DTM - DISPLAYED PERSPECTIVELY

Helmut Kager
Institute of Photogrammetry
Technical University of Vienna
Austria
Commission III

ABSTRACT

Thinking of graphical representation of digital terrain models one associates involuntarily: contour lines.

Despite of this, some DTM-users prefer to view their DTM so as they are used to see the original terrain - namely from a point of view chosen arbitrarily. This results in a perspective. This form of display seems to be useful e. g. for visual quality control in generating DTMs or in traffic design. Stereopairs may be of advantage too.

In this paper an algorithm is proposed which allows the perspective imaging of a DTM structured as raster-points (with break-lines in case) as delivered from the SCOP-program. Inner and outer orientation of the images is completely arbitary. Invisible (hidden) lines are displayed only on special request.

ZUSAMMENFASSUNG

Wenn man an die graphische Darstellung digitaler Geländemodelle denkt, assoziiert man unwillkürlich: Höhenlinienkartierung.

Jedoch wünschen sich manche DTM-Benutzer das DTM so zu sehen, wie sie es vom ursprünglichen Gelände gewohnt sind, nämlich von einem beliebig wählbaren Standpunkt aus - also als Perspektive. Nützlich erscheint diese Darstellungsform z.B. für die Qualitätskontrolle bei der Generierung des DTM oder für die Projektierung von Verkehrswegen etc.

Auch Stereoperspektiven können von Nutzen sein.

In diesem Aufsatz wird ein Algorithmus vorgeschlagen, der es erlaubt, ein rasterförmiges DTM (gegebenenfalls mit Bruchlinien) - wie es z.B. von SCOP erzeugt wird - bei beliebiger innerer und äußerer Orientierung des Bildes abzubilden. Unsichtbare Linien werden dabei nur auf Wunsch dargestellt.

1. INTRODUCTION

At first a question: What means *hiding* ?
The geometrical answer could be: *One object hides another object if the first one is nearer on the same visual path.*

Since a visual path is a straight line this definition implies a three dimensional formulation (collinearity equations) of the problem at first. Given some objects, we could test any two of them to see if they lie on the same halfray ending in our projection center. If they do, we test, which of the two objects is nearer: it hides the other. If not, they can't hide each other.

But for numerous objects - as may be supposed handling a DTM - comparing all pairs of objects would result in a vast amount of computation.

Let us consider another question: What exactly are these *objects* we just spoke

about ?
In the physical world, any non-transparent body could be such an object. A DTM belongs to the computational world and consists only of a set of numbers representing elevations above a regular grid. But this (fig. 1) appears to be very transparent.

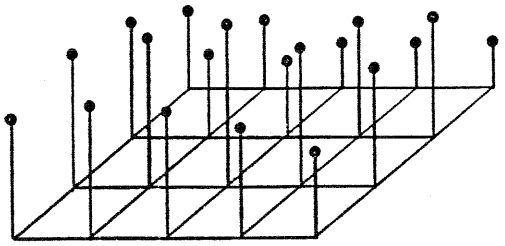


Fig. 1: DTM as elevation matrix

2. ELEMENTARY OBJECTS

If we connect immediately neighboured points we get a structure like a fisher's net (for simplicity we choose these connections as straight lines). It is transparent too, yet one might already imagine the surface as a suspended net.

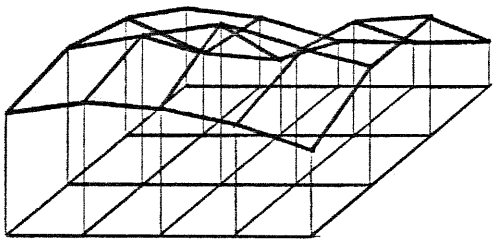


Fig. 2: DTM as suspended net

Now we could fill the meshes with opaque material. Since the four corners of a mesh aren't in a plane, the formuluation would become difficult. So we take another way. Onto each mesh side in the groundplan we put something like a firescreen: vertical trapezia, the upper edge coinciding with the fisher's net.

Now we have the objects questioned about: opaque trapezia, the upper edge only visible. These objects promise easy handling, well knowing they are only an approximation of a curved terrain surface.

In the moment we put one screen element into its place, we want to decide about its visibility. Fig. 3 shows a DTM partially set up with screen elements.

