GRAPHICAL MODELS AND IMAGE PROCESSING
Vol. 59, No. 4, July, pp. 253–264, 1997
ARTICLE NO. IP970434

# NOTE

# Directional Distance Transforms and Height Field Preprocessing for Efficient Ray Tracing

David W. Paglieroni

*Lockheed-Martin Western Development Labs, San Jose, California 95134*

Received June 14, 1996; revised April 15, 1997; accepted April 29, 1997

**It is known that height field ray tracing efficiency can be improved if the empty space above the height field surface is first parameterized in terms of apex heights and opening angles of inverted cones of empty space whose vertical axes are regularly spaced. Once such a parameterization has been performed, rays can be traversed in steps across inverted cones of empty space rather than across successive height field grid cells. As the cone opening angles increase, ray tracing efficiency tends to improve because steps along rays across the inverted cones get longer. Circular horizontal cross-sections of an inverted cone can be divided into contiguous nonoverlapping sectors. Given that the inverted cones can contain nothing but empty space, the maximum possible opening angle within any such sector may significantly exceed the opening angle of the inverted cone. It is shown that ray tracing efficiency can be significantly improved by replacing the inverted cones of empty space with cones that have narrow sectors. It is also known that the parameters of the inverted cones can be derived from distance transforms (DTs) of successive horizontal cross-sections of the height field. Each cross-section can be represented as a 2D binary array, whose DT gives the distance from each element to the nearest element of value 1. DTs can be directionalized by requiring the element of value 1 closest to a given element to lie within a sector emanating from that given element. The parameters of inverted cones within specific sectors can be derived from such directional DTs. An efficient new algorithm for generating directional DTs is introduced.** 

## 1. INTRODUCTION

Ray tracing is an important computer graphics and geographical information system (GIS) tool. It is used to determine the point where a ray through object space first intersects the surface of an object. When one is concerned only with terrain, object space is often modeled by a height field (i.e., a 2D array of evenly spaced elevation samples). Applications of height field ray tracing include geopositioning, intervisibility analysis, and photorealistic terrain rendering. Geopositioning is the process of determining the object space coordinates of a point imaged by a pixel in an aerial image. Intervisibility analysis is used to determine whether points in object space are visible to each other. Photorealistic terrain rendering is the process of generating views of real imagery draped over terrain. Intervisiblity and photorealistic terrain rendering applications can require a large number of rays to be traced each second. Algorithms for improving height field ray tracing efficiency are thus important.

For example, one intervisibility problem is to determine the minimal set of observation points from which one can view all points on a height field surface (see, e.g., [5]). The line-of-sight network problem is similar. Given a set of stations distributed in known fashion over terrain with a known elevation model, the problem is to construct a small network of relay towers in such a way as to link the stations in a visibility-connected network (see e.g., [6]). Alternatively, for a given viewpoint and terrain model, one may be interested in determining the viewshed, i.e., a representation of which elements of the surface are visible from that viewpoint and which are not (see, e.g., [7]).

In addition, an aerial image is typically rendered photorealistically by mapping its pixel values onto a rendered terrain surface. In one approach, rays or lines-of-sight that pass through pixels in an image being synthesized are traced to determine where they first intersect terrain. The object space coordinates of the points of intersection are then projected back onto the aerial image being rendered. Aerial image pixel values in the vicinity of the projection point are then combined to determine the value to assign to the corresponding pixel in the picture being photorealistically synthesized.

There are several classes of methods for height field ray tracing. With *incremental ray tracing*, rays are traced in steps across successive height field grid cells (see, e.g., [10] and Fig. 1a). This method is computationally expensive



1077-3169/97 $25.00

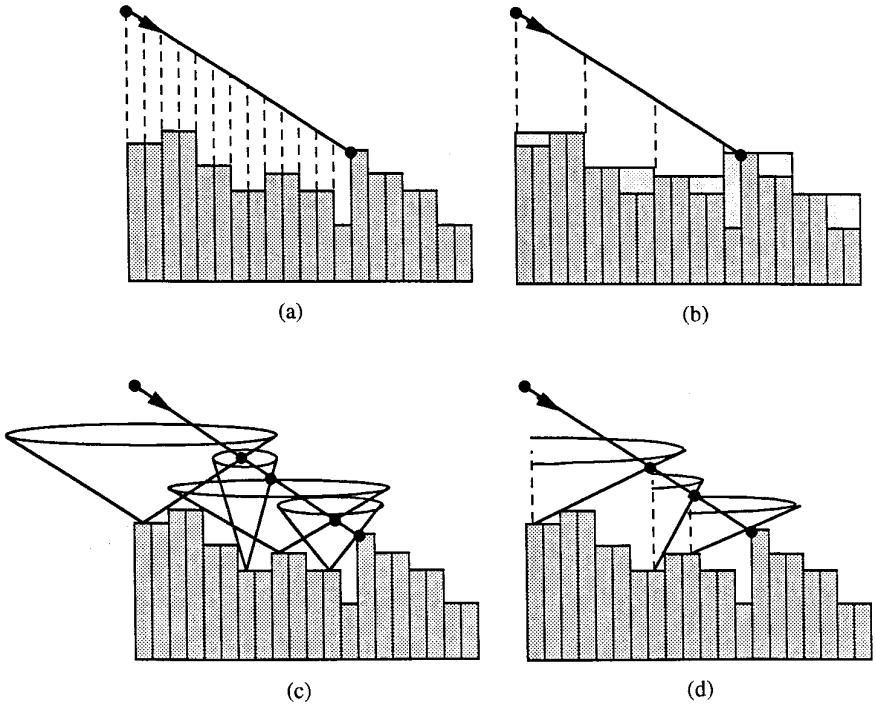


**FIG. 1.** Graphical illustration of ray tracing over height field cells (dark shading): (a) incremental, 13 steps; (b) hierarchical (4:1 resolution), 4 steps; (c) linear parametric, 4 steps; (d) linear directional parametric, 3 steps.

for oblique rays that traverse long distances over the height field surface before intersecting it. One way to improve ray tracing efficiency is to make use of height field preprocessing. One approach to height field preprocessing is to represent the height field by rectangular bounding volumes. For example, a resolution pyramid can be generated from the height field. Then with *hierarchical ray tracing*, rays can be traced incrementally, but always at the lowest possible resolution in the height field resolution pyramid (see, e.g., [3] and Fig. 1b). Another approach to height field preprocessing is to parameterize the empty space above the height field surface. For example, the empty space above the height field surface can be parameterized in terms of apex heights and opening angles of inverted cones of empty space whose vertical axes are regularly spaced. Then with *linear parametric ray tracing*, rays can be traversed in steps across inverted cones of empty space (see [12, 13] and Fig. 1c). Suppose there is one inverted cone associated with each height field cell. The first step along the ray is taken to the point where the ray exits the inverted cone associated with the cell that contains the starting point on the ray. As shown in Fig. 1c, the next step along the ray is taken to the point where the ray exits the inverted cone associated with the cell that contains the point previously stepped to, and so on.

As is evident from Fig. 1c, the efficiency of linear parametric ray tracing tends to improve as the inverted cones of empty space get wider because the traversal steps along the rays tend to get longer. Circular horizontal cross-sections of an inverted cone can be divided into contiguous nonoverlapping sectors. An inverted cone of empty space can be made wider by noticing that it may be much less obstructed by the height field surface within some sectors than within others. Since the width of an inverted cone is the same as the width of the cone within the most obstructed sector, the cone opening angles may be much larger within the other sectors. Use of inverted cones with narrow sectors can thus further improve ray tracing efficiency (see Fig. 1d). Linear parametric height field ray tracing methods are reviewed in Section 2. A rigorous analysis of the impact of sector width on ray tracing efficiency is beyond the scope of this paper. However, Section 2.4 describes a simple experiment that was conducted to show that ray tracing efficiency can be significantly improved by making use of inverted cones of empty space with narrow sectors. The results of this experiment indicate that it makes sense to study how to efficiently generate inverted cones of empty space within specified sectors from height fields.

