and found the Bresenham algorithm to be more efficient in both run time and memory usage.

Our work pays particular attention to two different LoS query algorithms on regular terrains: Bresenham's algorithm and the min/max quad tree algorithm. Bresenham's line algorithm for plotting raster lines was originally described in [5], and can be easily adapted to traverse a sight line's path over an elevation grid and compare the height of the sight line to the terrain height (as described in [23]). Spatial subdivision can be used to produce an algorithm that is asymptotically faster than Bresenham's algorithm in the average case. Duvenhage's min/max quad tree algorithm is such an algorithm, and is described in [10]. Brief descriptions of both algorithms can be found in Section 6.1.

Franklin and Ray [12] presented a fast algorithm for viewshed analysis on regular terrains, known as Xdraw. Andrade et al. adapted Franklin and Ray's algorithm to perform this analysis on terrain in external memory [2]. I/O operations on external memory form the bottleneck for the algorithm, and so are 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 is a powerful tool for reducing memory usage.

In [4], Ben-Moshe et al. experimented with terrain simplification to speed up individual LoS queries. They presented a novel terrain simplification algorithm specifically designed to preserve LoS query accuracy and compared its performance against three other simplification methods. They found their algorithm preserved visibility accuracy the best of the tested simplification methods. Note, however, that the simplification methods studied in their paper all introduce irregularities into the terrain model in the general case.

Forward and reverse subdivision have gained prominence in recent years as an important geometric modeling technique, the latter of which can be used for terrain simplification. These schemes form the basis of multiresolution frameworks, which have been studied by Samavati et al. [22]. Multiresolution has several applications, including but not limited to multi-scale editing, iris and terrain synthesis [27][6], and terrain void patching [26].

Samavati and Bartels have studied the reversal of subdivision schemes in detail. In [21], they use least squares data fitting to approximate a curve using a coarser representation. In [3], they limit least squares data fitting to local neighbourhoods to obtain local reverse subdivision matrices.

In our prior work, we have shown that regularity-preserving terrain simplification can be easily achieved using reverse subdivision schemes on curves [1]. By applying a curve subdivision scheme to the rows and columns of a regular terrain, the resulting terrain will also be regular. Some well-known curve subdivision schemes include Chaikin's corner cutting scheme [7] and the interpolatory scheme of Dyn et al [11].

Our hierarchical algorithm bears some similarity to the geometry clipmaps from Losasso and Hoppe's 2004 paper [18]. They generate a viewer-centric hierarchy for level-of-detail control in rendering, using Dyn-Levin subdivision to synthesize finer levels of the hierarchy. Rather than rendering, however, our work is concerned with the use of a reverse subdivided hierarchy in performing LoS queries. In a sense, our hierarchical algorithm takes a level-of-detail approach to LoS queries.

3 Definitions

Consider a terrain model T and a set of 3D points $P = \{p_1, p_2, \ldots, p_n\}$ on T. Let $O(p)$ be the observer positions for any $p \in P$, which is offset 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

$$V_T(p_1, p_2) = \begin{cases} 1 & \text{if the sight line between} \\ & O(p_1) \text{ and } O(p_2) \text{ does not} \\ & \text{intersect } T \\ 0 & \text{otherwise.} \end{cases}$$

Note that $V_T(p_1, p_2) = V_T(p_2, p_1)$.[1]

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

$$A(T, p_1, p_2) = \begin{cases} 1 & \text{if the sight line between} \\ & O(p_1) \text{ and } O(p_2) \text{ is thought to} \\ & \text{not intersect } T \\ 0 & \text{otherwise.} \end{cases}$$

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 [4], to be the ratio of the number of queries computed correctly to the total number of queries. More formally,

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

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 to not intersect (resp. intersect) the terrain to the total number of not intersecting (resp. intersecting) queries.

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

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

4 Terrain Simplification

Advances in technology are allowing for the creation of very high resolution terrain models. The greater detail on these terrains increases the run-time of algorithms that operate on them, such as LoS queries. Simplifying such terrain models not only reduces their memory usage, but also increases the running speed of said algorithms.

Different terrain simplification methods have different effects on a terrain's topology and on LoS query accuracy. In this section, we describe the simplification methods used in our experiments and comparisons.

4.1 Irregular Simplification Methods

In the work of Ben-Moshe et al. [4], the authors' novel simplification algorithm was tested against three different simplification algorithms that introduce irregularities in the general case. For convenience, we refer to them as irregular simplification methods. In the interest of allowing our work to be compared with theirs, we compare our regularity-preserving reverse subdivision schemes against these same irregular methods.

These methods are:

[1] In a realistic simulation, visibility can be impacted and rendered non-commutative by factors such as weather, eyesight impairments and limitations, the direction in which an entity is facing, etc. For the purposes of this work, we ignore all factors other than the terrain that can impact visibility for either entity.

1. Garland and Heckbert's [15][13] **quadric error metric based edge collapse**, which we abbreviate to QEC for short.
2. Garland and Heckbert's [14] **greedy insertion algorithm**, a generalization of the Ramer-Douglas-Peucker algorithm for approximating curves [19][9] to 3D polygonal meshes.
3. Silva et al.'s [25][24] **greedy cuts algorithm**.

4.2 Regularity-Preserving Simplification Methods

Our work concentrates on using reverse subdivision as a paradigm of regularity-preserving simplification. In this section, we provide a brief overview of subdivision and the various approaches to reversing it, followed by a description of the reverse subdivision schemes we have tested.

We limit our discussions to curves, as they are easier to understand and can be easily generalized to the rows and columns (u- and v-curves) of regular terrains.

