Non-Photorealistic Rendering of Stylized Block Diagrams

James E. Mower
Department of Geography and Planning
AS 218, University at Albany
Albany, New York 12222 USA
jmower@albany.edu

ABSTRACT: This paper will introduce object-space procedures for extracting silhouettes, slope lines, and drainage features from DEMs to direct the non-photorealistic rendering of landforms as stylized block diagrams. The ultimate goal of this work is to produce fully-automated tools that can imitate the style of illustrations characterized by traditional pen and ink techniques. Through the implementation of a Java 3D API prototype, this project establishes a testing platform for the application of stylistic elements to linework representing surface form lines. This project will begin to explore the aesthetic effects of slope ranges, angles between adjacent surface facets, and drainage features on the rendering of silhouette lines, surface creases, and hachures. It also introduces the use of adaptively resampled TINs as a basis for perspective rendering applications.

KEYWORDS: Non-photorealistic rendering, silhouette, DEM, block diagram

Introduction

Block diagrams are large-scale illustrations of surface features and their underlying geologic structures, drawn from a particular point of view following the rules of artistic perspective. Armin Lobeck, whose *Block Diagrams* has remained one of the most influential guides to their manual construction, argues that these illustrations gain much of their effectiveness from their visual simplicity (Lobeck 1958, p. 1). This paper will explore the application of his aesthetic to the automatic construction of block diagrams from digital elevation models (DEMs) under the rubric of non-photorealistic rendering. Following a review of earlier work on automated systems, I will introduce a new testing platform for stylistic experimentation, provide the relevant algorithms underlying its implementation, and discuss the contributions of particular graphic elements that contribute to good expression of landscape form.

Previous Related Work

Over the past decade, the rapid development cycle of fast, relatively inexpensive graphics processing chips along with concomitant developments in 3D application programming interfaces (APIs) has fostered a period of intense growth in perspective rendering applications in cartography, GIS, and other, non-geographic, application areas. Much of this work has been related to realistic renderings of landscapes, whether for gaming, site planning, or other uses that require a relatively ‘complete’ description of a scene. Another rendering paradigm, non-photorealistic rendering (NPR), provides a closer analogy to the construction of traditional perspective illustrations in cartography by fostering the communication of specific, often scientific information through a language composed of symbols with well-understood meanings (Kennelly and Kimerling 2006). This definition is consistent with the goals of other cartographic forms that use the rules and constraints of generalization to establish a message for an image and to focus a viewer's attention upon it.

Starting with an image space approach, Saito and Takahashi (1990) lay out many of the ground rules for NPR using 2D digital images as their base data. Their successful demonstration of techniques for generating simplified line drawings from photographed 3D objects established a vocabulary for further NPR research on the extraction of edges, rendered as silhouette lines, and components of surface curvature, represented by hachures. Isenberg, et al. (2003) provide a comprehensive classification of NPR modeling techniques for both image and object space approaches along with definitions of terminology. Kennelly and Kimerling (2006) provide an excellent overview of those approaches that are specific to renderings of landforms, placing them in the context of the manual methods defined and described by Raisz, Imhof, Lobeck, and other 20th century cartographers and illustrators.

Most current NPR research in landform representation has focused on object space approaches, deriving perspective line drawn images entirely from 3D spatial data represented in a DEM. Lesage and Visvalingam (2002) use this approach to implement p-stroke sketching, which is itself closely related to the inclined contour method of Tanaka (1932). Buchin, et al. (2004) model surface slope to apply scanned, hand-drawn slope and loose lines from a look-up table, dependent on surface slope and lighting conditions.