Although this paper focuses on how inverted cones of empty space with narrow sectors apply to height field ray tracing, they also apply to the computation of shadows. Shadows are often cast on height field surfaces by making use of horizon tables [2, 9]. A horizon table is a 2D array of horizon angles associated with height field samples. The horizon angle corresponding to a height field sample is the

angle, measured from vertical, of the line-of-sight from that point to the horizon as seen from that point. The horizon seen from a point depends on the direction of the line-of-sight projected onto the height field grid. Consider the ray from a height field sample through the center of the Sun. The two angles associated with this ray are its angle measured from vertical (i.e., the sun angle) and the angle $\theta$ of its projection onto the height field grid measured from east. A height field sample falls under a shadow if the sun angle is greater than its horizon angle in direction $\theta$. For each height field sample, horizon angles can be precomputed for the 8 compass directions $\theta = 0°, 45°, 90°, \ldots, 315°$ by computing angles of lines-of-sight through all samples to the east, northeast, north, northwest, west, southwest, south and southeast, respectively. Then, horizon angles for arbitrary directions can be estimated by interpolating horizon angles corresponding to the pair of closest directions drawn from the eight compass directions [9]. Better estimates can be obtained by using more compass directions. Now the opening angle of the widest possible inverted cone of empty space within an infinitesimally narrow sector is the horizon angle corresponding to the apex in the direction of that sector. However, the opening angle of an inverted cone within a sector of finite width generally underestimates the horizon angles corresponding to the apex in directions spanned by that sector (and thus overestimates the shadows). Better estimates can be obtained by using sectors that are more narrow. Note that this approach does not make use of interpolation.

A linear *parameter plane transform* (*PPT*) of a height field is a 2D array of parameters of inverted cones of empty space associated with height field cells. An efficient algorithm for generating PPTs from successive height field horizontal cross-sections was introduced in [12, 13]. Each cross-section is treated as a bit map (i.e., a 2D binary array of 0s and 1s). A *distance transform* (DT) of a bit map is a 2D array of distances from each bit map element to the nearest bit map element of value 1. One DT is generated for each cross-section. A *directional distance transform* is a DT in which the bit map element of value 1 closest to a given bit map element is required to lie within a sector emanating from that given element. The PPT algorithm can be extended to include parameters of inverted cones within specific sectors by computing directional DTs, as opposed to ordinary DTs, for each horizontal height field cross-section. The focus of this paper is on the introduction of an efficient new algorithm for generating directional DTs of bit maps (see Section 3).

## 2. REVIEW OF PARAMETRIC HEIGHT FIELD RAY TRACING

A height field can be preprocessed by parameterizing the empty space above the height field surface. As described in Section 2.1, linear parameter plane transforms (PPTs) are parameterizations of the empty space above a height field surface in terms of inverted cones of empty space whose vertical axes are regularly spaced. As discussed in Section 2.2, the linear parametric height field ray tracing algorithm uses linear PPTs to improve efficiency. Section 2.3 describes how to use distance transforms (DTs) to efficiently generate PPTs. Section 2.4 shows how ray tracing efficiency can improve by using inverted cones of empty space with narrow sectors.

### 2.1. *Linear Parameter Plane Transforms*

A linear PPT of a height field physically represents a 2D array of inverted cones of empty space. There is one inverted cone associated with each height field cell. Its vertical axis passes through the center of that cell and its apex must lie on or above the height field surface. A height field surface is generated from a height field by interpolating the height field samples in some way. For example, if bilinear interpolation is used, then the resulting height field surface would not be discontinuous like it is in Fig. 1. A linear PPT is said to be directional when the inverted cones have opening angles that are allowed to vary between sectors. PPTs based on cones are said to be linear because the parameters of a cone are those of a line through the apex that sweeps out the cone surface when spun about the vertical axis through its apex.

The opening angle of an inverted cone will increase as its apex is raised. In fact, within certain sectors, it is possible for the opening angle to lie between 90° and 180° if the apex is raised high enough. However, the amount of empty space immediately above the height field surface not covered by the inverted cones will also increase. It turns out that these two phenomena have opposite effects on height field ray tracing efficiency. By choosing each apex to be close to the height field surface, the PPT algorithm introduced in [12, 13] places greater emphasis on the importance of minimizing the amount of empty space left uncovered by the inverted cones.

A cone can be represented by two parameters, namely cone apex height and opening angle (or alternately, the slope of the line that sweeps out the cone surface). In a linear PPT, one pair of parameters is assigned to each cone in a 2D array of cones. This 2D array of cone parameters can be split into two *parameter planes*. One parameter plane contains values of the apex height parameter and the other contains values of the opening angle parameter. In a linear directional PPT, $N + 1$ parameters are assigned to each cone in a 2D array of cones with $N$ sectors. The first parameter is apex height and the remaining $N$ parameters are the cone opening angles assigned to each of the $N$ sectors.

The problem with directional PPTs is that the computa-

tional cost of generating them and the amount of memory required to store them is directly proportional to the number of sectors, which is large if all sectors are narrow. Two things can be done to manage this problem. First, the amount of memory required to store a directional PPT can be reduced by quantizing cone opening angles to a relatively small number of bits. For example, if height field samples and apex heights are each allocated 2 bytes, but only 1 byte is allocated to opening angles, then for large $N$, the directional PPT requires only about $N/2$ (as opposed to $N$) times more memory than the height field. Note that if only 8 bits are allocated to cone opening angles, the quantization will be $180°/256 \approx 0.7°$, which is still quite fine. The quantization in cone opening angle is properly defined as a fraction of 180° (as opposed to 90°) because, as discussed earlier, the opening angle of an inverted cone within a specific sector can lie between 0° and 180°.

Second, the computational cost and memory requirements associated with a directional PPT can be significantly reduced by decreasing the resolution of the PPT, i.e., by generating the PPT of a reduced resolution height field. In particular, it is not necessary to assign one inverted cone to every cell in a full resolution height field. Instead, one may choose to assign one inverted cone to every cell in a reduced resolution height field. When generating a reduced resolution height field, assume that the height of a reduced resolution height field cell is the height of the tallest full resolution height field cell that it contains, and the height of a full resolution height field cell is the height of the height field sample at its tallest corner. For large $N$, the computational cost and memory requirements associated with generating and storing a directional PPT decrease in direct proportion to the square of the reduction in resolution. For example, if height field samples and apex heights are each allocated 2 bytes, 1 byte is allocated to opening angles, PPT resolution is decreased by a factor of 3 and $N = 24$, then it requires only $13/9 \approx 1.4$ times more memory to store the directional PPT than is required to store the original height field. Furthermore, the computational cost associated with generating the reduced resolution directional PPT will be about 9 times less than for the full resolution directional PPT.

### 2.2. *Ray Tracing Algorithm*

As shown in Figs. 1c and 1d, with linear parametric ray tracing, rays are traversed in steps across inverted cones of empty space. In the directional case, the sector that the ray belongs to must first be determined. Then within that sector, ray tracing steps are always taken to the point where the ray exits the inverted cone assigned to the height field cell that contains the point stepped from. The resolution of the cell that contains this point is dictated by the resolution of the PPT.

