Calculation of Visibility from Raster Relief Models

Tomaž Trobec[†], Borut Žalik[‡], and Nikola Guid[‡]

[†]ZUM - Urban & Regional Planing, Engineering ltd
Grajska 7,SI-2000 Maribor,Slovenia
e-mail:zummb@medinet.si

[‡]Faculty of Electrical Engineering & Computer Science
University of Maribor
Smetanova 17, SI-2000 Maribor, Slovenia.
e-mail: {zalik, guid}@uni-mb.si

Abstract: This article is dealing with a determination of visible areas of a terrain represented by a raster model. Each pixel of the model displays a small area of a real terrain containing an altitude from the sea level. In this article, two algorithms for determining visibility of such areas are presented. The first one tries to reduce the error in the raster representation by the linear interpolation of the altitude of neighbouring pixels. The second algorithm does not care about errors in raster representation and thus, the algorithm is considerably quicker. Both algorithms work in $O(n^2)$, where n is the size of a raster square.

Key words: computational geometry, GIS, visibility.

1. Introduction

In many cases, the visibility between two arbitrary points located on the surface of the Earth is needed. Surveyors need to know if they can survey distances or angles between two points, radio transmitters transmitting ultra short waves must be visible from large areas to be effective, attractive tourist areas need sometimes a good overview of their surroundings etc. To support the search for such locations, the help of the computer is expected. The problem can be observed from different viewpoints considering representation of a terrain. For example, the model of a terrain can be represented by an interpolation surface, by a contour model, by polyhedral terrain model [DEFL93, STEW97, STEW98], or as a grid of rectangularity arranged 3D points [SAME90a, SAME90b]. Although being obtained through the process of interpolation the first two representations are considered as more accurate in comparison with the raster representation. However, for our task, to find all visible parts of a terrain from a given standpoint, the raster representation has some important advantages:

- Having the raster model of a terrain some simplification of the terrain data has already been done. Namely, each pixel in the raster representation represents a small square area of the terrain, and contains the average sea level of this portion of the terrain.
- Raster model of the terrain is standardised and easily obtainable in a computer form.
- Working with raster data there is the possibility to implement rapid algorithms using just with integer arithmetic.
- The result can be represented very quickly and in the way the users (landscape designer) can “read” is easy.

These were the reasons why the raster representation of the terrain is being chosen as the best suitable for our task.

The paper is divided into five chapters. The background of the problematic is given in the second chapter, in the third chapter two algorithms coping with the terrain visibility problems are considered. The fourth chapter discusses the time complexity and gives practical measurements of run time of both algorithms. The last section gives some new ideas how to improve the explained algorithms.

2. Background

A raster model of a terrain consists of a 2D array of pixels. The dimension of pixels must be chosen reasonable. Too small pixels give very large array of pixels, while too large pixels do not approximate well the rocky terrain. This relationship between pixel size and the size of the array is exponential (reducing the size of pixel on a half causes increasing four times the size of the array). As mentioned in the introduction, each pixel has associated a value representing the average sea level of the terrain.

The visibility problem is very easy to state. Let us have an array of pixels T representing the terrain. Let $\mathbf{p} \in T$ be the pixel representing the standpoint of the observer. We mark all pixels $\mathbf{q_i} \in T$ being visible from **p** with one colour, and all other pixels, which represent invisible regions of the terrain with other colour. The pixel visibility is influenced also by Earth curvature, especially if larger terrain is being explored. For our purposes, the Earth is considered as being a sphere. Because the visibility is calculated according to the plane, which passes through pixel **p** and which normal points to the centre of the sphere, the altitudes of all remaining pixels are decreased by the value dh. Fortunately, the radius of the Earth is very big, and therefore, lines SA, SB are almost parallel what means, that the angle between them are very small. In this way, the correction of the altitude can be done using the formula depicted on Figure 1.

However, the influence of the Earth curvature has to be taken into account. For example, when the length of the arc L is 50 km, distance D is 40.005 km, and **dh** is 195,98 m.