This paper will focus on the development of a robust technique for representing silhouettes and hachures from adaptively resampled DEMs in the form of triangulated irregular networks (TINs). By resampling a DEM with greater tolerance for surface error away from a selected viewpoint, triangle areas remain relatively constant, given constant surface variability, with distance from the viewpoint under perspective rendering. It is thus possible to extract and render features from a resampled DEM without need for line simplification in a post-rendering operation. This paper will also introduce the extraction of drainage features from DEMs to aid in the rendering of view-independent features (creases) that are essential to surface interpretation

Modeling Surfaces with Linework

Unlike most photorealistic rendering applications which determine shading values for polygonal surface facets based on the angle between incident illumination and the surface normal, this project constructs line images as if they were lit strictly by an ambient (non-directional) light source. And while virtually all modern photorealistic rendering systems use some kind of hidden surface algorithm to ensure that surfaces near a viewer block the view of those further along the viewer's line of sight, this project must also account for the viewer's position in constructing silhouettes.

Throughout this project, all facets are represented as 3D triangles within a variable resolution TIN (VTIN). Each VTIN triangle maintains a list of its neighboring triangles. The VTIN is also indexed by edges, where each edge record includes the IDs of its bordering triangles.

Silhouettes

A silhouette segment is a rendered edge separating a visible and invisible facet, where visibility is determined by the angle made between the viewer's line of sight and the facet's surface normal (Figure 1). In this sense, visibility has nothing to do with the presence or absence of obscuring surfaces between the viewer and the facet in question. A silhouette can be thought of as a 'visibility ridge' from the viewer's point of view. Silhouette segments are computed locally as specified by Algorithm Build Silhouettes:

Algorithm Build Silhouettes

To build silhouette lines:

```
1. For each facet,
    1.1. Determine the angle between the viewer's line of sight and the surface normal
    1.2. If the angle is between 0 and 90 degrees, mark the facet as visible. Otherwise, mark it invisible
2. For each edge,
    2.1. If one of the bordering facets is visible and the other is invisible,
        2.1.1. Render the edge as a silhouette segment
```

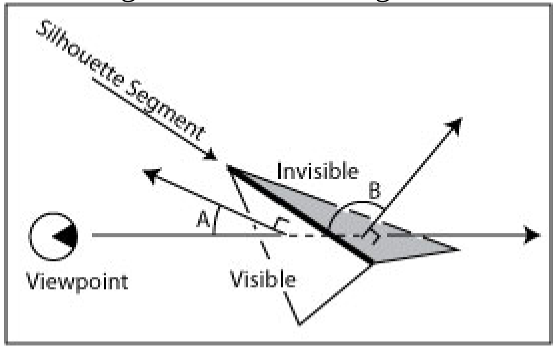

Figure 1. An edge between 2 surface facets is a silhouette segment if one facet is visible (with a normal less than 90 degrees from the line of sight (angle A)) and the other is invisible (normal greater than 90 degrees away from the line of sight (angle B)).

Note that hidden line removal is not an integral part of this algorithm. Instead, silhouette segments are rendered along with their referent surfaces, themselves rendered with emissive lighting in the background color of the rendered frame. Emissive lighting provides no differentiation in surface shading with respect to light source direction and surface normals. Surfaces rendered in this manner are invisible against the background, yet supply appropriate masking for silhouette segments (Figure 2).

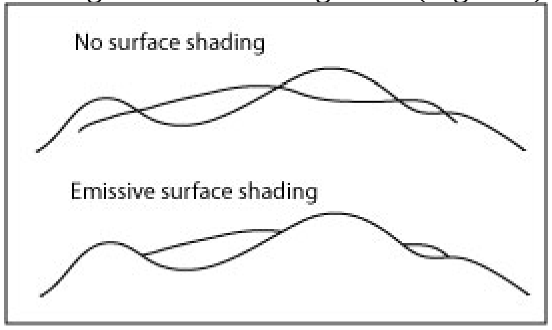

Figure 2. In the upper image, silhouettes are rendered without their associated surfaces and intersect inappropriately. In the lower image, surfaces are rendered with emissive shading in the background color, masking distant linework.