It gives rise to the next question: *Are there rules concerning the sequence of setting up the screen ?*

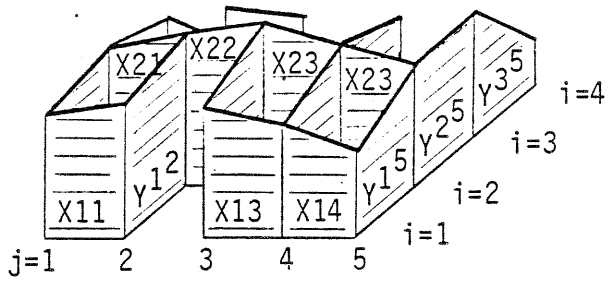


Fig. 3: Incomplete fire screen DTM

3. CONSTRUCTION OF THE OBJECT

If we put up only such screen elements which can't be hidden (partially or totally) by others, we don't have to care about comparisons between pairs of elements when considered as randomly distributed objects! The regular structure of our DTM grid does allow this.

Fig. 3 shows: Screen element X12 could hide screen element Y12 or X22 ; therefore X12 must be set up earlier than Y12 and X22 . More general: X1j elements can hide X2j elements; X2j elements may hide X3j elements and so on. A similar rule is valid for the Yij elements. So we can deduce priority rules based on grid-indices rather than on euklidean distances. A three-dimensional problem is reduced to two dimensions therewith.

But a threatening question occurs: What is the role played by the viewing point? The rules of hiding indicated above aren't general. From a viewing point far in the background of fig. 3, we see: X1j screen elements may be hidden by X2j elements, and so on. The Xij - rule is just reversed. The general rule for setting up screen elements may be seen in fig. 4, which is a groundplan of fig. 3: the viewing point P_o is chosen in mesh (2,2) .

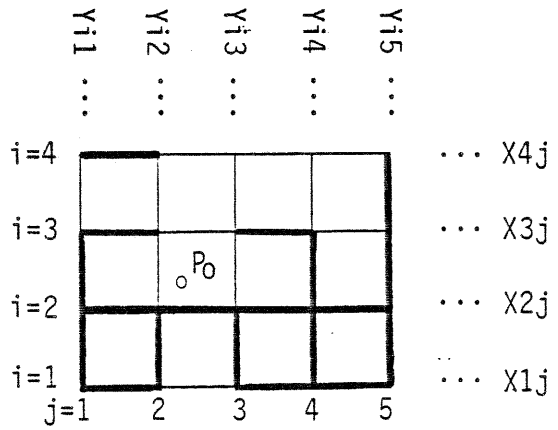


Fig. 4: Groundplan of the incomplete DTM shown in fig. 3

X3j - elements precede X4j - elements but
X2j - elements precede X1j - elements
Yi3 - elements precede Yi4 - elements & Yi5 - elements
Yi2 - elements precede Yi1 - elements

So we may formulate the hiding rule for screen elements:

Xij - elements precede Xkj - elements if i is *nearer* to k
Yij - elements precede Yil - elements if j is *nearer* to l

In a program, index loops should start near the projection center and run away using positive increments for positive coordinate direction, negative increments for negative direction. So each quadrant (referred to the projection centre as origin) is characterized by its own increment pair.

But: How do screen elements of different direction intervene?
Unfortunately Xij - elements and Yij - elements aren't independent.
If we want to display only profilelines our rules are sufficient, we may disregard the complementary screen set. For displaying both grid directions, we have to choose one direction as dominant, the other as subordinate, corresponding to the nesting of index loops. So we scan one quadrant of the grid. We put onto each grid point a pair of screen elements as fig. 5 shows:

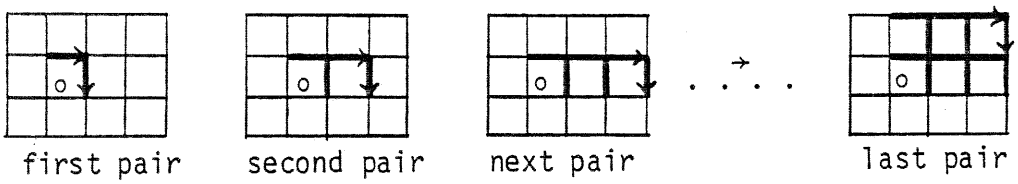


Fig. 5: Setting up pairs of screen elements in the first quadrant

We see: no member of any pair can obscure his partner, no pair can be hidden by a later one.