4.2.1 Subdivision and Reverse Subdivision

Subdivision is a family of refinement methods that increase the resolution of curves and surfaces. These methods introduce new vertices into a curve or surface and adjust the positions of old vertices such that the limit curve — resulting from repeated applications of subdivision — satisfies a geometric continuity constraint.

Several subdivision methods converge to B-Spline curves at the limit. Chaikin's well-known corner-cutting scheme [7], for instance, converges to a third-order B-Spline curve. Faber subdivision, named for Georg Faber [21], which introduces midpoints between vertices (see Figure 2), converges to a second-order B-Spline curve.

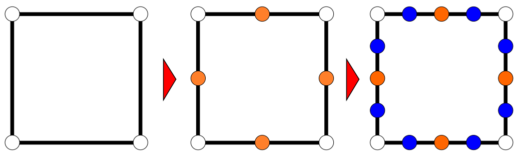


Fig. 2 Faber subdivision applied to a curve. The shape of the curve remains unchanged; only the number of vertices is affected. Orange vertices are introduced after one application of subdivision, blue vertices after two.

A convenient matrix notation has arisen for subdivision schemes, which are linear operations. A subdivision operation can be represented as a matrix P applied to a vector of coarse points c, resulting in a vector of refined points $f = Pc$. The Faber subdivision matrix, for instance, has the general form

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

Notice here that P in this case is sparse and banded, and features a repeating local pattern along the columns. Many subdivision schemes share this trait and can be more compactly represented as an ordered list of filter values. The filter values for Faber subdivision are

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

The effect of subdivision can be reversed using reverse subdivision. Reverse subdivision schemes simplify a curve by removing vertices and adjusting the positions of remaining vertices such that the resulting coarse curve, if subdivided by the corresponding subdivision scheme, will yield approximately the original curve. In general, high resolution details of the curve are lost in the process, requiring additional data for perfect reconstruction of the original curve.

Hence, for a given subdivision scheme, the reverse scheme is not unique. There exist several different general approaches to reverse subdivision.

Global least squares reverse subdivision, as formulated in [21], minimizes the least squares error $||Pc - f||_2$ between the subdivided coarse curve Pc and original curve f by solving the overdetermined system $Pc = f$ for c. The solution is a reverse subdivision matrix $A = (P^T P)^{-1} P^T$, where $c = Af$. Unlike their corresponding subdivision matrices P, in general these A are not sparse or banded and do not feature a repeating local pattern, and must be recomputed for each size of f.

Local least squares reverse subdivision [3] addresses this problem. Here, a local reverse subdivision matrix A_L is determined via least squares optimization in a local window of the data points, i.e. using a local subdivision matrix P_L. An ordered list of mask values can be extracted from the rows of A_L and used as part of a general formula for constructing reverse subdivision matrices A for any size of f.

For instance, reversing a 3×5 local Faber subdivision matrix (the smallest affine local subdivision matrix that can be formed for Faber subdivision) results in a reverse subdivision matrix with the general form

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

compactly represented by mask values

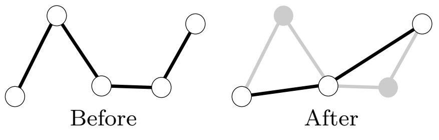
$$\{-\tfrac{1}{6}, \tfrac{1}{3}, \tfrac{2}{3}, \tfrac{1}{3}, -\tfrac{1}{6}\}.$$

4.2.2 Reverse Subdivision Methods

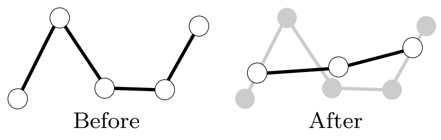
Although Faber subdivision is not a smooth subdivision scheme (it converges to a polyline with C_0 continuity) and thus undesirable for many modeling applications, it is precisely because Faber is not smooth that we use its reverse schemes for our work. Often, the reverse schemes of smooth subdivision schemes exaggerate the shape of the curve to compensate for the smoothing effect. Although energy minimization can be applied to make the result smooth [20], intuitively reverse Faber subdivided terrains should remain close to the original terrain shape even without energy minimization since Faber subdivision has no effect on terrain shape.

Additionally, as with other reverse subdivision schemes, reverse Faber subdivision features a convenient correspondence between the rows and columns of the original and simplified terrains and does not suffer from a shrinking boundary.

We have tested a selection of pre-existing reverse subdivision methods (see Figure 3) based on Faber subdivision in the context of speeding-up LoS queries. These are



(a) A curve before and after subsampling.



(b) A curve before and after reverse Faber subdivision.

Fig. 3 Reverse subdivision methods.

1. **Subsampling** by a factor of 2. The simplest scheme to reverse Faber subdivision is to discard every other point, assuming it is a midpoint introduced by the subdivision scheme.
2. **Global least squares reverse Faber subdivision**, described above.
3. **Local least squares reverse Faber subdivision**, described above.

4.2.3 Feature-Aware Reverse Faber Subdivision

In addition to these pre-existing reverse subdivision methods, we present a novel reverse subdivision scheme intended to better preserve terrain shape by identifying vertices of the curve critical to the terrain's important features (i.e. ridges and valleys) and preserving their positions relative to each other. Our scheme identifies these *critical points* and uses least squares optimization to preserve the vectors between them. We call this scheme **feature-aware reverse Faber subdivision**.

The critical points are found similarly to the ridge network computation in the work of Ben-Moshe et al [4]. In their work, an edge shared by two terrain faces is considered to be a local ridge or channel if the incident faces have opposite slopes. In our work, a vertex between two curve edges is considered to be a local maximum or minimum (therefore a critical point) if the edges incident to the vertex have opposite slopes (see Figure 4). We take the local maxima and minima and the endpoints of the curve to be the feature-critical points.