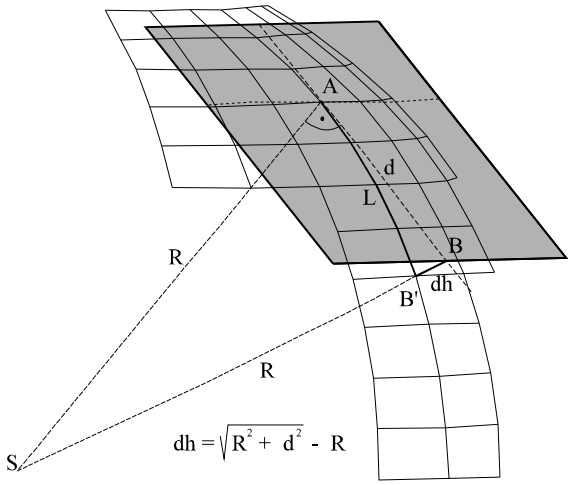

Figure 1. Influence of Earth curvature

3. A basic visibility algorithm

To solve the visibility problem, two algorithms have been developed. They differ in accuracy and in time needed to generate the visible map. However, at first, let us consider a visibility criterion used in the raster terrain representation.

3.1 Visibility criteria

3.1.1 A basic visibility criterion

The most natural way to solve our problem is to form a sweep line from the pixel **p** representing the standpoint to the border pixel of our array T. The sweep line is then rotated around the pixel **p** passing through each border pixel. For each position of the sweep line, all visible pixels have to be somehow determined. A key for this determination is an introduction of a visibility angle ϑ. As shown in Figure 2 ϑ is measured from a zenith to the pixel $\mathbf{q_i}$ being just observed. Pixels on the sweep line are scanned starting at the first standpoint neighbouring pixels $\mathbf{q_1}$ and moving slightly to the border pixel $\mathbf{q_n}$ of the array T. It is easily to see the pixels are visible until $\vartheta_i > \vartheta_{i+1}$ (pixels $\mathbf{q_1}$, $\mathbf{q_2}$, $\mathbf{q_3}$, and $\mathbf{q_4}$ in Figure 2). As soon $\vartheta_i \le \vartheta_{i+1}$, a sequence of invisible pixels starts (in Figure 2 $\vartheta_3 < \vartheta_2$). Let us store the visible angle of the last visible terrain pixel and denote it as ϑ_r. If pixel $\mathbf{q_j}$ with visibility angle $\vartheta_j < \vartheta_r$ is found, then this pixel is visible again, and the algorithm behaves as described. The simmilar approach is used by De Floriani and Magillo [DEFL93].

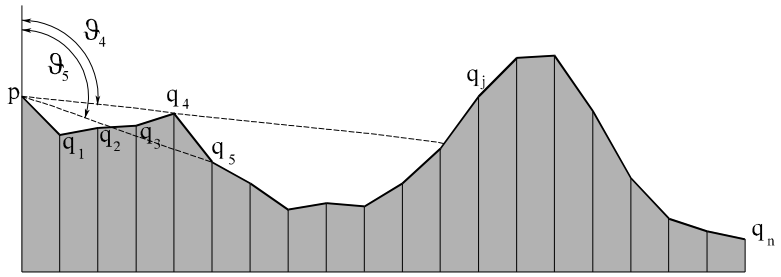

Figure 2. Determination of the visible pixels along the sweep line

The following pseudo code shows the basic algorithm for determination of visible pixels:

```
void CalcVisibPixels(pixel p, pixel qn)
{   CurVisibAng = 0;       VisibAng = 180; i = 1;
    do
    {   qi = NeighbourOf(p);
        CurVisibAng = CalculateVisibAngle(p, qi);
        if (CurVisibAng < VisibAng)
        {   VisibAng = CurVisibAng;
            qi->Visible = TRUE;
        }
        else qi->Visible = FALSE;
    } until (qi != qn);
}
```

At this approach, two problems are identified at first view:

- the sweep line has to be rasterized over the array T;
- the function *CalculateVisibleAngle* seems critical, because it contains probably some time-consuming trigonometric functions. However, they can be efficiently avoided by comparing just the ratio between difference dh of the altitude (*dh = abs(Altitude(**p**) - Altitude(**q**$_i$))*) and the distance dx (*dx = abs(**p**.x - **q**$_i$.x)*) as shown in Figure 3.