Sometimes, the point to step from on the ray lies beneath (rather than inside) the inverted cone associated with the height field cell containing that point. This can occur because the inverted cones do not cover all of the empty space immediately above the height field surface. In this case, ray tracing proceeds incrementally until either the point where the ray first intersects the height field surface has been found or the point incrementally stepped to on the ray again lies within the inverted cone assigned to the height field cell that contains it. The same height field cell can also sometimes contain both the point stepped from on the ray and the point where the ray exits the inverted cone associated with the cell containing the point stepped from. In this case, an incremental ray tracing step is taken instead. Mathematical details of the parametric ray tracing algorithm can be found in [12, 13] and will not be repeated here.

### 2.3. *Using Distance Transforms to Generate Parameter Plane Transforms*

A reduced resolution PPT is defined as the PPT of a reduced resolution height field. The resolution of the PPT is specified by a positive integer. A value of ''one'' is used for full resolution PPTs, a value of ''two'' is used for half resolution PPTs, etc. The resolution of a height field can be reduced as described in Section 2.1.

At any resolution, height field PPTs can be efficiently generated from successive height field horizontal cross-sections. The height field cross-sections are represented as bit maps in which elements of value 0 correspond to cells where the cross-sections cut through empty space and elements of value 1 correspond to cells where the cross-sections cut through the height field surface.

In practice, the horizontal cross-sections are often taken at evenly spaced heights. The spacing between successive horizontal cross-sections is thus dictated by the range of elevations within the height field and the desired number of cross-sections. In practice, the height of the apex of an inverted cone is then often given by the height of the lowest cross-section that does not lie below the top of the height field cell that contains the apex (see [12, 13]). As described later in this section, the opening angle of an inverted cone is calculated from its apex height and the DTs of the cross-sections that do not lie beneath the apex. The algorithm for computing the opening angle of an inverted cone within a specific sector is the same, except directional DTs (as opposed to ordinary DTs) are used.

The distance transform was introduced to image processing in [15] as a binary image transform with applications to image analysis. The DT of a bit map is the 2D array of pixel distances from each bit map pixel to the nearest pixel of value 1 (i.e., feature pixel). The *nearest feature transform* (*FT*) of a bit map is a 2D array of coordi-

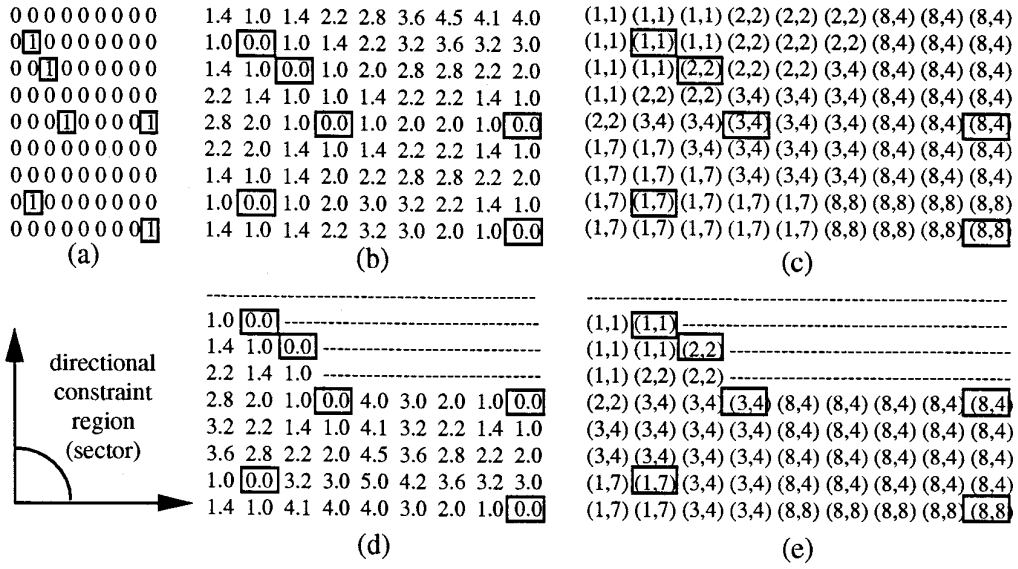


**FIG. 2.** (a) $9 \times 9$ bit map. (b, c) Unconstrained Euclidean DT and FT. (d, e) Directional Euclidean DT and FT for sectors that span the upper right quadrant.

nates of those nearest feature pixels. DTs are unique but FTs may not be. DTs can be derived from FTs but not vice versa.

For *unconstrained* DTs, nearest feature pixels are selected from among all bit map pixels. For *constrained* DTs, nearest feature pixels are selected from among pixels that lie within a constraint region. *Spatially invariant constraint regions* are fixed and apply to all bit map pixels. *Spatially variant constraint regions* vary from pixel to pixel. *Directional constraints* are spatially variant constraints in which the constraint region associated with a bit map pixel is a sector that emanates from that pixel. A directional DT is a DT with spatially variant directional constraints (see Fig. 2).

Consider how to determine the largest possible opening angle of an inverted cone of empty space centered above a specific height field cell. Assume that the apex height has been specified and that one has access to the Euclidean DT of each height field horizontal cross-section. The Euclidean DT of a bit map can be interpreted as a 2D array of circle radii values in which each circle contains only elements of value 0 but is as large as possible. For a specific height field cell, the DT values associated with successive height field horizontal cross-sections thus correspond to radii of successive horizontal circles that contain nothing but empty space (see Fig. 3). The circle that represents the horizontal cross-section of the narrowest cone (the shaded circle in Fig. 3) will correspond to the widest inverted cone that contains nothing but empty space. If there is a one-to-one correspondence between points on the horizontal plane and points directly above or below them on the height field surface (as is the case for typical models of terrain elevation), then the radii of these circles will be a nondecreasing function of height. One can thus guarantee that the interval between two successive cross-sections contains nothing but empty space by attributing circles identical to the one in the plane of the lower of the two cross-sections to *all* horizontal planes between the two cross-sections.

It is important to realize that cone opening angles can be updated iteratively each time the DT of a new height field horizontal cross-section is generated. It is therefore *not* necessary to store the DTs of all cross-sections in memory at the same time in order to compute a linear PPT. To this end, it is convenient to process the cross-sections in order of increasing height. A linear PPT algorithm based on the principles outlined in this section that processes only one cross-section in memory at a time is formally stated in [12, 13]. The mathematical details will not be repeated here.