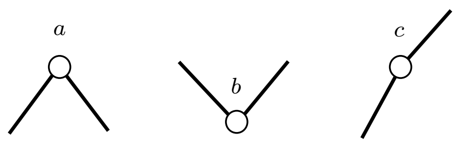


Fig. 4 Identification of critical points. Points a and b are critical, since their incident edges have opposite slopes, whereas point c is not critical.

Let $p \subseteq f$ denote the set of critical points and $\mathbf{v}_i = p_{i+1} - p_i$ be the vectors between neighbouring critical points (see Figure 5).

We preserve the relative positioning of the p_i by augmenting the linear system $Pc = f$ with additional constraints, one for each vector $\mathbf{v}_i$. We use a *feature weight* parameter $w \in \mathbb{R}$ where $w > 0$ to control the strength of the feature preservation.

For example, the weighted linear system for the curve shown in Figure 5 would be

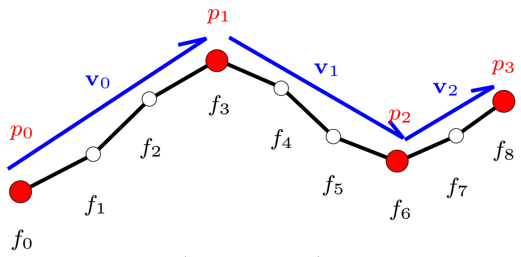


Fig. 5 Critical points (shown in red) and the vectors between them (shown in blue).

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

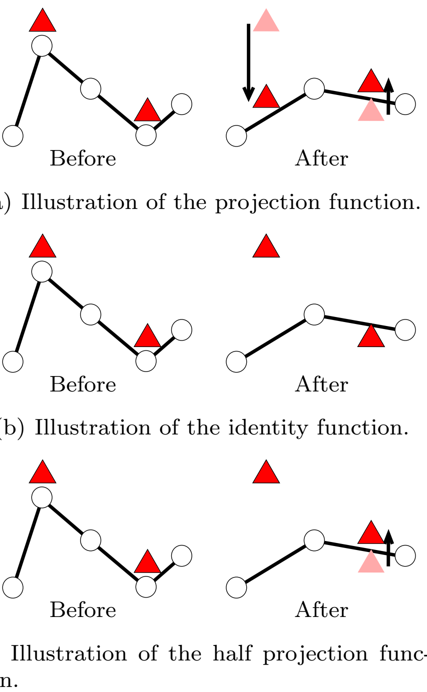
We append the vectors $w \cdot \mathbf{v}_i$ to the end of f and append additional rows, one for each of the $\mathbf{v}_i$, to P. For a given i, the row corresponding to $\mathbf{v}_i = f_j - f_k$ is given exactly two non-zero entries: $+w$ for the coarse point $c_{\lfloor j/2 \rfloor}$ and $-w$ for the coarse point $c_{\lceil k/2 \rceil}$. This adds the equations $w \cdot \mathbf{v}_i = w \cdot (c_{\lfloor j/2 \rfloor} - c_{\lceil k/2 \rceil})$ to the linear system. If $\lfloor j/2 \rfloor = \lceil k/2 \rceil$, the row is discarded.

5 Point Relocation

Terrain simplification, in general, results in perturbations in the terrain shape. This raises an important question: how should the entity positions be perturbed in order to accompany changes in the terrain shape? Should they be left unchanged, projected to the new terrain, or changed in some other manner? In this section, we present our approaches to this question under the label of **point relocation**.

For comparison, the point relocation methods described in the following sections are tested against the following three relocation methods (see Figure 6):

1. **Projection relocation**, the accepted method for point relocation (for instance, Ben-Moshe et al. [4] have it built-in into their problem definition) which projects all entities vertically onto the simplified terrain.



(a) Illustration of the projection function.

(b) Illustration of the identity function.

(c) Illustration of the half projection function.

Fig. 6 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).

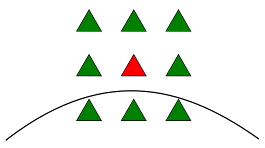
2. **Identity relocation**, which leaves the entity positions unchanged. We use this relocation method to provide an LoS accuracy baseline for our tests.
3. **Half projection relocation**, a custom relocation method that combines identity and projection relocation, projecting entities in the half-space beneath the simplified terrain up onto the simplified terrain. This method sacrifices total LoS query accuracy in order to favour true positives.

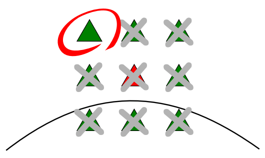
We first use an optimization framework to show that room for improvement exists over the projection method. Afterwards, we introduce a generalization of projection relocation for regular terrains, and apply our optimization framework as part of a pre-processing step to maximize LoS query accuracy.

5.1 Optimization Problem

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:

$$\min_R \sum_{i \neq j} |V_T(p_i, p_j) - A(S(T), R(p_i), R(p_j))|. \quad (1)$$


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


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

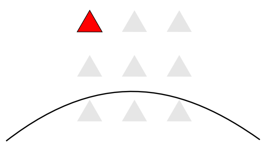

(c) The entity is relocated.

Fig. 7 Illustration of our iterative method in 2D. The entity's current position is highlighted in red. The terrain is represented by the curved line.

Note that the presence of V_T in Equation 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.