We continue with the 2nd Quadrant:

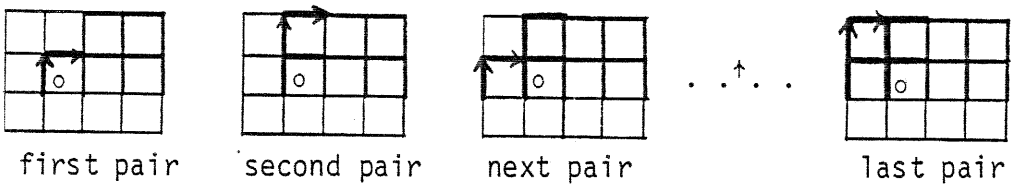


Fig. 6: Setting up pairs of screen elements in the second quadrant

And so on:

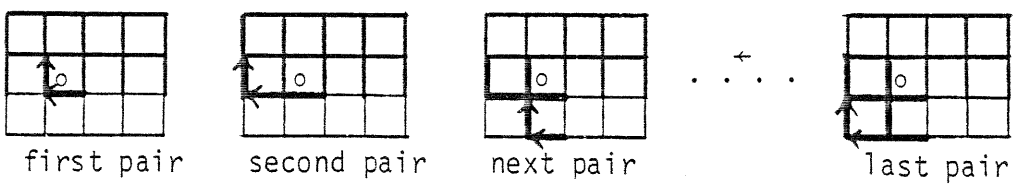


Fig. 7: Setting up pairs of screen elements in the third quadrant

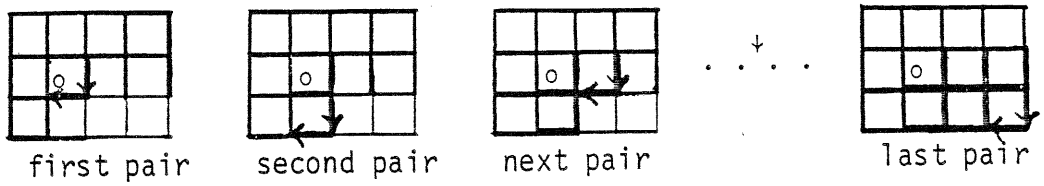


Fig. 8: Setting up pairs of screen elements in the fourth quadrant

When we look at the arrows, we see that they all obey the same *rotation* (from left to right) with respect to the viewing point. We will use this feature later.

Now we have filled the whole DTM with screen elements. Unnecessary doubles are contained merely in the mesh-strips starting from the viewing point following the four directions of wind. No problems will arise if we handle this special case when initializing our index loops.

As we have stated earlier, we don't want to compare individual screen elements with one another. The following question becomes so more and more pressing: When putting up any one of the screen elements, what really is its counterpart deciding its visibility?

4. THE HORIZON

For this purpose we use the perspective image of the highest contour of all screen elements already set up. This contour serves as horizon: It hides *lower* objects, but an object appearing *higher* modifies its shape (fig. 9).

This horizon will be called *hiding polygon* later on.

What happens when we set up a pair of screen elements?
At first normalized image coordinates of the upper two corners of the element are computed. Then this unique polygon side is searched for, the starting point belongs to. It is found, if this point is projected exactly *above* or *beneath* this side. Since the edges of screen elements appear to run from left to right the hiding polygon is then followed to the right too, until the image of this edge crosses the polygon or it ends. An intersection point is inserted into the hiding polygon as soon as it occurs. Points of the polygon bypassed *beneath* the edge are discarded from the hiding polygon. The point ending the edge ist inserted if it lies *above* the polygon. The hiding polygon requires therefore a highly dynamical data structure to ensure efficient behaviour of the whole algorithm. The second member of the screen pair is handled similarly. It should be noted, that there may occur only an even number of intersections when setting up one screen pair.

The screen edges respectively that parts of screen edges, which appear *above* the hiding polygon are output to a graphic device or written to a file as visible DTM: The normalized image coordinates are transformed into a fictitious user-camera, defined by inner orientation. This allows different inner-orientations for different - simultaneously serviced - graphic output devices. The complementary parts may be output too, but marked as invisible (e.g. as dashed line).