### 2.4. *Experiment*

For specific height field cells, the radii associated with circles in the planes of successive height field horizontal cross-sections (as depicted in Fig. 3) will tend to be larger within specific sectors. The cone opening angles will thus tend to be wider within specific sectors. This can give rise to significant improvements in height field ray tracing efficiency. The results of a simple experiment conducted to show that ray tracing efficiency can be significantly improved by using inverted cones of empty space with narrow sectors are documented below. The intent is to lend credibility to the claim that directional distance transforms are

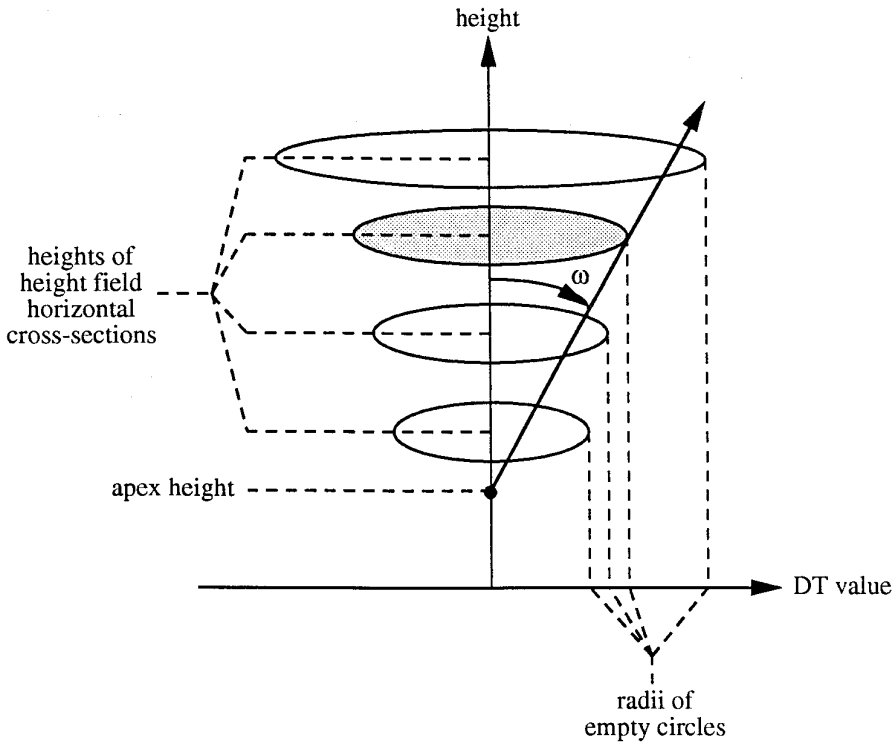


**FIG. 3.** Graphical illustration of how Euclidean DTs of height field horizontal cross-sections are used to determine opening angles $\omega$ of inverted cones of empty space in a linear PPT.

important for efficient height field ray tracing so as to justify the need to develop efficient directional DT algorithms.

Figure 4 depicts a type of height field available from the USGS called a digital elevation model or DEM. This DEM corresponds to a 7.5′ quadrangle of Washoe Lake Nevada. Gray-scale intensity increases with terrain elevation. This height field has 450 rows and 345 columns of square cells with 30 m widths. Its easterly extent relative to the central meridian of its universal transverse Mercator (UTM) zone ranges from 252,360 to 262,680 m. Its northerly extent measured from the equator ranges from 4,348,320 to 4,631,790 m. The terrain elevation within this height field ranges from 1417 to 2687 m. The Washoe lake area is characterized by mountainous terrain that partially surrounds a lake basin.

**FIG. 4.** USGS digital elevation model corresponding to the 7.5 min quadrangle of Washoe Lake, Nevada (white represents higher elevation).

Consider a horizontal ray at a height of 1650 m pointing 23° clockwise from north with a starting point at an easterly location of 254,550 m and a northerly location of 4,348,890 m. The endpoint of this horizontal ray thus lies towards the lower left corner of the height field and generally points north across the basin. It requires 500 incremental ray tracing steps but only 73 hierarchical ray tracing steps to determine where this ray first intersects the DEM surface. Now suppose the height field is sliced into 80 evenly spaced horizontal cross-sections. Then 63 linear parametric ray tracing steps are required if inverted cones are used but only 16 linear parametric ray tracing steps are required if inverted cones with 24 contiguous non-overlapping 15° sectors are used. The linear directional parametric ray tracing algorithm thus requires more than 30 times fewer ray tracing steps than the incremental ray tracing algorithm, more than 4 times fewer ray tracing steps than the hierarchical ray tracing algorithm, and nearly 4 times fewer ray tracing steps than the linear nondirectional parametric ray tracing algorithm.

## 3. DIRECTIONAL DISTANCE TRANSFORM ALGORITHM

DTs can be based on any valid distance measure $D$, where $D(x_1 - x_0, y_1 - y_0)$ represents the distance from $(x_0, y_0)$ to $(x_1, y_1)$. Euclidean distance ($D(x, y) = (x^2 + y^2)^{1/2}$) is the most popular. Chamfer distance ($D(x, y) = w_0|x| + w_1|y|$ for $|x| > |y|$ and $w_1|x| + w_0|y|$ for $|x| \leq |y|$) can be used to approximate Euclidean distance. Typical examples of chamfer distance measures include city-block ($(w_0, w_1) = (1, 1)$), chessboard ($(w_0, w_1) = (1, 0)$), and chamfer-Euclidean ($(w_0, w_1) = (1, \sqrt{2} - 1)$) distance measures. Chamfer DTs appear frequently in the DT literature because they are easier to compute than Euclidean DTs.

Section 3.1 reviews some of the more popular classes of DT algorithms. One class contains rasterized line traversal algorithms. These algorithms apply to a broad class of distance measures that includes both Euclidean and chamfer distance. Section 3.2 shows how to extend rasterized line traversal algorithms to efficiently compute directional distance transforms (refer to Fig. 2).

### 3.1. *Some Classes of DT Algorithms*

There are several classes of practical DT algorithms. *Raster propagation algorithms* propagate distance along bit map rows and columns. They have both serial and massively parallel implementations (e.g., [1, 4, 15, 17]), can be extended to compute FTs (e.g., [8]), and can accommodate spatially invariant constraints (e.g., [14]). The Euclidean DT of any bit map with $N$ pixels can be generated in $O(N)$ operations by serial raster propagation and the number of operations is fixed, but the DT will generally contain errors. *Contour propagation algorithms* iteratively propagate distance out from bit map pixels of value 1 along boundaries (contours) between regions of 0s and 1s (e.g., [16]). The DT of a bit map with $N$ pixels can typically be generated in $O(N)$ operations by contour propagation, but the exact number of operations can vary. Contour propagation can be extended to compute FTs and can accommodate spatially invariant constraints.

*Rasterized line traversal algorithms* are based on traversal of *rasterized* (or *scan-converted*) *lines*, i.e., sequences of connected pixels that some ideal line which spans the raster passes through. The pointing direction attributed to a rasterized line is specified by its orientation angle $\theta$. Assume for the moment that $\theta$ is measured clockwise from a horizontal ray pointing left to a ray pointing along the direction of the rasterized line (the reason for this particular choice will be made clear in Section 3.2). A *rasterized line segment* will be defined as a segment of a rasterized line that emanates from the same pixel that the rasterized line emanates from. Rasterized lines contain exactly one array element per column for $|\tan\theta| \leq 1$ and exactly one array element per row for $|\tan\theta| > 1$ (see Fig. 5). Rasterized line traversal algorithms generate bit map DTs and FTs in two stages (see Fig. 6). The input to the first stage is the bit map itself and the output is a 2D intermediate array that has the same number of rows and columns as the bit map does. This intermediate array is input to a second stage which then produces the DT and FT as output. Each stage of processing represents one pass through the input array. An intermediate array is generated by traversing a complete set of rasterized lines at orientation $\theta_1$ in a first pass through the bit map. The DT and FT are then generated by traversing a complete set of rasterized lines at orientation $\theta_2 \neq \theta_1$ in a second pass through the intermediate array. Rasterized line traversal algorithms differ from raster propagation algorithms in that the result of processing a rasterized line does *not* depend on the results of processing previous rasterized lines. They can thus be implemented serially or in parallel and the rasterized lines can be processed in any order. A rasterized line traversal algorithm for computing unconstrained DTs and FTs is given in [11]. Section 3.2 introduces a rasterized line traversal algorithm for computing directional DTs and FTs.