5.2 Estimation Methods

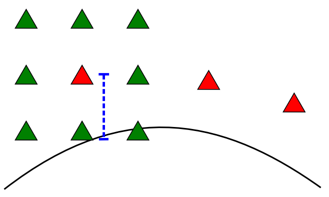
While a closed form solution does not appear to exist, the optimal solution to Equation 1 can be estimated using discrete, iterative methods. We present here our methods for estimating the optimal solution, designed to emulate iterative numerical optimization techniques in a discrete geometric setting (see Figure 7).

For each method, we begin with an initial guess at the optimal entity positions. Iteratively, we refine these positions until no further improvements in LoS accuracy can be found or a maximum number of iterations is reached.

The pre-simplification entity positions are used as an initial first guess at the optimal post-simplification positions. At each iteration, we discretize a grid of candidate positions about each entity, test the LoS accuracy at each candidate position, and then relocate the entities to the candidate position that produces the highest accuracy.

While none are guaranteed to determine the optimal solution, we have tested four different implementations of this general method. These methods are

1. **Jacobi iterative estimation**, in which we discretize the candidate grid using a constant distance and relocate entities once the new positions have been identified for all entities.


(a) Local neighbourhoods of candidate positions are discretized using the distance to the simplified terrain (shown by the dashed line).

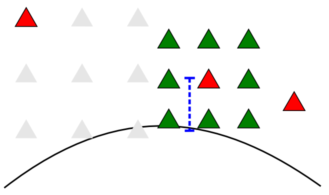

(b) Entities are relocated one at a time.

Fig. 8 Illustration of the projection distance approach in 2D. The entity's current position is highlighted in red, with candidate positions shown in green. The terrain is represented by the curved line.

2. **Gauss-Seidel iterative estimation**, in which the candidate grid is discretized using a constant distance and each entity is relocated once its new position is identified.
3. **Decreasing distance iterative estimation**, in which the candidate grid distance decreases by a constant factor with each iteration and each entity is relocated once its new position is identified.
4. **Projection distance iterative estimation**, inspired by projection relocation. The candidate grid is discretized using the vertical distance between the entity and the simplified terrain (i.e. the distance of projection) and each entity is relocated once its new position has been identified. See Figure 8.

Among our estimation methods, our results have shown that projection distance relocation preserves LoS accuracy the strongest, and indicate that room for improvement in LoS query accuracy does exist over the projection relocation method.

5.3 Point Relocation by Residual Vectors

We have attempted to realize these improvements in LoS accuracy using a point relocation approach based on multiresolution residual vectors, which we refer to as **residual vector relocation**.

Reverse subdivision simplifies a curve of fine points f to a curve of coarse points c such that $Pc \approx f$. Since Faber subdivision does not change the shape of the curve, the changes in geometry due to simplification are encapsulated by the residual vectors $r = Pc - f$. Hence, these residual vectors can be used as translation vectors to project entities from the original curve to the simplified curve. Notice that $f + r = f + (Pc - f) = Pc$, which are points on the simplified curve.

Although the residual vectors will project entities onto the simplified curve, this projection is not necessarily completely vertical. Residual vectors reflect changes in the (x, y, z) coordinates between f and Pc while projection only reflects changes in one coordinate (namely, the z coordinate). Hence, residual vector relocation is a generalization of projection.

The residual vectors $r = Pc - f$ can be computed directly from the original and simplified curves by

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

This can be generalized to n steps of subdivision as follows: let c^k be the vector of coarse points after $n - k$ applications of reverse subdivision on f and P^k be the subdivision matrix for the given size of c^k, for $k = 0, \ldots, n-1$. Then the residual vectors are given by $r = P^{n-1} \cdots P^1 P^0 c^0 - f$, 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:

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

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

Notice that the residual vectors are only defined at the vertices of the original terrain. Entities that do not lie at one of the original terrain's vertices (i.e. that are not in f) lie over an edge or face of the original terrain. In these cases, we take a barycentric combination of the residual vectors of the edge's endpoints/face's corner points to be the entity's translation vector.

Taken all together, these vectors define a vector space over the original terrain that allows us to determine relocation vectors for each entity in real time.

5.4 Pseudo-Optimized Residual Vector Relocation

For further improvements in LoS query accuracy, we propose the use of iterative estimation in a pre-processing step to estimate the optimal relocation vectors. Here, we apply a process similar to line-search optimization, estimating the optimal relocation vector in the direction of the residuals. As we again cannot guarantee that the optimal solution is found, we refer to this method as **pseudo-optimized residual vector relocation**.

Our pre-processing step uses iterative estimation to estimate optimal scaling factors for each residual vector. Each residual vector corresponds to a vertex of the original terrain. To prevent the run-time of the pre-processing step from spiralling out of control on large terrains, we calculate scaling factors for only a subset of the residual vectors (those corresponding to vertices of the simplified terrain) and compute LoS accuracy using a *local* neighbourhood of entities for each vector.

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 residual vector r_i, with LoS accuracy computed relative to a local neighbourhood of entities e_j.

