A New Approach to Perspective Views of Spherical Coordinate Functions *

Alessandro Amoroso†, Giulio Casciola ‡

Abstract

We present an algorithm for rendering perspective views of grid surfaces for continuous functions defined in spherical coordinates where hidden lines are removed. The algorithm operates in screen coordinates, is pixel exact and renders any continuous function from any viewpoint. The worst-case time complexity of the algorithm is linear in the number of facets and less than linear in the size of the viewport. Our experiments show that for dense grids it provides high-quality images at very low cost.

Categories and subject descriptors from Computing Reviews: I.3.1[**Computer Graphics**] Raster display devices, I.3.5[**Computer Graphics**] Surfaces, Geometric algorithms, I.3.7[**Computer Graphics**] Hidden line/surface removal, G.4[**Mathematical Software**] Algorithm analysis.

General Terms: algorithms, theory, performances.

Additional keywords: hidden line elimination, spherical coordinate functions, screen coordinates, pseudopolar coordinates.

1 Introduction

An important coordinate system for representing mathematical and scientific functions (and data) is the spherical coordinate system. A spherical coordinate system (P, Θ, Φ) defines a radial distance P from the origin for each angle value of Θ and

*This research was supported by CNR-Italy, contract n.96.03845PS01

†Dept. of Computer Science, University of Bologna, Mura A.Zamboni 7, 40127 Bologna, Italy

‡Dept. of Mathematics, University of Bologna, P.zza Porta S.Donato 5, 40127 Bologna, Italy Phone +39 051354439, Fax +39 051354490, Email casciola@dm.unibo.it

Φ. For example, spheres, ellipsoids, toroids, superquadrics, spherical harmonics and many others shapes and functions are most conveniently represented in spherical coordinates. Methods for interpolating data over a spherical domain [8, 9, 10], such as methods for modelling [11], have received attention in Computer Aided Geometric Design.
In practice, a function is often presented as a set of values on the points of a grid. Its graph can then be approximated by a "grid surface" made up of straight-edge regions.
The hidden-line problem for plotting perspective views of objects is simpler when restricted to grid surfaces [4]. In literature several authors have proposed algorithms, some preferring speed of execution to the detriment of precision and generality [1, 2, 3, 5, 6, 14, 15, 16], while others prefers precision and generality [4, 13] though with some limitations on the point of view. The only scholar who deals with spherical coordinate functions, extending Anderson's proposal [4], is Suffern [13].
The proposals mentioned can be classified as those that solve the hidden-line problem using floating point arithmetic in the projection plane and those that work in the screen plane using integer arithmetic. The first to propose this second procedure method was Skala [12] for parallel projections. The present study proposes a new hidden-line algorithm for spherical coordinate functions that operates in the screen plane using almost exclusively integer arithmetic and is general, pixel exact and fast. Our proposal even goes beyond the limits of the Suffern [13] method: the function can be observed from any viewpoint and spherical coordinate functions with zero radius can be represented. For example our proposal allows for real time animations by rendering the grid surface from any viewpoint on a trajectory.
The proposed method, as is already noted in literature [4], exploits the property of being a function (single-valued) and continuous so that:

1. given the viewpoint, it is possible to determinate an occlusion compatible order of the facets (OCFO) in a simple manner;

2. when the facets are processed in accordance with an OCFO, the region obtained at each step is star-convex.

The main point of our proposal is to adapt Anderson's star-convex region to a discrete space. Then it is possible to store the boundary of the pixel invisibility star-convex region as a discrete function, TOP, in pseudopolar coordinates. The main advantage is that the pixel invisibility classification can be now performed with little effort: a given pixel (θ, ρ), lies within the invisibility region if $\rho < TOP(\theta)$.
Section 2 states the problem and defines the geometry of the view that has been adopted. Section 3 describes the fundamental ideas on which the proposed method is based. Section 4 provides a variant which is again theoretically seen to be pixel exact and computationally efficient. Next comes the analysis of the algorithm's

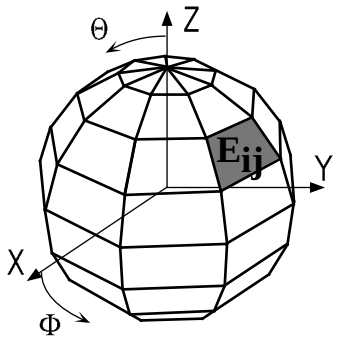

Figure 1: grid points and grid elements

complexity followed by a comparison with Suffern's proposal. They close with some implementation details.

2 Problem specification

Let

$$P = F(\Theta, \Phi) \quad with\ (\Theta, \Phi) \in [0, \pi] \times [0, 2\pi)$$

be a continuous function in spherical coordinates which has to be graphically represented on a raster display. In practice, the function is represented as a set of values on the points of a grid. Its graph can then be approximated by a grid surface consisting of straight-edge regions.