Slope Lines

On a planimetric topographic map, hachures are linear symbols representing paths of steepest slope on a surface patch, drawn perpendicular to contours. On a block diagram, slope lines perform a similar function. Although hand-drawn examples are typically rendered as complex curves, this project renders short slope lines as straight-line segments for initial simplicity of implementation (Lobeck 1958, p.33). Other non-silhouette lines in this project are referred to as creases and are rendered as sections of drainage networks (these are discussed in the section below on surface creases). The following algorithm, Build Slope Lines, outlines a procedure for constructing straight line segments along the line of steepest descent through a facet (Figure 3):

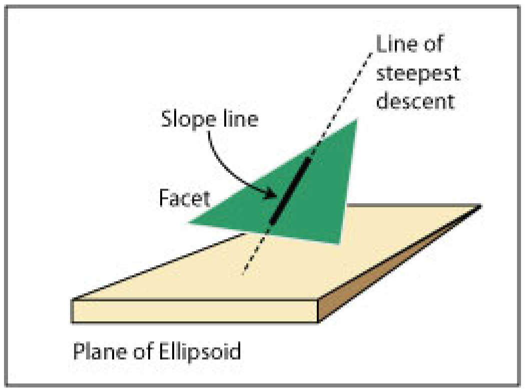

Figure 3. A slope line is computed as a segment of the line of steepest descent through the facet.

```
Algorithm Build Slope Lines
    To build slope lines:
    1.  For each facet,
        1.1.    Find the coordinates of the facet centroid
        1.2.    Find the equation of the plane of the facet
        1.3.    If the facet slope exceeds the preset minimum value,
            1.3.1.  Find the equation of the line of steepest descent through the facet
                    centroid
            1.3.2.  Render the segment of the line within the facet up to a preselected
                    margin bordering the edge of the facet.
```

Following common practice for manually constructed block diagrams, the algorithm creates slope lines of uniform width. Although it could be modified to scale the width of the rendered line to the slope of the facet, this is largely unnecessary. As vertical resolution decreases away from the viewpoint, the screen space areas of rendered triangles remain relatively uniform given constant surface variability. Therefore, the separation of rendered slope lines remains a good visual indicator of surface slope regardless of the 3D distance of the surface patch from the viewpoint.

Surface Creases

In the NPR literature, a crease is a line representing an edge that carries important visual information independent of the viewer's line of sight (Isenberg, et al. 2003, p. 28). For block diagrams, hydrographic features such as ridges and drainage channels would typically be resented by creases which can be extracted and rendered in a number of different ways. One such method examines the angle made by normals of adjacent triangles and renders their shared edge as a crease if the angle falls within a given range, either as a convex or concave feature with respect to the projected plane of the ellipsoid (Figure 4).

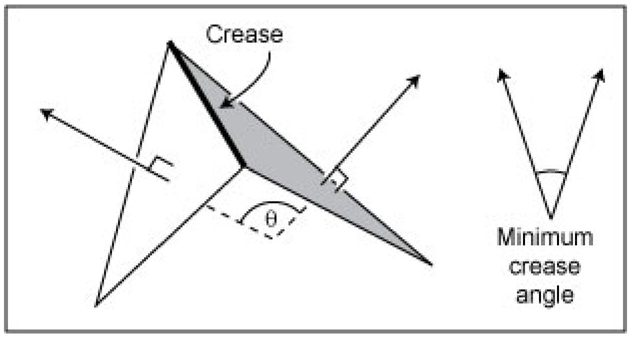

Figure 4. The angle θ between the 2 adjacent facets is greater than the minimum crease angle. The edge between the 2 facets will be rendered as a crease.

However, this analysis does not provide particularly good results for topographic surfaces. In relatively high-resolution elevation models (such as that used by this project), ridge and channel features rarely exhibit sudden slope breaks along connected triangle edges (Figure 5).

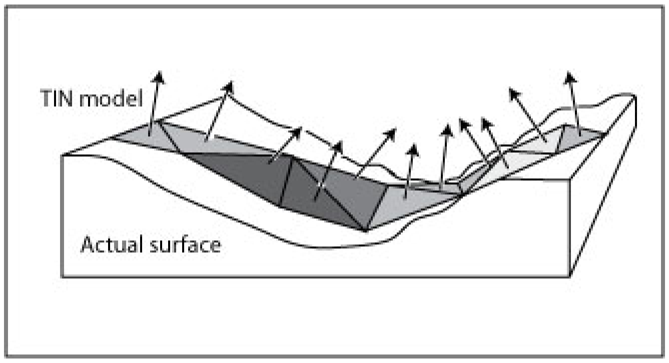

Figure 5. Although the actual surface presents a visually interesting feature, the feature is “spread out” among a region of triangles with relatively small slope transitions along adjacent edges. The crease detection method outlined in Figure 4 would likely fail here.