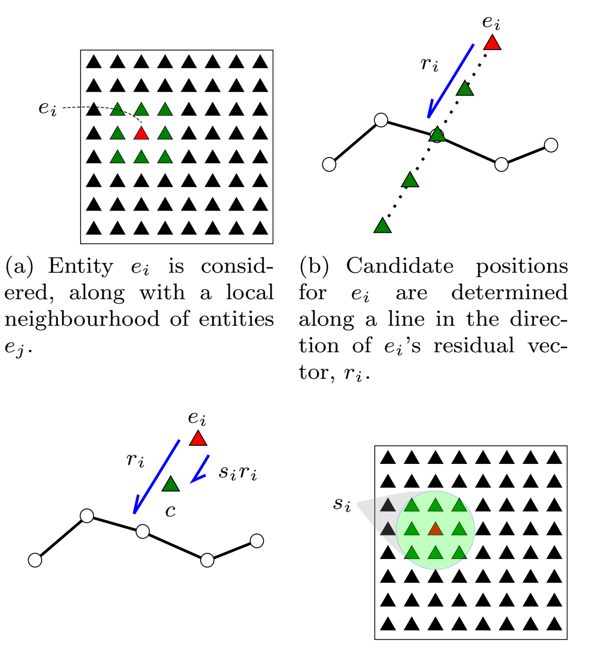
Given optimized positions for each entity e_i, we can determine optimized relocation vectors for the e_i. These relocation vectors have the form $s_i r_i$, where s_i is a scaling factor. Finally, we distribute scaling factors to all the relocation vectors in the vector field and scale them.

That is, at each iteration (see Figure 9), 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. $r_i' = s_i \cdot r_i$).

Residual vectors that do not correspond to one of the simplified terrain's vertices lie over an edge or face of



(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 residual 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.

Fig. 9 Illustration of the optimized residuals pre-processing step.

the simplified terrain. In these cases, we bilinearly interpolate the scaling factors of the edge's endpoints/face's corner points to find a scaling factor for the relocation vector.

We have found point relocation to be a powerful tool for improving LoS accuracy after simplification. However, there are impacts on accuracy introduced by simplification that no general purpose relocation scheme can address. For instance, consider a region of the original terrain simplified to a single face. Under many relocations, all the entities over that region will be able to see other, regardless of their initial visibility relations. Global simplifications, which address global speed concerns for LoS queries, wreak havoc on local LoS accuracy.

Point relocation, by itself, cannot address these accuracy issues. A new algorithmic approach, however, can.

6 Hierarchical LoS Algorithm

For our intended application — real-time simulations, particularly military battlefield simulations — LoS queries need to be fast *and* accurate in local areas as combatants converge on strategic locations and battlefields. Limitations on effective sight line distance can reinforce the importance of fast local queries. However, on a global scale — when combatants are spread out and do not have limitations on sight line distance — LoS queries also need to be fast.

Here we present an algorithm that only speeds up LoS queries expected to be too slow, as determined by a user-defined distance threshold. Our algorithm combines several different techniques, using their respective strengths for improved LoS query computation.

6.1 Bresenham's Algorithm and the Quad Tree Algorithm

Bresenham's line algorithm is well-known within the computer graphics community as an algorithm for plotting a line segment on a raster image. As it traverses the raster cells between the line segment's endpoints, the algorithm tracks the error between the center of the current raster cell and the actual line segment. At each step, the raster cell that minimizes the error is identified and moved to. The algorithm can be easily adapted to serve as a line-of-sight algorithm on regular terrains (which are raster grids of elevation values) by replacing the default drawing behaviour with a height comparison between the terrain's elevation and the height of the sight line [23].

Bresenham's algorithm has no memory overhead, and its run-time is $O(d)$, where d is the L_1-norm (or Manhattan distance) between the entities over the regular terrain grid. Hence, the Bresenham algorithm runs fastest when the entities are near to each other (relative to the terrain grid) or when the entities cannot see each other (e.g. when the terrain has many sharp features).

Duvenhage's **min/max quad tree algorithm** [10] uses a quad tree to quickly cull regions of the terrain from the query computation. At each node of the quad tree, the maximum and minimum elevations of its child nodes are stored (leaf nodes contain the elevation for a grid cell of the regular terrain). If the sight line lies completely above a node's maximum elevation, we can avoid testing the child nodes. If the sight line lies partially below a node's minimum elevation, then the query may return 'not visible' at once. If neither case holds, the child nodes must be visited. Fast child node intersection may be accomplished using the ray tracing technique in [17].

The quad tree algorithm features a quad tree over the regular terrain as memory overhead, and runs in $O(\log d)$ time on average, $O(d \log d)$ in the worst case (again, d is the Manhattan distance between entities). Due to culling, the algorithm works best on relatively flat terrains. Lack of overhead for Bresenham's algorithm makes it faster than the quad tree algorithm in cases where d is sufficiently small.

Using regularity-preserving terrain simplification, the Manhattan distance d can be reduced between all pairs of entities, which speeds up each algorithm and reduces memory usage. The regularity-preserving simplification methods described in this paper cut d in half with each application.

6.2 Algorithm Details

Our **hierarchical LoS algorithm** seeks to combine the strength of Bresenham's algorithm given small d, the strength of the quad tree algorithm given large d, and the power of speed-ups due to simplification. It is based on a simple observation: given a LoS query, we can choose, based on its expected execution time, a LoS algorithm and simplified terrain that will best meet the application's run-time and accuracy requirements. In other words, we can take a level-of-detail approach to LoS queries: queries expected to take too long to compute can be computed with less fidelity.

Using regularity-preserving terrain simplification, we generate a hierarchy of simplified regular terrains (say $T_1, T_2, \ldots, T_{top}$ where $T_1 = T$) and add a min/max quad tree on one of the levels (say T_{tree}). Two distance thresholds, t_B and t_Q, are defined by the user and determine when d is sufficiently small to use the Bresenham algorithm ($d \le t_B$) and when it is sufficiently large to use the quad tree algorithm ($d > t_Q$).

If $d > t_Q$, then the query is computed using the quad tree algorithm on T_{tree}. Otherwise, the query is computed with Breseham's algorithm on T_i, where $i \in \mathbb{Z}^+$ is the smallest value such that $(\frac{1}{2})^{i-1}d < t_B$. We take the level index $tree$ to be $\lceil \log_2(\frac{t_Q}{t_B}) \rceil + 1$, to ensure consistent accuracy with queries as $d \to t_Q$.