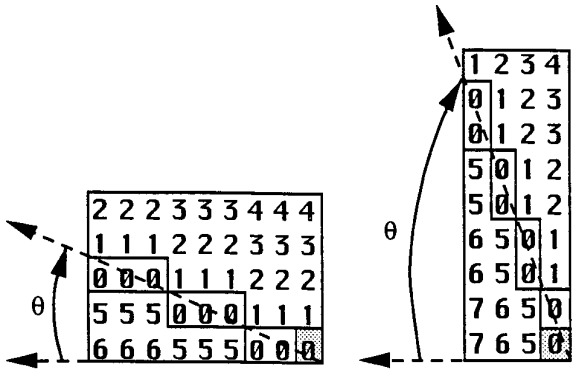


**FIG. 5.** Complete sets of parallel rasterized lines through two arrays for $|\tan\theta| < 1$ and $|\tan\theta| > 1$. The rasterized lines labeled with 0s emanate from the shaded pixels.

### 3.2. *Generating Directional DTs by Rasterized Line Traversal*

The directional DT algorithm described below applies when $0° < |\theta_2 - \theta_1| \leq 90°$. Each element of the input bit

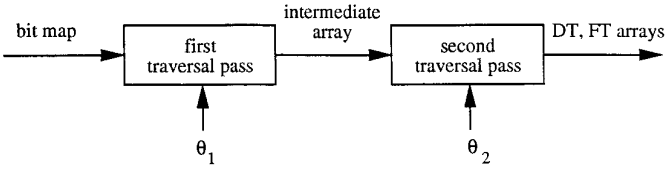


**FIG. 6.** Overview of the rasterized line traversal approach for generating distance transforms of bit maps.

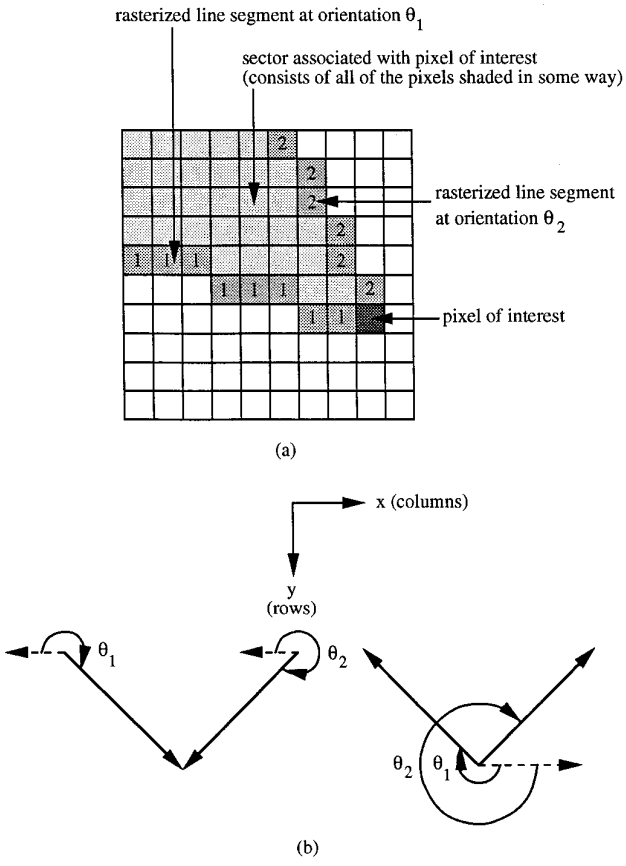


**FIG. 7.** (a) The sector bounded by rasterized line segments at orientations $\theta_1$ and $\theta_2$ that converge on a pixel of interest. (b) The relationship between the orientation of a rasterized line and the angular range of a sector in a digital image.

map represents the unique intersection between exactly one rasterized line at orientation $\theta_1$ and one rasterized line at orientation $\theta_2$. Furthermore, as illustrated in Fig. 7a, the rasterized line segments at orientations $\theta_1$ and $\theta_2$ that converge on a bit map pixel bound a sector that emanates from that pixel. Recall from Section 3.1 that the orientation of a rasterized line is measured clockwise from a horizontal ray pointing left to a ray pointing along the direction of the rasterized line. This measure of orientation is illustrated in Fig. 7b. To see why this measure was chosen, consider the fact that for digital images, the columns axis points to the right and the rows axis points down. In such a coordinate system, it is customary to measure the angles of rays that bound a sector clockwise from the columns axis. Figure 7b illustrates the fact that the orientations of two convergent rasterized line segments will match the angular range of the resulting sector only if the definition of rasterized line orientation given in Section 3.1 is used.

### 3.2.1. *First Traversal Pass*

In the first traversal pass, each rasterized line that belongs to a complete set of rasterized lines passing through the input bit map at orientation $\theta_1$ is independently traversed and processed in the same way. For each pixel on a rasterized line at orientation $\theta_1$, the objective of the first traversal pass is to determine the nearest feature pixel that lies on the portion of the rasterized line that has been or is being visited. The distance between two pixels along a rasterized line is taken to be one plus the number of pixels between them along that line. Distances of infinity are attributed to pixels on a rasterized line for which no such feature pixels exist. The output of the first traversal pass is thus an intermediate array whose elements contain coordinates of feature pixels together with distances along the rasterized line to those feature pixels.

Let $\{b_{1,k}, k = 1, \ldots, n\}$ be the input sequence of bit map pixel values along a rasterized line $L_1$ at orientation $\theta_1$ through the bit map. The subscript "1" indicates that the sequence elements refer to pixels along rasterized lines at orientation $\theta_1$. Let $\{\mathbf{q}_{1,k}, k = 1, \ldots, n\}$ be the output sequence of (column, row) coordinate two-tuples associated with the feature pixel closest to pixel $k$ among pixel $k$ and its predecessors on $L_1$. Let $\{L_{1,k}, k = 1, \ldots, n\}$ be the output sequence of distance values (in pixels) along $L_1$ from pixel $k$ on $L_1$ to $\mathbf{q}_{1,k}$. Also, consider the initialization variables $\mathbf{q}_{1,0}$ and $L_{1,0}$. The algorithm for first pass traversal along a rasterized line at orientation $\theta_1$ can now be stated as follows:

ALGORITHM FOR FIRST PASS TRAVERSAL ALONG A RASTERIZED LINE.

$\mathbf{q}_{1,0} \leftarrow (\infty, \infty)$, $L_{1,0} \leftarrow \infty$
**for** $k = 1, \ldots, n$
  **if** $b_{1,k} = 1$ **then**
    $\mathbf{q}_{1,k} \leftarrow$ (column,row) coordinates of $b_{1,k}$ in bit map,
    $L_{1,k} \leftarrow 0$
  **else**
    $\mathbf{q}_{1,k} \leftarrow \mathbf{q}_{1,k-1}$, $L_{1,k} \leftarrow L_{1,k-1} + 1$

Figure 8 depicts a first pass traversal along a rasterized line at orientation $\theta_1 = 225°$. These rasterized lines emanate from the upper leftmost pixel in the sequence. The dashes represent values of infinity.