I found that a drainage accumulation analysis provided a visual solution much closer to that of hand-drawn illustrations. Algorithm Build Creases summarizes this approach:

Algorithm Build Creases

To build creases:

```
1. Initialize the drainID for each facet to RIDGE
2. For each facet,
   2.1. Find the centroid of the neighboring facet with the steepest downhill slope
        from the current facet. Note its ID as the current facet’s drainID. If a facet has
        no downhill neighbor, call it a pit.
3. Initialize all facets with 1 unit of input water and 0 units of accumulated water
4. While any but pit facets contains greater than 0 units of input water,
   4.1. For each facet,
        4.1.1. Decrement its total input water value and increment both the drainID
               facet’s input water value and its accumulated water value by the total
               input water value.
5. For each facet,
```

5.1. Render a line between the current facet's centroid and that of the drainID if the drainID facet's accumulated water value is greater than a preset threshold value.

Build Creases provides a relatively localized method for constructing drainage lines. Steps 1 through 2 create a drainage direction map, indicating the drain for each triangular facet. Following step 2, those triangles with no uphill source are on a ridge. Pits are triangles that have no exterior drainage.

Most drainage direction models built from data originally extracted from grid cell DEMs (like that of the current project) will find numerous 'false pits.' A false pit commonly occurs when a grid sampling pattern extracts an elevation from a stream channel but its 8 surrounding neighbors all remain uphill on the stream bank above it (Mower 1994). Since the channel itself is not explicitly extracted, channel samples are rarely adjacent to one another and thus have no drainage outlet. Unlike a realistic hydrological drainage model, the crease algorithm does not, for now, 'flood' false pits to assume a regular drainage pattern. This feature will be added in the next version.

Steps 3 through 4 run a drainage accumulation model through the drainage direction map. At each step of the algorithm, an uphill neighbor transfers its water supply to its downhill neighbor (indicated by drainID) and tracks the total amount of water passing through the downhill neighbor over the life of the procedure. Step 4 ends when all the remaining water is in the pits,

Step 5 inspects the drainage accumulation of each downhill neighbor of a facet. If the accumulated value is above a preset threshold, a line segment is rendered between the 2 triangle centroids. Depending on the value of the threshold, the procedure will draw more or less of the stream network detail. For the moment, at current threshold values, ignoring false pits does not negatively impact the line drawing. However, given that drainage networks are promising crease generators, future enhancements may require an additional flooding procedure to remove potential image artifacts.

The Implementation

The block diagram implementation has been written in Java and the Java 3D API. Although earlier work on this and related rendering projects used C# and the Direct3D API, I found that Direct3D (the Microsoft 3D rendering library) provided insufficient control for line widths, critical to the successful implementation of this project. Since OpenGL provides such control, and since the Java 3D API makes OpenGL calls on a client computer by default, this environment provides a natural platform for experimentation and collaboration across the Internet.

The implementation supports the rendering of a scene in layers. After loading the TIN, the user typically renders the scene with the emissive background color. For debugging purposes, the surface can be rendered with flat shading in any non-emissive color, enabling the type of image shown in Figure 6. Once the surface has been rendered, slope lines, silhouettes, or creases can be rendered with a button push. The user may toggle a rendered layer once it has been created and can zoom, pan, or tilt the surface with the mouse. Since the creation of silhouette lines is view dependent, the user would normally render a new set of silhouette lines after a significant shift in viewing position.