Pseudocode for the algorithm is given in Algorithm 1.

Algorithm 1 HierarchicalLoS(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 QuadTreeLoS($T_{tree}, p_1, p_2$)
else
  while $dist > t_B$ AND $k < top$ do
    $k := k + 1$
    $dist := dist/2$
  end while
  return BresenhamLoS($T_k, p_1, p_2$)
end if
```

It is trivial to note that by setting $t_Q = \infty$, or by placing a sight line distance limitation $< t_Q$, the algorithm may be used without the quad tree.

6.3 Runtime Analysis

The average runtime of our algorithm is given piecewise as

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

and the worst case runtime is

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

Without the quad tree (i.e. with $t_Q = \infty$), the asymptotic run-time of the algorithm collapses down to $O(t_B)$. Notably, this run-time is independent of the terrain size and entity positions, hence, given a constant t_B and $t_Q = \infty$, the algorithm can be run in effectively constant time (albeit with significant drops in accuracy).

We show the derivation for the algorithm run-time assuming $t_Q = \infty$. The piecewise run-time given above follows from this derivation and from the run-time of the 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 $d_i \le t_B$, we use the $O(d_i)$ time Bresenham line algorithm on T_i. Hence, the runtime of our algorithm can be given piecewise as

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

Notice that $d_i \le t_B$ in each case. Hence, we can replace the $O(d_i)$ by $O(t_B)$ in Equation 2 to yield

$$\begin{cases} O(t_B) \text{ when } d_1 \le t_B \\ O(t_B) \text{ when } \frac{1}{2}t_B < d_2 \le t_B \\ \cdots \\ O(t_B) \text{ when } \frac{1}{2}t_B < d_{top} \le t_B. \end{cases} \tag{3}$$

Equation 3 collapses down to

$$O(t_B), \tag{4}$$

which is what we wanted to show.

It should be noted that, since the Bresenham line algorithm reaches its worst-case run time when the sight line does not intersect the terrain, 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.

7 Results

In this section we describe our results for the LoS query accuracy resulting after application of terrain simplification and point relocation, in addition to the LoS accuracy and run-time of our hierarchical algorithm.

All distances given are relative. A distance of 1 is taken to be the horizontal distance between neighbour vertices in the original terrain.

7.1 Simplification and Relocation Results

For each pair of simplification method and point relocation method, we computed the resulting LoS query accuracy on five 120×120 sized height map terrains at three levels of simplification — 25%, 6%, and 1.5% of the original terrain size (corresponding to 1, 2, and 3 applications of reverse subdivision) — with limitations on sight line distance (i.e. the Manhattan distance d between entities) and at 1.5% of the terrain size without such limitations. In the limited case, we used a maximum sight line distance of 30.

Queries were performed between fifty entities (for a total of 1,225 sight lines) scattered about the terrain in six random configurations. The observer position $O(p)$ of each entity was offset from the entity's position by a height of 0.1.

For these tests, we converted our DEM terrains to polygonal meshes by forming faces between the elevation values, and used a ray casting approach to conduct LoS queries. This made it possible to use a common LoS algorithm on both the regular and irregular simplified terrains.

Preliminary tests on our six terrains indicate that a feature weight of $w = 0.75$ best preserves LoS query accuracy when using the feature-aware scheme. Hence, in our tests we use a feature weight of 0.75.

Our iterative estimation methods discretized a $5 \times 5 \times 5$ grid of candidate positions surrounding each entity with a maximum of ten allowed iterations. Results are shown only for the projection distance method, which performed the strongest.

For our pseudo-optimized residual vector relocation method, those entities e_j considered to be in the local neighbourhood of an entity e_i were those lying within three simplified faces of e_i. At each iteration, nineteen candidate positions were uniformly discretized along the line segment between $e_i - 2(1-s_i)r_i$ and $e_i + 2(1-s_i)r_i$, where s_i is e_i's current scaling factor and r_i is e_i's residual vector. A maximum iteration count of 5 was allowed.

See Tables 1 and 2 for the average LoS query accuracy of our simplification/point relocation pairs with and without sight line limitations, respectively.

These results indicate that regularity-preserving terrain simplification preserves LoS query accuracy well, with comparable accuracy rates to our results for the irregular simplification methods. The accuracy differences between the regularity-preserving methods, however, appear to be fairly minimal, although subsampling appears to perform the worst of them.

The strong accuracy rates exhibited by the projection distance method (see Table 2) indicate that room for improvement exists over the projection method, however in practice it is difficult to achieve these improvements. Our residuals relocation methods offer slight improvements over projection relocation at high degrees of simplification, with and without sight line limitations.