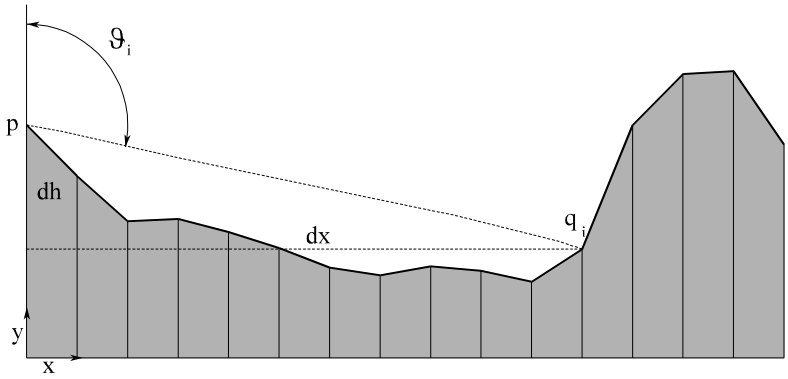

Figure 3. Rapid Visible angle determination

3.1.2 Improved visibility criterion

The improved visibility criterion has been set by taking into account, which operations are performed quickly in computer. We know, trigonometric functions are implemented as mathematical series and therefore are slow. The divisions and multiplications are slower than additions, subtractions, and comparisons. Therefore, a good computer visibility criterion must reduce slow arithmetic operations to the minimum. In our implementation, the following visibility criterion is used: a ray is constructed for each terrain pixel on a sweep line. Pixels are tested if they are below or under the ray. There are two possibilities (see Figure 4):

- If the considered pixel is below the ray, then it is invisible and the ray remains unchanged. To realise this case, only one addition and one comparison is needed.
- If the pixel is under the ray, the pixel is marked as visible and a new ray passing through the standpoint **p** and current pixel **q$_i$** is constructed.