### 3.2.2. *Second Traversal Pass*

Consider a bit map pixel $\mathbf{p}$ on a rasterized line $L_2$ at orientation $\theta_2$ (see Fig. 9). The pixels on $L_2$ that precede $\mathbf{p}$ occur at unique points of intersection between $L_2$ and the parallel rasterized line segments $L_1$ at orientation $\theta_1$ that terminate on $L_2$. The coordinates of the pixel of value 1 closest to $L_2$ on such a rasterized line segment $L_1$ are stored in the element of the intermediate array that corresponds to the pixel where $L_1$ intersects $L_2$. If $D(x, y)$ is nondecreasing in both $|x|$ and $|y|$, then the pixel of value 1 closest to $\mathbf{p}$ that lies in the region bounded by $L_2$ and

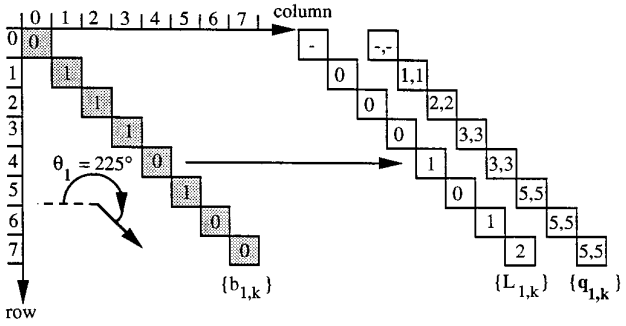


**FIG. 8.** An example of first pass traversal along a rasterized line (shaded) at orientation $\theta_1$.

the rasterized line segment at orientation $\theta_1$ that terminates on **p** is one of the pixels whose coordinates are stored in the intermediate array on the rasterized line segment at orientation $\theta_2$ that terminates on **p**. Directional DTs are thus generated by traversing rasterized lines at exactly two orientations.

In the second traversal pass, each rasterized line that belongs to the complete set of rasterized lines passing through the intermediate array at orientation $\theta_2$ is independently traversed and processed in the same way. For each pixel on a rasterized line at orientation $\theta_2$, the objective of the second traversal pass is to determine the nearest feature pixel that lies in the region bounded by the unique pair of rasterized line segments at orientations $\theta_1$ and $\theta_2$ that converge on that pixel. Distances of infinity are attributed to pixels for which no such feature pixels exist. The output of the second traversal pass is thus, by definition, a directional FT of the bit map. The sector emanating from a given pixel on the bit map is bounded by the unique pair of rasterized line segments at orientations $\theta_1$ and $\theta_2$ that converge on that pixel.

Let $\{\mathbf{s}_{2,k}, k = 1, \ldots, n\}$ be the (column, row) coordinates of the sequence of pixels that lie along a rasterized line $L_2$ at orientation $\theta_2$ through the bit map. The subscript 2 indicates that the sequence elements refer to pixels along

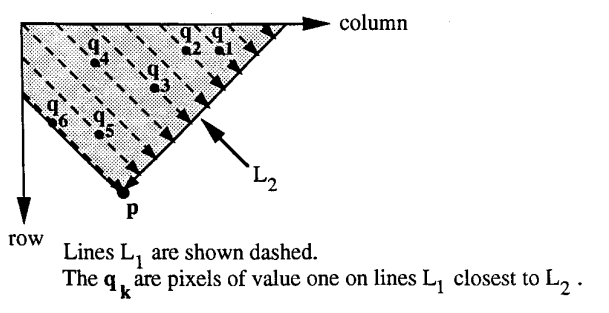


**FIG. 9.** The pixel of value 1 closest to **p** in the shaded directional constraint region is one of the pixels $\mathbf{q}_k$.

rasterized lines at orientation $\theta_2$. Let $\{\mathbf{q}_{2,k}, k = 1, \ldots, n\}$ be the input sequence of (column, row) coordinate two-tuples of feature pixels stored in the intermediate array along $L_2$. Let $\{d_{2,k}, k = 1, \ldots, n\}$ be the output sequence of directional DT values along $L_2$ corresponding to traversal passes along rasterized lines at orientations $\theta_1$ and $\theta_2$. Finally, let $\{\mathbf{p}_{2,k}, k = 1, \ldots, n\}$ be an output directional FT sequence of (column, row) coordinate two-tuples of the nearest feature pixels associated with the sequence $\{d_{2,k}, k = 1, \ldots, n\}$ of directional DT values. The algorithm for second pass traversal along a rasterized line at orientation $\theta_2$ can now be stated as follows:

ALGORITHM FOR SECOND PASS TRAVERSAL ALONG A RASTERIZED LINE.

**for** $k = 1, \ldots, n$
  $\mathbf{p}_{2,k} \leftarrow$ the $\mathbf{q}_{2,i}$ closest to $\mathbf{s}_{2,k}$ for $i = 1, \ldots, k$
  $d_{2,k} \leftarrow$ distance from $\mathbf{s}_{2,k}$ to $\mathbf{p}_{2,k}$

Figure 10 depicts a second pass traversal along a rasterized line at orientation $\theta_2 = 315°$. These rasterized lines emanate from the upper rightmost pixel in the sequence. The $d_{2,k}$ represent values of squared Euclidean distance and the dashes represent values of infinity.

In the algorithm for second pass traversal, the brute-force approach to computing $\mathbf{p}_{2,k}$ is to compute $k$ distances between pixels for each of $k = 1, \ldots, n$. By this approach, it requires $O(n^2)$ operations to process rasterized lines that contain $n$ pixels. The computational cost can often be reduced to $O(n)$ operations by embedding special computational techniques into the algorithm for second pass traversal. The idea is to reduce the number of pixels that need to be considered. There are two independent techniques for doing this. Both are valid if $D(x, y)$ is nondecreasing in $|x|$ and $|y|$. As shown in Section 3.2.2.1, the first technique is based on a rule that applies to bit map pixels on rasterized lines from the first pass. As shown in Section 3.2.2.2, the second technique is based on a rule that applies to elements of the intermediate array on rasterized lines from the second pass.

3.2.2.1. *The First Pass Rule.* Consider the set of $k$ rasterized line segments at orientation $\theta_1$ with endpoints $\mathbf{s}_{2,i} = 1, \ldots, k$ that lie along a rasterized line $L_2$ at orientation $\theta_2$ through the input bit map. $\mathbf{q}_{2,i}$ is the pixel of value 1 closest to $\mathbf{s}_{2,k}$ on the rasterized line segment at orientation $\theta_1$ with endpoint $\mathbf{s}_{2,i}$. Let $L_{2,i}$ be the distance from $\mathbf{q}_{2,i}$ to $\mathbf{s}_{2,i}$ along the rasterized line at orientation $\theta_1$ that contains $\mathbf{s}_{2,i}$. Note that the data contained in the intermediate array on the rasterized line segment along $L_2$ that terminates on $\mathbf{s}_{2,k}$ consists of $\mathbf{q}_{2,i}$ and $L_{2,i}$ for $i = 1, \ldots, k$. In the brute force approach, $\mathbf{p}_{2,k}$ is found by computing the distance between $\mathbf{s}_{2,k}$ and $q_{2,i}$ for $i = 1, \ldots, k$.