7.2 Hierarchical Algorithm Results

Our hierarchical algorithm was tested using local least squares reverse Faber subdivision as a simplification method and half projection as a relocation method (in order to favour true positives). Distance thresholds $t_B = 40$ and $t_Q = 320$ were used. Our choice of t_B was motivated by preliminary tests, which indicated that the performance of Bresenham's algorithm falls behind the quad tree algorithm when $d > 40$. A threshold of $t_Q = 320$ allows four levels of the hierarchy to be used with Bresenham's algorithm. As our results in the previous section show, 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.

Tests were conducted on three different large height map terrains (refer to Figure 10) using random distributions of 300 entities (44,850 sight lines), with and without sight line distance limitations. In the limited case, we used a maximum sight line distance of 320. Results for Bresenham's algorithm and the hierarchical approach were compared against the quad tree algorithm, which was considered to produce the correct LoS query results.

Refer to Table 3 for the average accuracy and run-time of each LoS algorithm under these test settings.

(a) Results for simplification to 1.5% of terrain size.

		Optimized			Half
	Residuals	Residuals	Identity	Projection	Projection
1.50%					
Subsampling	72.50%	68.14%	68.16%	73.01%	67.13%
LLS Faber	73.85%	75.43%	72.83%	72.98%	68.97%
GLS Faber	73.05%	73.03%	70.07%	72.83%	68.90%
Feature Aware	72.79%	73.25%	70.36%	73.06%	69.10%
QEC			71.41%	75.57%	71.20%
Greedy Cuts			62.79%	61.80%	57.32%
Greedy Insertion			64.43%	65.31%	63.43%

(b) Results for simplification to 6% of terrain size.

				Half
	Residuals	Identity	Projection	Projection
6.00%				
Subsampling	82.78%	80.20%	82.95%	80.92%
LLS Faber	83.06%	82.26%	82.39%	82.25%
GLS Faber	83.22%	81.99%	83.30%	82.87%
Feature Aware	82.71%	82.12%	83.05%	82.87%
QEC		81.36%	84.62%	82.98%
Greedy Cuts		73.47%	77.68%	75.04%
Greedy Insertion		80.00%	82.85%	83.16%

(c) Results for simplification to 25% of terrain size.

				Half
	Residuals	Identity	Projection	Projection
25.00%				
Subsampling	90.56%	90.74%	90.79%	91.39%
LLS Faber	89.50%	89.56%	90.01%	91.32%
GLS Faber	89.48%	89.42%	90.12%	91.44%
Feature Aware	89.23%	89.36%	89.56%	90.91%
QEC		89.24%	91.09%	92.01%
Greedy Cuts		84.62%	88.67%	89.18%
Greedy Insertion		91.60%	92.78%	94.00%

Table 1 LoS query accuracy results for terrain simplification/point relocation pairs with sight line limitations.

		Optimized			Half	Projection
	Residuals	Residuals	Identity	Projection	Projection	Distance
1.50%						
Subsampling	82.99%	80.54%	80.53%	82.94%	78.51%	95.05%
LLS Faber	83.67%	86.23%	84.38%	82.88%	80.29%	95.82%
GLS Faber	83.16%	83.79%	82.62%	82.77%	79.93%	95.31%
Feature Aware	83.44%	84.30%	82.91%	83.72%	81.15%	95.58%
QEC			81.93%	84.67%	81.72%	94.41%
Greedy Cuts			76.17%	77.31%	72.40%	92.58%
Greedy Insertion			80.50%	79.00%	77.10%	93.68%

Table 2 LoS query accuracy results for terrain simplification/point relocation pairs without sight line limitations at 1.5% of terrain size.

These results show the hierarchical algorithm to be quite fast, with a fairly minimal loss of accuracy. On average the hierarchical approach was 40% faster than the quad tree algorithm, and in our tests never dropped below 97% accuracy.

8 Conclusions

We have examined regularity-preserving terrain simplification in the context of speeding-up line-of-sight queries, and paired it with the study of point relocation as a means for improving LoS query accuracy. Novel techniques, including a feature-aware reverse subdivision scheme and residual vector-based point relocation method, have been presented. Together, these techniques can be used to speed-up LoS queries while keeping accuracy reasonably high.

We have presented a hierarchical line-of-sight algorithm that is both fast and accurate, and can prove useful in applications with real-time speed requirements. As future work, it may be interesting to consider different metrics other than a distance threshold for deciding whether a LoS query will take too long, and additionally the effect of dynamically changing the distance thresholds to meet application needs on the fly.

Although we place entities slightly above the terrain before simplification, the distance is kept constant and small. It would be interesting to consider the impact of our research in situations where entities are allowed to be airborne. Whether or not the same results will hold remains an open question. Additional work may also examine the impact of different metrics for LoS accuracy.

Acknowledgements Thanks go out to C4i Consultants, Inc. for motivating the problem, assisting in research, and providing SRTM terrain data. Thanks go also to the Mitacs Accelerate program for sponsoring the collaboration.

Terrain map images were generated with Global Mapper software. Conceptual figures were created using Inkscape.

(a) Results for algorithms with sight line limitations.

	Bresenham's Algorithm	Hierarchical Algorithm	Min/Max Quad Tree
Camp LeJeune			
Average of 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 of Total Accuracy	98.75%	97.84%	100.00%
Kingston			
Average of 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 of Total Accuracy	99.80%	98.91%	100.00%
Rocky Mountains			