Let $\Theta_0, \ldots, \Theta_M$ be a strictly increasing sequence of latitude angles where $\Theta_0 = 0$ and $\Theta_M = \pi$, and let $\Phi_0, \ldots, \Phi_{N-1}$ be a strictly increasing sequence of longitude angles where $\Phi_0 = 0$ and $\Phi_{N-1} < 2\pi (\Phi_N = 2\pi)$ (the spacing need not be uniform).

Points in spherical coordinate of the form $(1, \Theta_i, \Phi_j)$ will be called **grid points**.

Grid points with $\Theta = 0$ $(i = 0)$ and $\Theta = \pi$ $(i = M)$ will be called north and south pole respectively and must be considered as two grid points only.

We call **grid element** E with index i, j, (E_{ij}), the planar patch (quadrilateral or triangular) bound by the edges $(1, \Theta_i, \Phi_j)$-$(1, \Theta_{i+1}, \Phi_j)$, $(1, \Theta_i, \Phi_{j+1})$-$(1, \Theta_{i+1}, \Phi_{j+1})$, $(1, \Theta_i, \Phi_j)$-$(1, \Theta_i, \Phi_{j+1})$ and $(1, \Theta_{i+1}, \Phi_j)$-$(1, \Theta_{i+1}, \Phi_{j+1})$. The image of a grid element under a function will be called a **facet**.

A **grid surface** is a continuous function $g : [0, \pi] \times [0, 2\pi) \rightarrow \mathbb{R}$ which has the given P values at the grid points and is linear between adjacent grid points.

We represent the function $g(\Theta, \Phi)$ on a raster device by drawing the image of line segments connecting adjacent grid points, under a perspective projection and window-viewport transformation.

To perform hidden-line removal we assume the following condition: the perspective projection of any facet (i.e. the image of a grid element under g) is bounded by

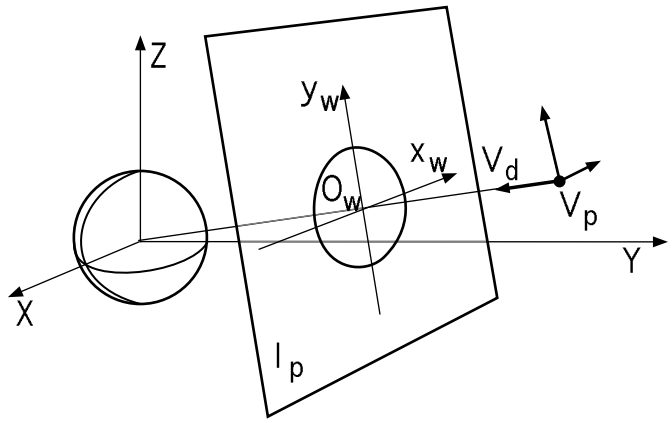

Figure 2: viewing geometry

the projection of the facet boundary. This condition means that a facet must never obscure its edges.
The following assumptions are made about the viewing geometry:

- the grid surface $g(\Theta, \Phi)$ is viewed from the viewing position, or center of projection $V_p = (X_p, Y_p, Z_p)$ in world Cartesian coordinates;
- the viewing direction is $V_d = (-X_p, -Y_p, -Z_p)$ so that it passes through the origin of the world spherical (P, Θ, Φ) and Cartesian (X, Y, Z) coordinate systems;
- the image plane, or projection plane I_p, is perpendicular to V_d.
- An x_w, y_w Cartesian system is defined on the projection plane with origin O_w obtained as the intersection between the plane and V_d, and y_w as the projection of the Z axis on the plane.

The viewing geometry described implies that the grid surface projection is contained in a circumference with center O_w and a proper radius r_w.

3 Basic algorithm

Anderson was able to exploit the following two geometric properties of the images of cartesian grid surfaces. Suffern realized that they was also valid for spherical grid surfaces.

1. An occlusion compatible facet order (OCFO) exists from the nearest to the furthest facet with respect to V_p. We can define this order to be an enumeration

of the facets $g(E_{i,j})_1, g(E_{i,j})_2, \ldots, g(E_{i,j})_{NM}$. Therefore if $g(E_{i,j})_l$ occludes $g(E_{i,j})_k$, then $g(E_{i,j})_l < g(E_{i,j})_k$.

2. When the facets are processed in an occlusion compatible order, the image-region is star-convex with respect to the origin O_w.

Since g is single-valued, the OCFO is not dependent on g and it is sufficient to define the above order for the $E_{i,j}$ grid elements. The OCFO depends only on the viewing position V_p, and can be obtained in a simple manner with low computational costs. There are many OCFO's; the one utilized in this paper has already been proposed by Suffern [13].

3.1 Pseudopolar coordinate transformation

Let r_s be the transformed r_w in accordance with the window-viewport transformation. Let C be the set of pixels inside the circle centered at the origin and radius r_s; for every pixel in C we define the transformation in pseudopolar coordinates as:

$$\begin{array}{ll} & \Gamma : C \rightarrow [0 \ldots m-1] \times [0, \infty) \\ & \Gamma : (x, y) \rightarrow (\theta = round(\frac{arctan(y/x)}{\Delta\theta}), \rho = \sqrt{x^2 + y^2}) \\ with & [0 \ldots m-1] \ \ integer \ \ interval \\ & [0, \infty) \ \ real \ \ interval \\ & \Delta\theta = 2\pi/m \end{array}$$

For our purposes it is important to utilize a system of pseudopolar coordinates where Γ is injective. Note that infinite choices exist for m so that this may be possible. Trivially a good example of m is given by:

$$m = 8 \ round\left(\frac{\sqrt{2}}{2} r_s\right) + 4$$

which corresponds to the number of pixels generated by Bresenham's algorithm [7]for the circumference of radius r_s.