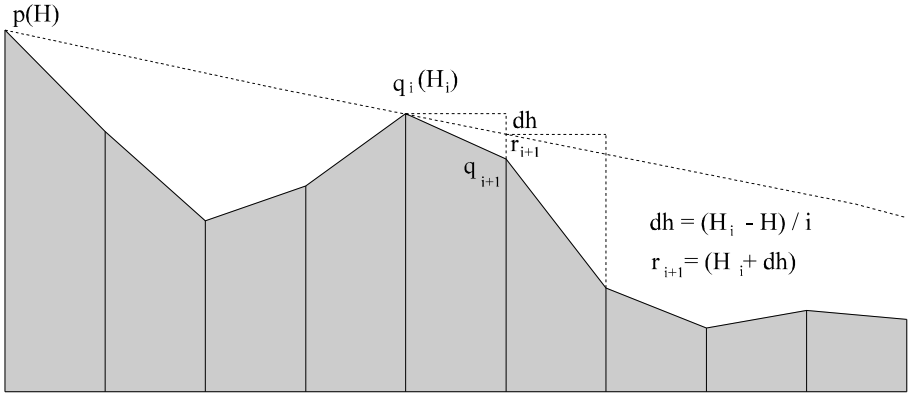

Figure 4. Improved visibility criterion

3.2 Terrain Pixel Altitude Correction Algorithm

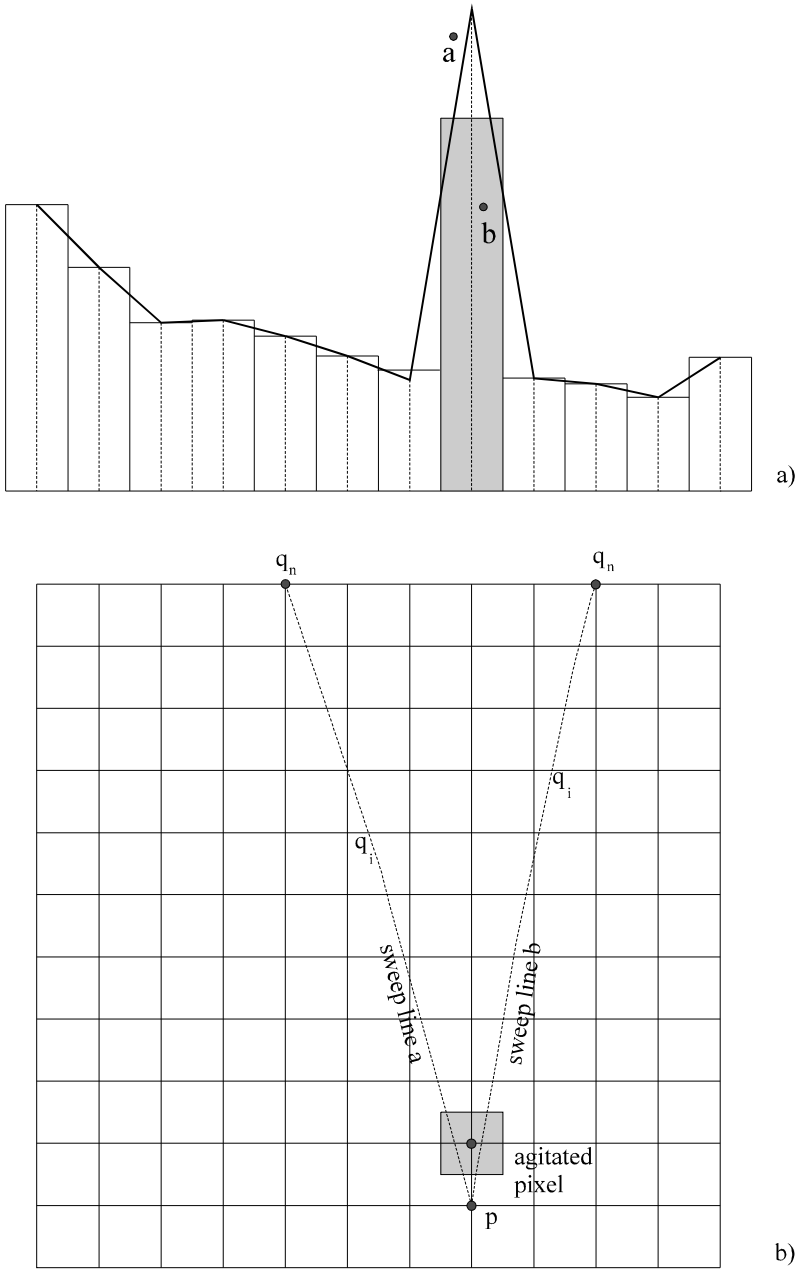

Figure 5. The average see level of terrain pixels

In a very agitated terrain where, by the way, determination of the visibility is the most important, it could happen that a high tiny pick can raise enormously the average see level of the terrain pixels covering this area. This causes that too much terrain pixel are marked as *Invisible*. If such a “high terrain pixel” is very close to the standpoint, the mistake can be noticeable. An example in Figure 5a shows that the terrain pixel shaded in grey is too wide to represent suitably the agitated terrain. This causes all pixels determined by the sweep line a (see Figure 5a and b) are marked wrongly as *Invisible* although they are visible in reality from the stand point s. To reduce this effect, an algorithm called Terrain Pixel Altitude Correction Algorithm - TePAC is

developed. The described drawback is minimised by interpolating the altitude of the terrain pixels on both sides of the sweep line. Figure 6 shows how this improvement is realised. Each terrain pixel gets so-called reference point. In our case, the reference points are placed in the middle of each terrain pixel. Through these reference points a virtual regular grid is laid. Let us consider a case when the slope of the sweep line is smaller than 45°. Then, at any sweep line which connects the reference point of standpoint **p** and the reference point of the border terrain pixel $\mathbf{q_n}$, intersections with vertical virtual grid lines are calculated (marked by "x" marker in Figure 6). It is easy to calculate the interpolated altitude at the intersection points. This interpolated altitude value is then used for the visibility calculation. If the slope of the sweep line is greater than 45°, the intersections between horizontal virtual grid lines and the sweep line are calculated.