Adaptive Resampling and TIN creation

The project data originally came from grid cell DEMs in native USGS 1:24000 format produced by the New York State Department of Environmental Conservation and distributed by the Cornell University Geospatial Information Repository (CUGIR). For ease of use, the data was transformed into 10 km by 10km grids from which a dense TIN (DTIN) was created (Mower, in press). The DTIN is formed by a recursive resampling of the region files such that every retained sample is recorded along with an error term that measures its orthogonal distance from its parent plane. A variable TIN (VTIN) is created by scanning the DTIN and selecting those samples whose error term is greater than a tolerance whose value increases linearly with distance from a specified viewpoint. The VTIN provides a relatively constant number of samples per pixel for perspective rendering applications.

Results

Figures 6 through 10 show the renderings of a scene at the north edge of the Catskill mountains, starting with a rendered surface with flat shading for reference (Figure 6). All of the images were taken as screen shots from the current implementation, keeping the viewpoint, view angles, field of view, and other rendering parameters constant. Silhouettes, slope lines, and creases are purposefully not rendered within 3 km of the viewpoint world coordinates. Each of the screen shots were copied into Adobe Illustrator where they underwent a 70% uniform scale reduction (to fit the formatting constraints of this paper) and were framed with a border.

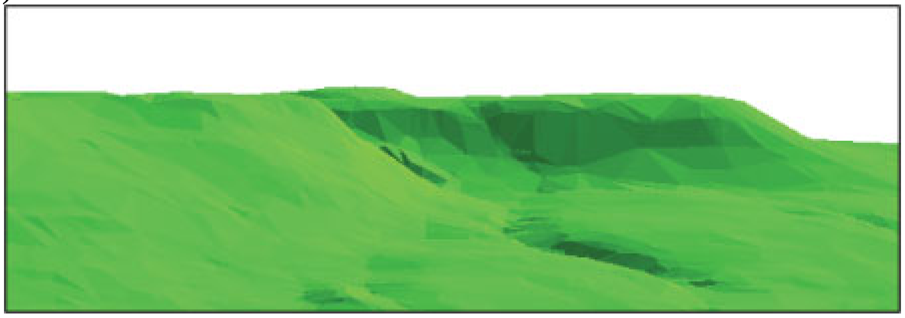

Figure 6. Flat shaded rendering. The images in this sequence share a common viewpoint, view angles, and other rendering parameters.

Figure 7 renders the scene with slope lines. Rendering was limited to only those facets whose slopes were greater than 22.5 degrees with respect to the ellipsoid. Since the VTIN maintains a relatively constant rendered triangle size throughout the scene, slope lines do not coalesce in the background. Some slight overlap of slope lines does occur along the central stream channel just below the dominant silhouette at the top of the image. The current implementation limits the rendering of slope lines to straight line segments.

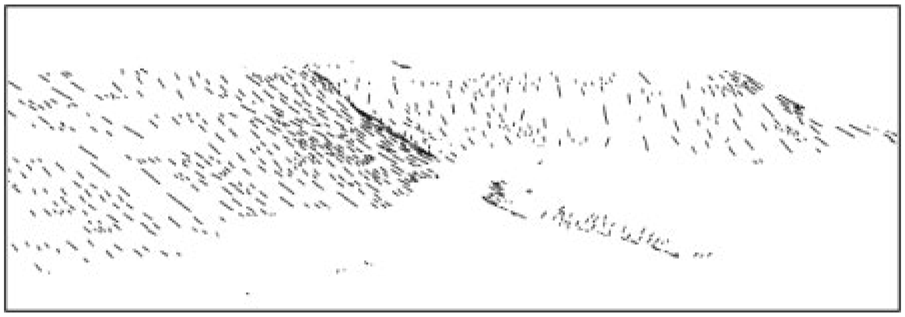
Figure 7. Slope line rendering. Only facets with slopes of 22.5 degrees and greater are rendered. This and the remaining figures also render the surface with white emissive rendering to enable hidden line removal.

Figure 8 renders the scene in silhouette lines. The image captures the profile of the hill and the spur extending down the center of the image particularly well but adds a few horizontal features to the lower right that, although true to the letter of the definition of a silhouette, would probably be ignored in a manual sketch.