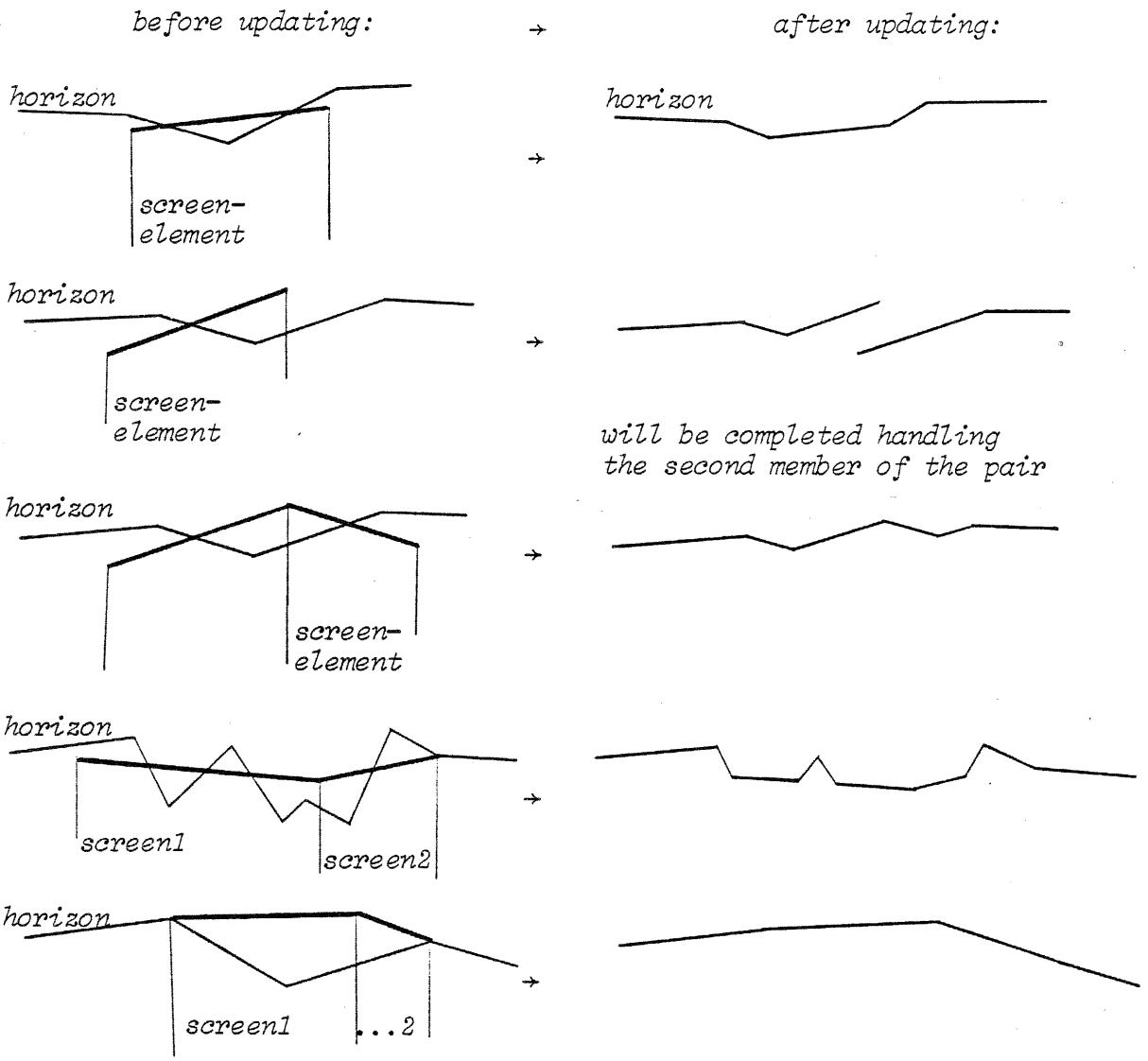


Fig. 9: Some situations updating the horizon

Repeating this process for all points with all screens we gain an image of the DTM in a central perspective (fig. 10).

5. WHAT PARAMETERS ARE USED TO CONTROL THE IMAGE ?

We have already mentioned the projection centre in its important role as starting point scanning the DTM-grid. The attitude of the image plane is determined by a rotation matrix defined using tree angles - as perceptual as possible - or defined with the coordinates of its axes. The vanishing plane (a plane parallel to the image plane through the projection centre) may dissect the DTM: the part *behind* it, is never visible. Beyond that, the image's attitude influences our hiding polygon, even left and right, up and down may be changed (rotation around the optical axis by π). But no problems arise, using vector algebra strictly (see appendix).