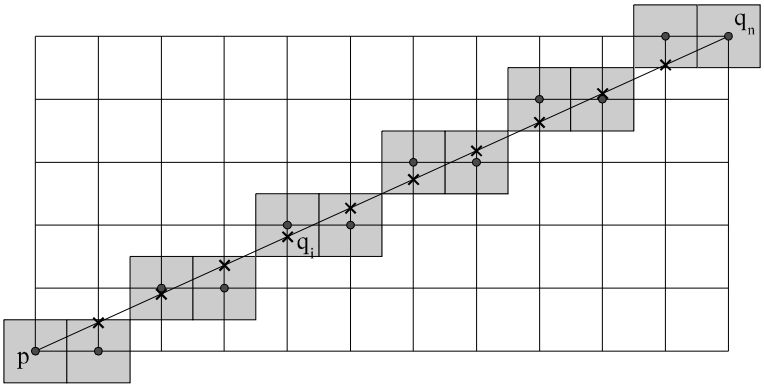

Figure 6. Altitude interpolation between terrain pixels

There is, however, no need to calculate the intersections. Instead, a simple incremental algorithm has been constructed. Let us consider how it works in the first quadrant. Let px is a drawing axis and it is initially set to the x coordinate of the standpoint reference point (px = p.x). At the each step of the algorithm, this value is incremented by Δx, which represents the distance between two neighbouring terrain pixels. The second variable is py, which is initially set to the y coordinate of standpoint reference point (py = p.y) and is incremented for the value δy, which is calculated as:

$$\delta y = \Delta x \, (q_n.x - p.x) / (q_n.y - p.y).$$

The pseudocod of the algorithm gives more details:

```
// initialyze the first pixel on the sweep line
Xtr := X+px;                                    xa  := Trunc(Xtr);
Ytr := Y+py;                                     ya  := Trunc(Ytr);
Dtr := pd;                                       Xtr := X+px;

if(abs(px) == 1)
{ while((Xmin  <=  xa)&&(xa <= Xmax))      //to the end of sweep line
  { ya = Trunc(Ytr);                       //calculate pixels on both sides of sweep line
    yb = ya + 1;
    Ha = Dmr.Get(xa, ya);                  //get their altitudes
    Hb = Dmr.Get(xa, yb);
```

```
    if(Ytr - ya)                              //interpolate altitude on the sweep line
      Ha = Ha + (Hb - Ha)*(Ytr - ya);

    if(EarthPlane)                            //Earth curvature corection
      Ha = Ha - (sqrt(EARTH_RR + sqr(Dtr * Dmr.Pix)) - EARTH_R);

    CurVisAng = (Ha - H)/ (Dtr * Dmr.Pix);    //visibility angle ration
    if(CurVisAng > VisAng)                    //determine visibility
     { VisAng = CurVisAng;
       P.Put(1, Round(Xtr), Round(Ytr));      //pixel visible
      }
    else
      P.Put(0, Round(Xtr), Round(Ytr));       //pixel invisible

    Xtr = Xtr+px;                             //move to the next pixel on the sweep line
    xa  = Trunc(Xtr);
    Ytr = Ytr+py;
    Dtr = Dtr+pd;
   }
}
```

3.3 Bresenham-Based Visibility Algorithm

If we do not expect the problems mentioned in the previous section, which are efficiently solved by TePAC algorithm, the visibility algorithm can run much faster giving us the first estimation of the visibility problem in a very short time. If a correction of the terrain pixel altitudes is not needed, there is no need to calculate the intersection points between virtual grid and sweep line. In this case, just terrain pixels which approximate the sweep line are needed (with other words, a line rasterisation has to be made). There are two groups of algorithms copping with line rasterisation problem:

- in the first group there are algorithms determining all pixels touched by the sweep line [CLEA88, AMAN87, ZEMČ93, ŽALI97],
- in the second group there are algorithms which trays to produce the visual acceptable rasterized line. The first algorithm from this group is well-known Bresenham algorithm [BRES65], but quicker algorithms also exist (for example Symmetric Double Step Line Algorithm [WYVI90]).

At our implementation, Bresenham algorithm is used at the moment (Bresenham-Based Visibility Algorithm - BBV). The BBV algorithm is simple. It performs rasterisation of the sweep line between points $\mathbf{p}$ and $\mathbf{q}_n$ and then for each terrain pixel it is considered if the pixel is visible or not using the visibility criteria already described.