Average of 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 of Total Accuracy	99.81%	99.23%	100.00%

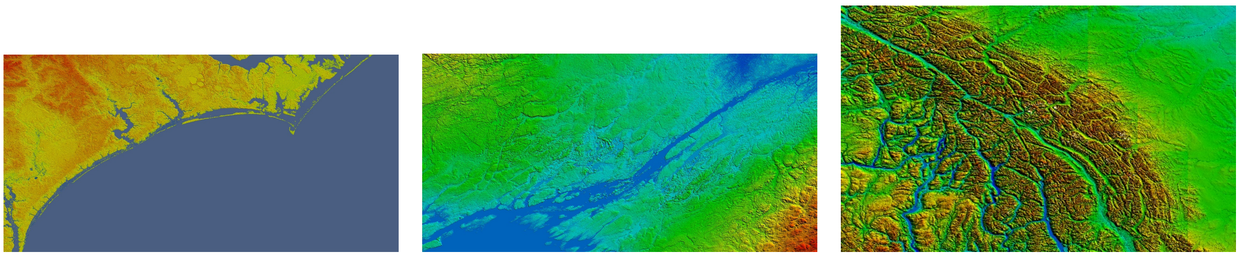
(b) Results for algorithms without sight line limitations.

	Bresenham's Algorithm	Hierarchical Algorithm	Min/Max Quad Tree
Camp LeJeune			
Average of 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 of Total Accuracy	98.98%	97.25%	100.00%
Kingston			
Average of 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 of Total Accuracy	99.71%	98.74%	100.00%
Rocky Mountains			
Average of 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 of Total Accuracy	99.96%	99.76%	100.00%

Table 3 LoS query accuracy and run-time results for line-of-sight algorithms.

References

1. Alderson, T.F., Samavati, F.F.: Reverse subdivision for optimizing visibility tests. In: Proceedings of the International Conference on Computer Graphics Theory and Applications, GRAPP '12, pp. 143–150 (2012)
2. Andrade, M.V.A., Magalhães, S.V.G., Magalhães, M.A., Franklin, W.R., Cutler, B.M.: Efficient viewshed computation on terrain in external memory. Geoinformatica **15**(2), 381–397 (2011)
3. Bartels, R.H., Samavati, F.F.: Reversing subdivision rules: Local linear conditions and observations on inner products. Journal of Computational and Applied Mathematics **119**(1-2), 29–67 (2000)
4. Ben-Moshe, B., Mitchell, J.S.B., Katz, M.J., Nir, Y.: Visibility preserving terrain simplification: An experimental study. In: Proceedings of the 18th Annual Symposium on Computational Geometry, SCG '02. ACM, New York, NY, USA (2002)
5. Bresenham, J.E.: Algorithm for computer control of a digital plotter. IBM Systems Journal **4**(1), 25–30 (1965)
6. Brosz, J., Samavati, F.F., Sousa, M.C.: Terrain synthesis by-example. Communications in Computer and Information Science: Advances in Computer Graphics and Computer Vision **4**, 58–77 (2008)
7. Chaikin, G.M.: An algorithm for high-speed curve generation. Computer Graphics and Image Processing **3**(4), 346–349 (1974)
8. De Floriani, L., Magillo, P.: 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 (1993)
9. Douglas, D.H., Peucker, T.K.: 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 (1973)
10. Duvenhage, B.: Using an implicit min/max kd-tree for doing efficient terrain line of sight calculations. In: Proceedings of the 6th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa, AFRIGRAPH '09. ACM, New York, NY, USA (2009)
11. Dyn, N., Levin, D., Gregory, J.A.: A 4-point interpolatory subdivision scheme for curve design. Computer Aided Geometric Design **4**(4), 257–268 (1987)
12. Franklin, W.R., Ray, C.K.: Higher isn't necessarily better: Visibility algorithms and experiments. In: Proceed-


(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 Bresenham's algorithm.

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

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

Fig. 10 The three test terrain maps for the hierarchical algorithm.

ings of the 6th International Symposium on Spatial Data Handling (1994)

13. Garland, M.: Quadric-based polygonal surface approximation. Ph.D. thesis, Carnegie Mellon University (1999)
14. Garland, M., Heckbert, P.S.: Fast polygonal approximation of terrains and height fields. Tech. Rep. CMU-CS-95-181, Carnegie Mellon University (1995)
15. Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: Proceedings of SIGGRAPH 1997 (1997)
16. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD Conference on Data Engineering, pp. 47–56 (1984)
17. Langetepe, E., Zachmann, G.: Geometric Data Structure for Computer Graphics, chap. 1. A K Peters, Ltd. (2006)
18. Losasso, F., Hoppe, H.: Geometry clipmaps: Terrain rendering using nested regular grids. In: ACM SIGGRAPH 2004 Papers, SIGGRAPH '04. ACM, New York, NY, USA (2004)
19. Ramer, U.: An iterative procedure for the polygonal approximation of plane curves. Computer Graphics and Image Processing **1**(3), 244–256 (1972)
20. Sadeghi, J., Samavati, F.F.: Smooth reverse subdivision. Computers and Graphics **33**(3), 217–225 (2009)
21. Samavati, F.F., Bartels, R.H.: Multiresolution curve and surface representation: Reversing subdivision rules by least-squares data fitting. Computer Graphics Forum **18**(2), 97–120 (1999)
22. Samavati, F.F., Bartels, R.H., Olsen, L.: Local b-spline multiresolution with examples in iris synthesis and volumetric rendering. In: Image Pattern Recognition: Synthesis and Analysis in Biometrics, *Series in Machine Perception and Artificial Intelligence*, vol. 67, pp. 65–102. World Scientific Publishing (2007)
23. Seixas, R.d.B., Mediano, M.R., Gattass, M.: Efficient line-of-sight algorithms for real terrain data. In: Proceedings of the Simpósio de Pesquisa Operacional y Logística da Marinha, SPOLM '99 (1999)
24. Silva, C.T., Mitchell, J.S.B.: Greedy cuts: An advancing front terrain triangulation algorithm. In: Proceedings of GIS 1998 (1998)
25. Silva, C.T., Mitchell, J.S.B., Kaufman, A.E.: Automatic generation of triangular irregular networks using greedy cuts. In: Proceedings of the IEEE Conference on Visualization 1995, pp. 201–208, 453 (1995)
26. Wecker, L., Samavati, F.F., Gavrilova, M.: Contextual void patching for digital elevation models. The Visual Computer **23**(9-11), 881–890 (2007)
27. Wecker, L., Samavati, F.F., Gavrilova, M.: A multiresolution approach to iris synthesis. Computers and Graphics **34**(3), 468–478 (2010)