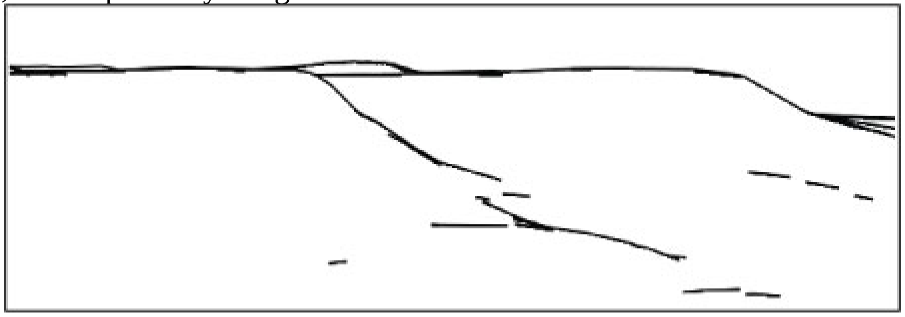
Figure 8. Silhouette rendering. Lines represent borders between edges facing toward and away from the viewer.

Figure 9 renders the drainage accumulation data as a set of crease lines. Although the image provides strong depth cues and models the north face of the Catskills appropriately, it provides a somewhat different view of the landform than do most traditional pen and ink sketches. In those produced by Imhof (2007) and others, where the erosive patterns of exposed rock may be the most important information in the image, creases typically run parallel to or branch away from ridges in an inverted tree pattern.

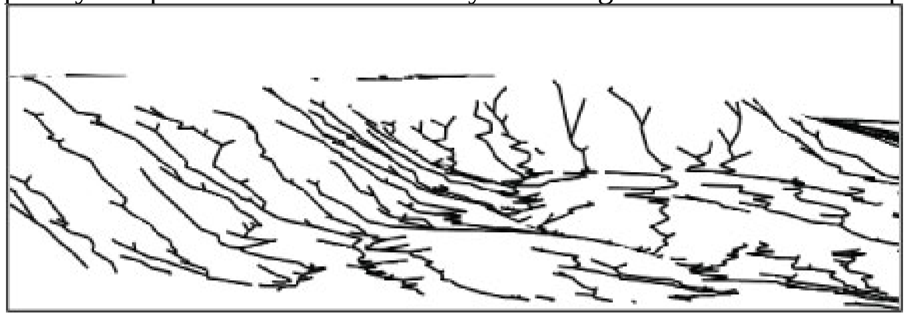
Figure 9. Crease rendering as drainage lines with white emissive surface rendering. False pits have not been removed from the drainage direction map.

Figure 10 shows a screen shot of the composite image. The remaining portion of this paper will outline several research paths that will move toward a more faithful pen and ink style representation, resulting in an improved visual experience.

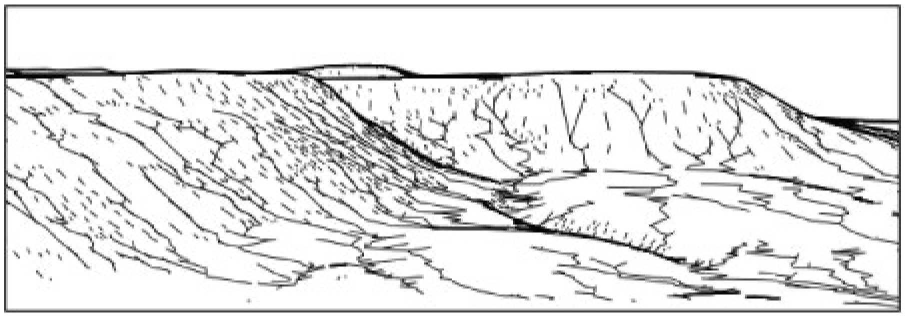

Figure 10. The composite image.

Future Work

The next phase of this project will investigate 2 major themes: 1) changing the representation of slope lines and creases from simple, straight segments to complex curves, and 2) implementing several features to support 'on the fly' calculations.