4. Results and comparisons

Figure 7 shows a visibility map generated by the presented algorithms. In the upper left window (triglav.tvi) the raster terrain model from Triglav, the highest Slovenia pick is shown. The left

small window displays visibility map calculated by TePAC algorithm, and the right small window shows map obtained by BBV algorithm. The big right window (Matrix1) containts a comparition between results generated by both algorithms. The dark regions are visible and the white regions are invisible in both cases, the gray shaded pixles denotes differences between obtained results.

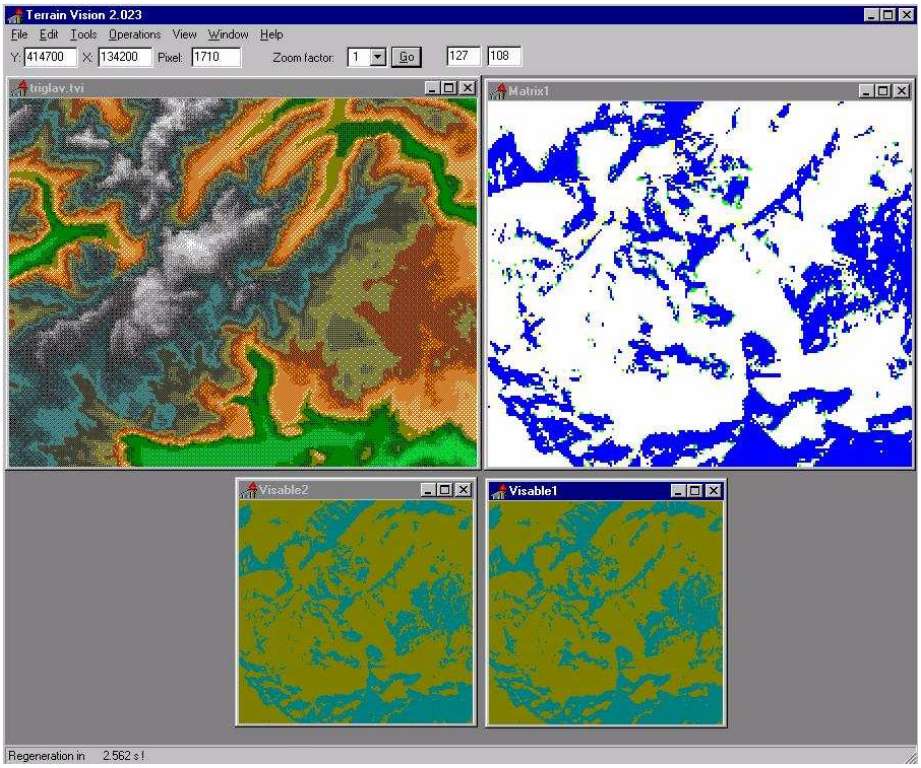

Figure 7. A snapshot of the visibility program

Theoretical estimation of the time complexity can be easily derived. Let us say the array of terrain pixels is a square with dimension of n pixels. From a standpoint to each edge pixel, a sweep line is constructed. There is exactly 4×n sweep lines. On each sweep line there is at most n×√2 terrain pixels when the sweep line goes through the square diagonal. In general, this number is smaller. The total number of pixels examined is therefore $4\times\sqrt{2}\times n\times n = O(n^2)$. However, theoretical estimation does not give us always a good filling about speed of the algorithm. In Table 1 and Figure 8 some statistical results measured on computer with i486 processor are given.

n	TePAC execution time (ms)	BBV execution time(ms)
20	0	0
40	60	0
80	220	0
160	830	60
320	3400	330
370	5389	490
600	11479	1260