The inner orientation is completely arbitrary, using normalized coordinates throughout the algorithm. Only for outputting the image vectors, they are denormalized according to the data of interior orientation. At this stage the image vectors may be clipped to an rectangular image field so far as the algorithm hasn't done this, caused by sophisticated initialization of the hiding-polygon.

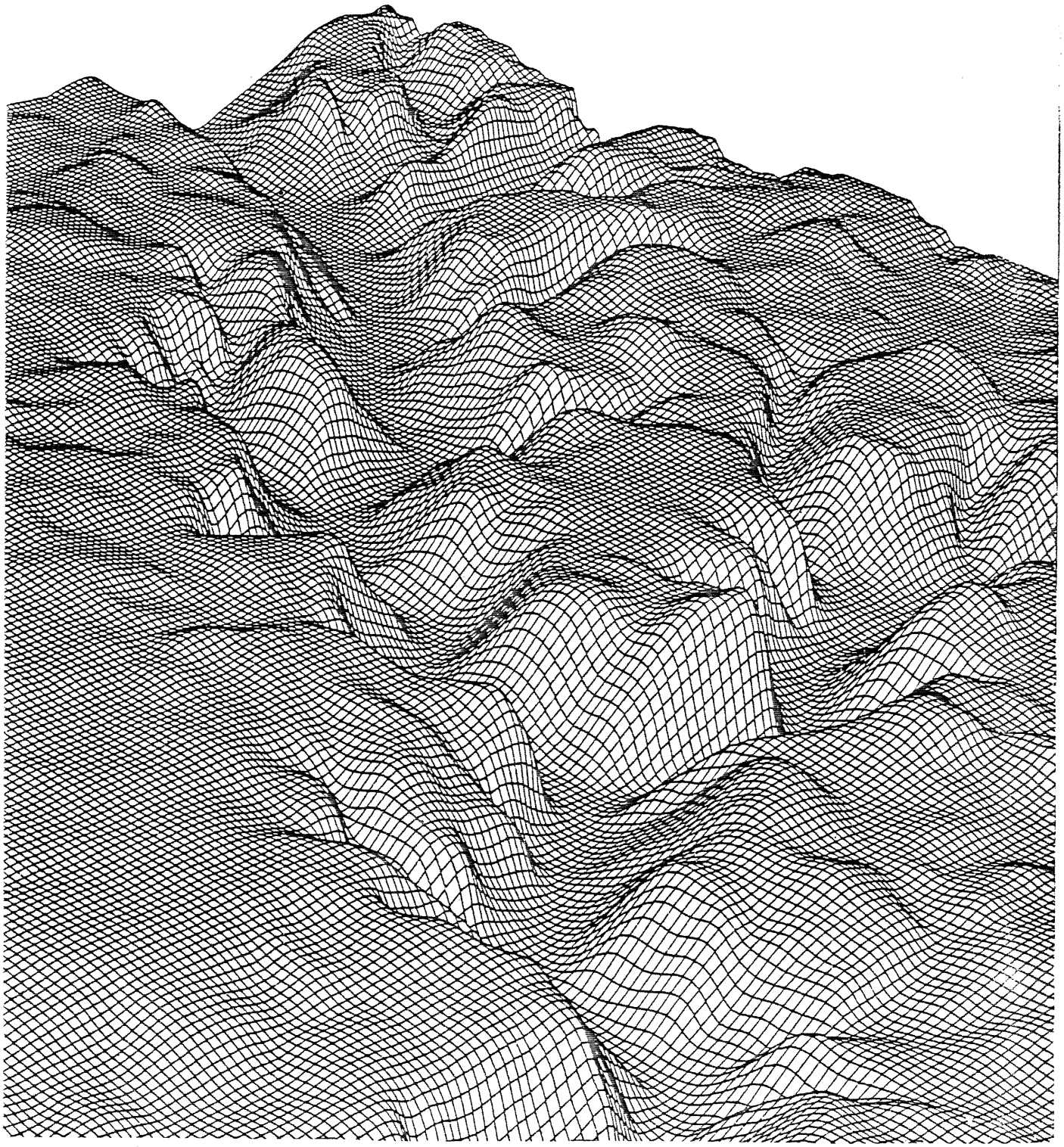


Fig. 10: Central Perspective of a DTM in Raster-structure (photogrammetric data acquisition: Landesvermessungsamt Stuttgart, computation with SCOP: Institut für Photogrammetrie Uni Stuttgart)

6. DTM STRUCTURE

Are there limits in the extension of the DTM?