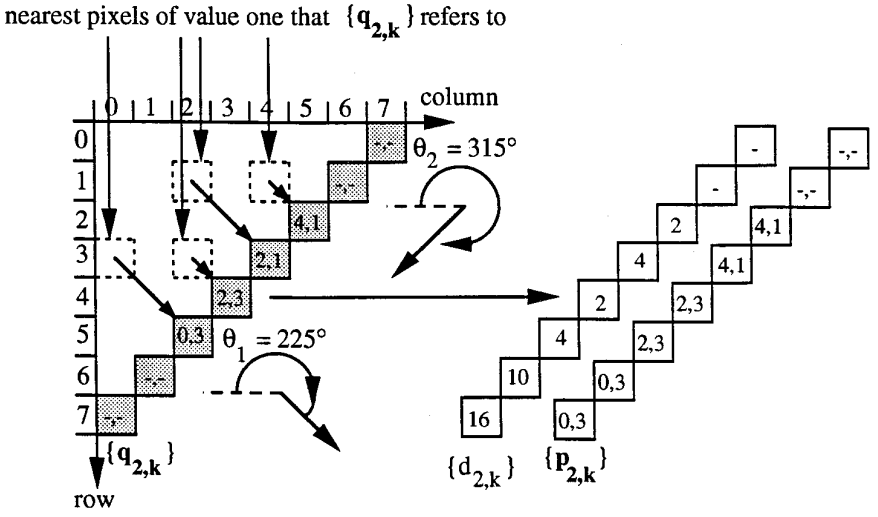


**FIG. 10.** An example of second pass traversal along a rasterized line (shaded) at orientation $\theta_2$. The $d_{2,k}$ represent values of squared Euclidean distance.

The *first pass rule* applies to bit map pixels $\mathbf{q}_{2,i}$ $i = 1, \ldots, k$ that lie on rasterized lines from the first pass, and it can be stated as follows:

FIRST PASS RULE. If $D(x, y)$ is nondecreasing in both $|x|$ and $|y|$, then $\mathbf{q}_{2,i}$ does not need to be considered in determining the directional FT of $\mathbf{s}_{2,k}$ if there exists an integer $j$ such that $i < j \leq k$ and $\mathbf{q}_{2,j}$ is at least as close to $L_2$ as $\mathbf{q}_{2,i}$ is.

If the first pass rule is used, then $\mathbf{p}_{2,k}$ can be found by computing the distance between $\mathbf{s}_{2,k}$ and points $\mathbf{q}'_{2,i}$, $i = 1, \ldots, k'$, where $\{\mathbf{q}'_{2,i}, i = 1, \ldots, k'\}$ is a subset of $\{\mathbf{q}_{2,i}, i = 1, \ldots, k\}$ $(k' \leq k)$.

Let $L'_{2,i}$ be the distance from $\mathbf{q}'_{2,i}$ to $L_2$ along the rasterized line at orientation $\theta_1$ that contains $\mathbf{q}'_{2,i}$. Then the algorithm for maintaining the list $\{(\mathbf{q}'_{2,i}, L'_{2,i}), i = 1, \ldots, k'\}$ as $L_2$ is traversed can be stated as follows:

ALGORITHM FOR IMPLEMENTING THE FIRST PASS RULE.

$k' \leftarrow 0$
**for** $k = 1, \ldots, n$
  **if** $L_{2,k} \neq \infty$ **then**
    $i \leftarrow 1$
    **while** $i \leq k'$ **and** $L'_{2,i} < L_{2,k}$
      $i \leftarrow i + 1$
    $(\mathbf{q}'_{2,i}, L'_{2,i}) \leftarrow (\mathbf{q}_{2,k}, L_{2,k}), k' \leftarrow i$

Note that the $L'_{2,i}$ and $\mathbf{q}'_{2,i}$ can be changed by replacement and $k'$ can increase, decrease, or remain constant as $L_2$ is traversed. Also, at each step of traversal along $L_2$, the sequence $\{L'_{2,i}, i = 1, \ldots, k'\}$ that results is increasing.

3.2.2.2. *The Second Pass Rule.* The *second pass rule* applies to intermediate array elements $\mathbf{s}_{2,i}$, $i = 1, \ldots, k$, on rasterized lines $L_2$ from the second pass, and it can be stated as follows:

SECOND PASS RULE. If $D(x, y)$ is nondecreasing in both $|x|$ and $|y|$, then the pixel $\mathbf{q}_{2,i}$ referenced by intermediate array element $\mathbf{s}_{2,i}$ does not need to be considered in determining the directional FT of $\mathbf{s}_{2,k}$ if the distance between $\mathbf{s}_{2,k}$ and $\mathbf{s}_{2,i}$ (i.e., $D(\mathbf{s}_{2,k} - \mathbf{s}_{2,i})$) is greater than or equal to the distance between $\mathbf{s}_{2,k}$ and the directional FT of $\mathbf{s}_{2,k-1}$.

Note that the distance from $\mathbf{s}_{2,k}$ to the directional FT of $\mathbf{s}_{2,k-1}$ is less than or equal to $D(1,1)$ plus the directional DT $d_{2,k-1}$ of $\mathbf{s}_{2,k-1}$. In the brute force approach, $\mathbf{p}_{2,k}$ is found by computing the distance between $\mathbf{s}_{2,k}$ and $\mathbf{q}_{2,i}$ for $i = 1, \ldots, k$. If the second pass rule is used, then $\mathbf{p}_{2,k}$ can be found by computing the distance between $\mathbf{s}_{2,k}$ and points $\mathbf{q}'_{2,i}$, $i = k_0, \ldots, k_1$, where $\{\mathbf{q}'_{2,i}\ i = k_0, \ldots, k_1\}$ is a subset of $\{\mathbf{q}_{2,i}, i = 1, \ldots, k\}$ $(k_0 \geq 1, k_1 \leq k)$.

As before, let $L'_{2,i}$ be the distance from $\mathbf{q}'_{2,i}$ to $L_2$ along the rasterized line at orientation $\theta_1$ that contains $\mathbf{q}'_{2,i}$. Similarly, let $\mathbf{s}'_{2,i}$ be the intermediate array element on $L_2$ that references $q'_{2,i}$. Then the algorithm for maintaining the list $\{(\mathbf{s}'_{2,i}, \mathbf{q}'_{2,i}, L'_{2,i}), i = k_0, \ldots, k_1\}$ as $L_2$ is traversed can be stated as follows:

ALGORITHM FOR IMPLEMENTING THE SECOND PASS RULE.

$k_0 \leftarrow 1, k_1 \leftarrow 0, d_{2,0} \leftarrow \infty$
**for** $k = 1, \ldots, n$
  **if** $L_{2,k} \neq \infty$ **then**
    $i \leftarrow k_0, k_1 \leftarrow k_1 + 1,$
    $(\mathbf{s}'_{2,k_1}, \mathbf{q}'_{2,k_1}, L'_{2,k_1}) \leftarrow (s_{2,k}, \mathbf{q}_{2,k}, L_{2,k})$
    **while** $i \leq k_1$ **and** $D(\mathbf{s}_{2,k} - \mathbf{s}'_{2,i}) \geq D(1,1) + d_{2,k-1}$
      $i \leftarrow i + 1$
    $k_0 \leftarrow i$

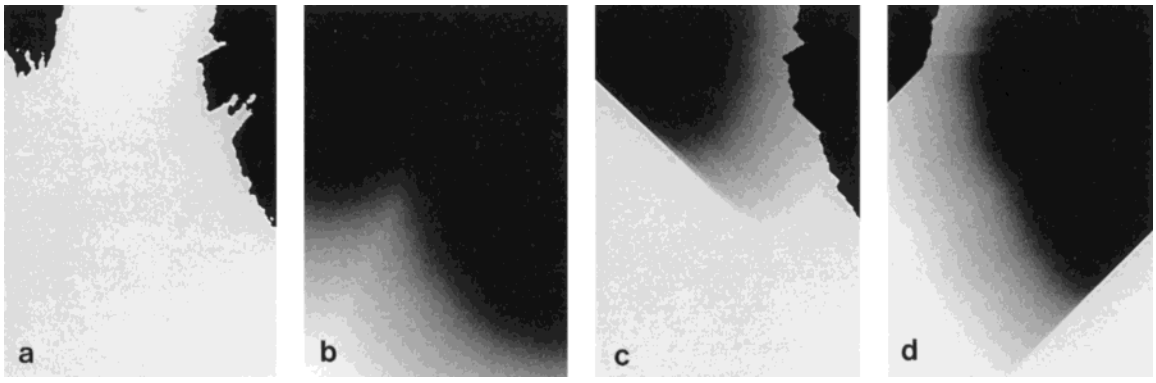


**FIG. 11.** A bit map representing a horizontal cross-section of terrain and some of its Euclidean DTs. (a) Cross-section; (b) unconstrained DT; (c) directional DT: $(\theta_1, \theta_2) = (180°, 225°)$; (d) directional DT: $(\theta_1, \theta_2) = (315°, 360°)$.