Table 1. Execution times for both algorithms

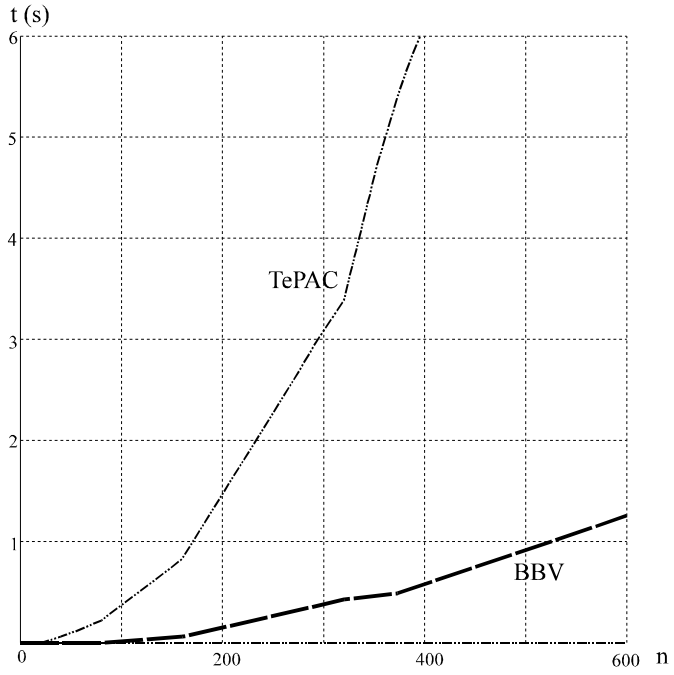

Figure 8. Graph of data from Table 1.

5. Conclusions

The paper considers the problem of a terrain visibility determination where the terrain is represented by a raster. In the first part of the paper, theoretical background is given. In the continuation, two algorithms coping with the mentioned problem are explained. The first algorithm, TePAC is more accurate, because it reduces the error arising during the rasterisation of the terrain. The second algorithm, BBV, does not care about rasterisation error and therefore it is much quicker. The best results in practice are obtained by combining both algorithms when the larger terrain maps are solved with BBV algorithm, while detailed analysis is always performed on smaller regions, where TePAC is efficient enough. Both algorithm take into consideration the curvature of the Earth.

The visibility problem can be solved concurrently, because the processing of a sweep line is totally independent. At the moment, the implementation of a parallel version of both visibility algorithms is in the initial phase.

The next interesting field is 3D extension of the visibility algorithm. The 3D visualisation should be quick, because invisible pixels are already determined. Terrain pixels are converted to voxels (or bricks) and since they are already sorted on each sweep line, 3D visualisation with the painting algorithm could be done rapidly.

6. Literature

[AMAN87] Amanatides, J., and A. Woo, “A Fats Voxel Traversal Algorithm for Ray Tracing”, in Proceedings of Eurographics’87, G. Marechal (ed.), North-Holland, pp. 3 - 10, 1987.

[BRES65] Bresenham, J. E., “Algorithm for Computer Control of a Digital Plotter”, IBM System Journal, Vol. 4, pp. 25-30, 1965.

[CLEA88] Cleary, J. C., and G. Wyvill, “Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision”, The Visual Computer, Vol. 4, pp. 65-83, 1988.

[DEFL93] De Floriani, L. and P. Magillo, “Computing Visibility Maps on a Digital Terrain Model”, In: “Spatial Information Theory - A Theoretical Basis for GIS”, A. U. Frank, I. Campari (Eds.), Lecture Notes in Computer Science, No.716, Springer-Verlag, 1993, pp. 248-269

[SAME90a] Samet, H., “The Design and Analysis of Spatial Data Structures”, Addison-Wesley, 1990.

[SAME90b] Samet, H., “Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS”, Addison-Wesley, 1990.

[STEW98] Stewart, A. J., “Fast Horizon Computation at All Points of Terrain with Visibility and Shading Applications”; (downloaded from “http://www.dgp.toronto.edu /people/JamesStewart/visibility/”).

[STEW97] Stewart, A. J., “Hierarchical Visibility in Terrains”, Eurographics Rendering Workshop, 1997 (downloaded from “http://www.dgp.toronto.edu/people/ JamesStewart/papers/egrw97.html”).

[WYVI90] Wyvill, B., “Symmetric Double Step Line Algorithm” in Graphics Gems, A. S. Glassner (ed.), Academic Press, pp. 101-104 1990.

[ZEMČ93] Zemčik, P., “An Efficient Algorithm for 3D Line Generation”, Machine Graphics and Vision, Vol. 2, pp. 231-235 1993.

[ŽALI97] Žalik, B., G. Clapworthy, and Č. Oblonšek, “An Efficient Code-Based Voxel-Traversing Algorithm”, Computer Graphics Forum, Vol. 16, pp. 119-128, 1997.