A whole DTM can scarcely be stored in the main memory of a computer. The index loops scanning the DTM run into all wind directions, therefore not allowing sequential precessing. To minimize I/O we need an organisation of the DTM using random access. The SCOP-DTM is built up of so called *computing units*, shortly CU's, representing - simplified - relatively small submatrices of the whole DTM-grid. So handy data portions can be transferred.

Further more - implied by this structure - many computing units don't need to be read, since knowing the extremal heights of the DTM, they can be skipped immediately.

After reading a computing unit, a similar test is done using its extremal elevations. So some more CU's may be eliminated before checking visibility in detail.

In what way are the index loops affected by the DTM-structure?

If we let the index loops run up to the DTM border, we had to read many CU's again, as soon as processing the neighboured gridline. This isn't necessary anyway. CU's - taken as an unit - follow the same hiding rules as our screens. So we get further nesting levels for index loops applied to CU's. The index loops discussed earlier are limited to the CU-boundaries.

Empty CU's may be handled too, as fig. 11 shows.

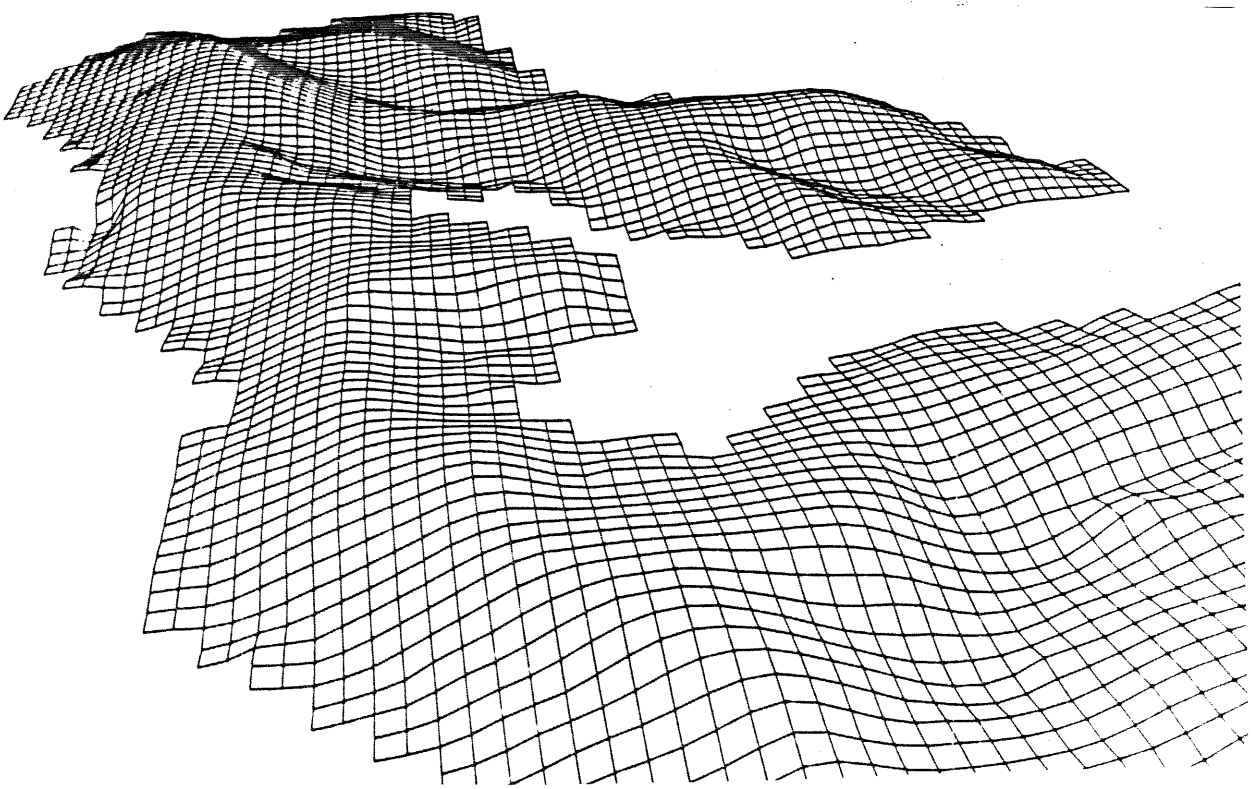


Fig. 11: Central Perspective of a DTM where Computing Units are missing (photogrammetric data acquisition: Consulting Engineer A. Legat, Leibnitz, Austria, computation with SCOP: Institute for Photogrammetry TU Wienna)

7. CONCLUDING REMARKS

The algorithm proposed was implemented in FORTRAN 4. It runs as part of the program system SCOP (Stuttgart Contour Program). It may be used also as stand alone program, provided that the DTM is organized in raster form. A sequential DTM may be converted to random format, using a special conversion program.