Note that $k_0$ and $k_1$ cannot decrease, but they can increase (and by different amounts) as $L_2$ is traversed. Also, at each step of traversal along $L_2$, the sequence $\{\mathbf{s}'_{2,i}, i = k_0, \ldots, k_1\}$ that results consists of points that get ever closer to the pixel $\mathbf{s}_{2,k}$ of interest on $L_2$. Finally, since $\mathbf{s}_{2,k} - \mathbf{s}'_{2,i}$ is an ordered pair of integers, the distances $D(\mathbf{s}_{2,k} - \mathbf{s}'_{2,i})$ can be precomputed and stored in a look-up table if desired. This can reduce the computational cost of the second traversal pass if $D$ is expensive to compute (e.g., if a square root operation is required as for Euclidean distance).

The first and second pass rules should be applied in succession, i.e., the output $\{\mathbf{q}'_{2,i}, i = 1, \ldots, k'\}$ of the algorithm for the first pass rule should be applied as input to the algorithm for the second pass rule. The overhead incurred increases in nearly direct proportion to the average number of feature pixels that need to be considered in computing the DT of a bit map pixel. If this number is small (as is often the case), the computational savings will be much greater than the overhead incurred and the number of operations required to compute the DT of a bit map with $N$ pixels will be reduced to $O(N)$. However, the actual number of operations required to compute the directional DT of a bit map by rasterized line traversal does depend on the nature of the bit map.

An example of the efficiency of the directional DT algorithm is demonstrated with the aid of Fig. 11. Figure 11a depicts a horizontal cross-section of terrain as a bit map with 456 rows and 322 columns of pixels. The black regions represent slices through terrain and contain pixels of value 1. The white regions represent empty space and contain pixels of value 0.

Figure 11b depicts the unconstrained Euclidean DT of the bit map in Fig. 11a. The intensities represent integer portions of DT values clipped to 255. The image in Fig. 11b. thus gets darker as distance to the nearest black pixel in Fig. 11a decreases. Figure 11b contains no discontinuities in intensity because the DT is unconstrained.

Figures 11c and 11d depict directional Euclidean DTs of the bit map in Fig. 11a for $[\theta_1, \theta_2] = [180°, 225°]$ and $[315°, 360°]$. In Figs. 11c and 11d, the intensity of a pixel represents the integer portion of the Euclidean distance (in units of pixels and clipped to 255) to the nearest black pixel on the bit map that lies within the directional constraint region emanating from that pixel. Discontinuities from gray to white occur at bit map pixels with white directional constraint regions adjacent to pixels with constraint regions that contain black pixels. Discontinuities from gray to black occur at black bit map pixels adjacent to white pixels. It took approximately 4 s to generate each of these directional Euclidean DTs on a Sun Sparcstation 10 computer.

## 4. CONCLUSIONS

Height field ray tracing efficiency can be improved by preprocessing the height field such that the empty space above the height field surface is parameterized in terms of apex heights and opening angles of inverted cones of empty space whose vertical axes are regularly spaced. Ray tracing efficiency can be improved further by using inverted cones of empty space that have narrow sectors.

2D arrays of inverted cones of empty space whose vertical axes are regularly spaced (i.e., linear parameter plane transforms or PPTs) can be efficiently generated from distance transforms (DTs) of height field horizontal cross-sections. These cones can be generated within specific sectors by using directional DTs. The computational cost and memory requirements associated with a linear directional PPT containing a large number of sectors can be managed by quantizing the cone opening angle to a relatively small number of bits and by reducing the resolution of the PPT.

The error-free directional Euclidean DT of a bit map with $N$ elements can be efficiently generated by rasterized line traversal in as little as $O(N)$ operations, but the exact number of operations does vary between bit maps. Rasterized line traversal algorithms can be implemented serially or in parallel, and they produce error free results for a broad class of distance measures that includes both Euclidean and chamfer distance. Distance measures $D(x, y)$ that belong to this class have the property of being nondecreasing in both $|x|$ and $|y|$.

## REFERENCES


1. G. Borgefors, Distance transformations in digital images, *Comput. Vision Graph. Image Process.* **34,** 1986, 344–371.
2. B. Cabral, N. Max, and R. Springmeyer, Bidirectional reflection functions from surface bump maps, *Comput. Graph.* **21,** No. 4, 1987, 273–281.
3. D. Cohen and A. Shaked, Photo-realistic imaging of digital terrains, *Comput. Graph. Forum* **12,** No. 3, 1993, 363–373.
4. P. E. Danielsson, The Euclidean distance mapping, *Comput. Graph. Image Process.* **14,** 1980, 227–248.
5. L. De Floriani, B. Falcidieno, C. Pienovi, D. Allen, and G. Nagy, A visibility-based model for terrain features, in *Proceedings 2nd International Symposium Spatial Data Handling, 1986,* pp. 235–250.
6. L. De Floriani, G. Nagy and E. Puppo, Computing a line-of-sight network on a terrain model, in *Proceedings, 5th International Symposium Spatial Data Handling, 1993,* pp. 672–681.
7. P. Fisher, Algorithm and implementation uncertainty in viewshed analysis, *Int. J. Geograph. Inform. Systems* **7,** No. 4, 1993, 331–347.
8. F. Leymarie and M. Levine, Fast raster scan distance propagation on the discrete rectangular lattice, *Comput. Vision Graph. Image Process. Image Understanding* **55,** No. 1, 1992, 84–94.
9. N. Max, Horizon mapping: Shadows for bump-mapped surfaces, *Visual Comput.* **4,** 1988, 109–117.
10. F. K. Musgrave, *Grid Tracing: Fast Ray Tracing for Height Fields,* Research Report YALEU/DCS/RR-639, 1988.
11. D. Paglieroni, A unified distance transform algorithm and architecture, *Mach. Vision Appl.* **5,** 1992, 47–55.
12. D. Paglieroni and S. Petersen, Parametric height field ray tracing, in *Proceedings, Graphics Interface '92, Vancouver, British Columbia,* pp. 192–200.
13. D. Paglieroni and S. Petersen, Height distributional distance transform methods for height field ray tracing, *ACM Trans. Graph.* **13,** No. 4, 1994, 376–399.
14. J. Piper and E. Granum, Computing distance transforms in convex and non-convex domains, *Pattern Recognition* **20,** No. 6, 1987, 599–615.
15. A. Rosenfeld and J. Pfaltz, Sequential operations in digital picture processing, *J. Assoc. Comput. Mach.* **13,** 1966, 471–494.
16. L. van Vliet and B. Verwer, A contour processing method for fast binary neighborhood operations, *Pattern Recognition Lett.* **7,** 1988, 27–36.
17. H. Yamada, Complete Euclidean distance transform by parallel operation, in *Proceedings, 7th International Conference on Pattern Recognition, Montreal, Canada, 1984,* pp. 69–71.