Improving the representation of creases and slope lines

Most pen and ink style illustrations implement slope lines as long, complex curves. Buchin, et al. (2004) describe a procedure for creating slope lines as an application of a vector field visualization, requiring a complex iterative procedure for line intersection detection. Such lines could be drawn more simply within a drainage framework by using the points extracted from the accumulation model as Bezier control points. Bezier curves are particularly attractive forms for this application because such curves are guaranteed to remain entirely within the convex hull described by their control points (Foley and Van Dam, p. 521). Thus, drainage lines for separate basins should not intersect one another. By lowering the minimum drainage accumulation value for rendering, more such lines could be rendered along slope faces. Points along drainage basin ridges (extracted, but unused in the current implementation) could also be rendered as Bezier segments, especially for surfaces modeling exposed rock faces. Similarly, the rendering of shorter, hatch-like segments within facets (slope lines in the current implementation) could also be represented with curves by using the positions of edge-adjacent facet centroids as directional weights.

The current implementation renders creases directly as hydrologic branching patterns based on drainage channels. Imhof (2007, pp. 235-260), provides numerous examples of rock face drawing in which both hydrologic channels and ridges are rendered by a variety of techniques, some involving connected, branching linework and other, more generalized drawings in which the linework is largely disconnected. The next implementation of this project will experiment with the rendering of hydrologic ridges and also with disconnected ridge and drainage network curves.

Other considerations for a complete application

To support development and debugging, the current prototype allows the user to recalculate slope lines, creases, and silhouettes at any time through a button push. In a finished application, silhouette construction would need to be recomputed on the fly given a change in the viewpoint. If its position shifts beyond an established threshold, the implementation would also need to calculate a new VTIN and underlying surface rendering. Furthermore, since the shift in position would change the resolution of the surface upon which the slope lines and creases are built, they should be recalculated as well. The next project will begin to implement automatic resampling and rendering on changes in the user's viewpoint. It will also clean up the drainage model by adding a procedure for false pit elimination.

Conclusions

Much of the beauty of pen and ink renderings of landscapes is expressed through an economy of strokes. It is hoped that this project and others like it will focus attention on the work of Lobeck, Imhof, and other masters of line drawing techniques and help to develop adequate methods for their imitation in digital form.

REFERENCES

Buchin, K., Sousa, M.C., Dollner, J., Samavati, F., Walther, M. (2004) Illustrating Terrains Using Direction of Slope and Lighting. *4th ICA Mountain Cartography Workshop in Vall de Núria, Catalunya, Spain, September 30-October 2 2004*, http://www.mountaincartography.org/ publications/papers/papers_nuria_04/buchin.pdf.

Foley, J.D. and Van Dam, A. (1984) *Fundamentals of Interactive Computer Graphics*. Reading, MA: Addison-Wesley Publishing Co.

Imhof, E. (2007) *Cartographic Relief Presentation*. Redlands, CA: ESRI Press.

Isenberg, T., Freudenberg, B., Halper, N., Schlechtweg, S., Strothotte, T. (2003) A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Computer Graphics and Applications*, 23, 4, pp. 28-37.

Kennelly, P.J. and A.J. Kimerling. (2006) Non-Photorealistic Rendering and Terrain Representation. *Cartographic Perspectives* 54, pp. 35-54.

Lesage, P.L. and Visvalingam, M. (2002) Towards Sketch-Based Exploration of Terrain. *Computers and Graphics*, 26, pp. 309-328.

Lobeck, A.K. (1958) *Block Diagrams and Other Graphic Methods Used in Geology and Geography*, Amherst, MA: Emerson Trussell Book Company.

Mower, J. E. (1994) Data-Parallel Procedures for Drainage Basin Analysis. *Computers and Geosciences*, 20, 9, pp. 1365-1378.

Mower, J.E. (in press) Creating and Delivering Augmented Scenes. *International Journal of Geographic Information Systems*.

Saito, T. and Takahashi, T. (1990) Comprehensible Rendering of 3-D Shapes. *Computer Graphics*, 24, 4, pp. 197-206.

Tanaka, K. (1932) The Orthographic Relief Method of Representing Hill Features on a Topographic Map. *Geographical Journal*, 79, 3, pp. 213-219.