The SCOP-DTM contains some other valuable information too - as breaklines and borderlines. The next step in further development of the algorithm proposed, will be the inclusion of these informations. The concept of screen elements changing the horizon will be suitable too. The changing density of lines might cause difficulties for graphical presentation.

It should be noted, that the algorithm works very efficient, since each computing unit of the DTM needs to be read only once at the worst. So it is best suited for quality control during the process of DTM-generation.

It can be run interactively too, so supporting the design phase in road-building.

APPENDIX

The mathematical tools used:

perspective transformation:

$$\underline{x} - \underline{x}_0 = \lambda \cdot \underline{R}^T \cdot (\underline{X} - \underline{X}_0)$$

$\underline{x} - \underline{x}_0$: image-point; $\underline{x}_0$: inner-orientation; λ: point scale; $\underline{R}^T$: rotational matrix; $\underline{X}$: object point; $\underline{X}_0$: projection center

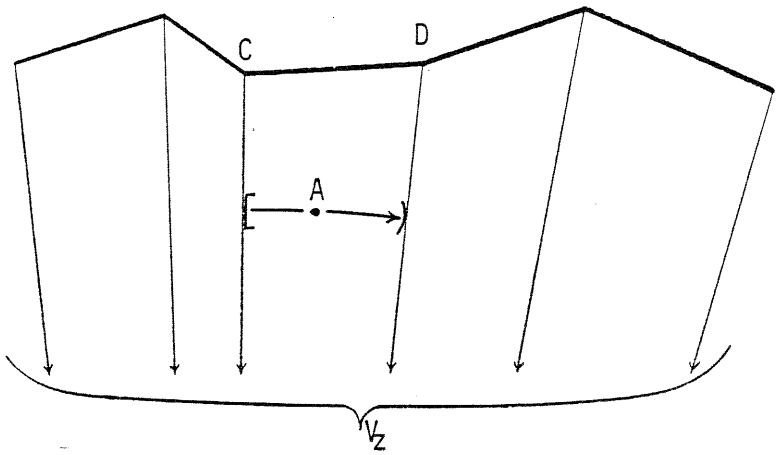
normalized image coordinates $(\bar{x}, \bar{y})$:

$$\begin{pmatrix} \bar{x} \\ \bar{y} \\ 1 \end{pmatrix} = \bar{\lambda} \cdot \underline{R}^T \cdot \begin{pmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{pmatrix}$$

user's image coordinates (x, y):

$$\begin{pmatrix} x - x_0 \\ y - y_0 \\ -c \end{pmatrix} = \frac{\lambda}{\bar{\lambda}} \cdot \begin{pmatrix} \bar{x} \\ \bar{y} \\ 1 \end{pmatrix} = \mu \cdot \begin{pmatrix} \bar{x} \\ \bar{y} \\ 1 \end{pmatrix}$$

A point (A) and its interval ($\overrightarrow{CD}$) on the horizon (i. e. the hiding polygon)



V_z: Direction to the vanishing point of all plumblines

A lies *between* $\overrightarrow{CD}$:

$$\begin{Vmatrix} \bar{x}_c & \bar{x}_a & \bar{x}_v \\ \bar{y}_c & \bar{y}_a & \bar{y}_v \\ 1 & 1 & \bar{z}_v \end{Vmatrix} \geq 0 \quad \text{and} \quad \begin{Vmatrix} \bar{x}_a & \bar{x}_d & \bar{x}_v \\ \bar{y}_a & \bar{y}_d & \bar{y}_v \\ 1 & 1 & \bar{z}_v \end{Vmatrix} > 0$$

A lies *beneath* $\overrightarrow{CD}$:

$$\begin{Vmatrix} \bar{x}_c & \bar{x}_d & \bar{x}_a \\ \bar{y}_c & \bar{y}_d & \bar{y}_a \\ 1 & 1 & 1 \end{Vmatrix} > 0$$