3.2 A screen coordinate segment as a pseudopolar discrete function

Let $(x_A, y_A) - (x_B, y_B)$ be a segment in screen coordinates defined by its end-points. Let θ_A and θ_B be the θ transforms of (x_A, y_A) and (x_B, y_B) respectively according to Γ. We can define a discrete function relative to the segment as:

$$\begin{array}{l} G : [\theta_A \ldots \theta_B] \rightarrow [0, \infty) \\ G : \theta \rightarrow \| \ Q \ \|_2 \quad for \ \ \theta = \theta_A, \ldots, \theta_B \end{array}$$

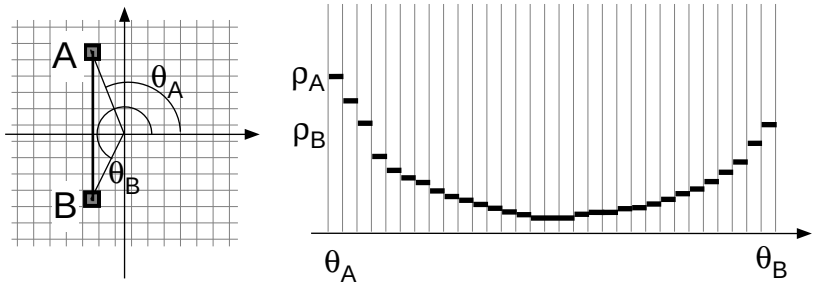

Figure 3: segment AB and its discrete function G

with $[\theta_A \ldots \theta_B]$ being an integer interval, Q being the intersection between segment $(x_A, y_A) - (x_B, y_B)$ and the straight line passing through the origin of slope $tan(\theta)$ (see Fig.3).

3.3 Proposed method

In algorithmic form the proposed method is summarized in the following steps:

Transform the grid surface into screen coordinates by means of perspective projection and window-viewport transformation;
Initialize the discrete function $TOP : [0 \ldots m-1] \rightarrow [0, \infty)$ into pseudopolar coordinates at 0 value;
For every grid element $(E_{i,j})_k$ $k = 1, \ldots, NM$, in accordance with an OCFO, do
    For every boundary segment of the $g(E_{i,j})_k$ projection not yet processed do
        For every pixel (x, y) (generated with the Bresenham algorithm) of the segment do
            transform the pixel in (θ, ρ) according to Γ;
            if $\rho < TOP(\theta)$ then draw the pixel;
        calculate the discrete function G of the segment;
    update the function TOP as the maximum point by point between the TOP itself and the G functions determined by the segments.

The exactness of this proposal is guaranteed by the fact that the pseudopolar coordinate transformation is injective, hence two pixels in screen coordinates may be differentiated for visibility by means of the TOP function. At each step the TOP function perfectly preserves the star-convex image-region that we call **pixel invisibility region**.

4 Improved method

Unfortunately, the hereby proposed method is not efficient. Every segment, particularly those close to the origin, produces a G function of great domain, even if made up of few pixels. The determination of this function is not competitive, in fact the value $\| Q \|_2$ for all $\theta_i \in [\theta_A \dots \theta_B]$ needs to be compute.
For such a proposal to become competitive it is necessary to simplify the determination of the G function in pseudopolar coordinates while continuing to precisely preserve the pixel invisibility region.
The basic idea of a less expensive G is to consider θ_i of each segment pixel, that is already known. For each of these θ_i a value of G is calculated, then the remaining values for G are determined in a simple manner.
With reference to the pixel invisibility region; let B_1 be the polygon defined by the sides of its boundary (see Fig.4A). Let B_2 be the polygon whose vertices are the centers of the pixels that are generated by applying the Bresenham algorithm to the sides of B_1(see Fig.4B).
Note that, for exactness of the algorithm, it is necessary to preserve only the pixels inside the invisibility region. Preserving the boundary pixels as well (pixels already processed and drawn), does not affect the exactness of the algorithm and leads to a reduction in complexity because there is no need to re-draw already drawn pixels. Then we consider B_2 as the boundary of the pixel invisibility region.
Definition: a closed polygon is called **star-convex** with respect to one of its internal points if the vertices, which are transformed into polar coordinates that originate in the defined internal point, have monotonic angles in a strict order.
Definition: a closed polygon is called **star-convex, but not strictly so**, with respect to one of its internal points if the vertices transformed into polar coordinates have monotonic angles that are not in strict order.
Note: if a polygon with floating point vertices is star-convex, the polygon defined in screen coordinates (that is, with vertices transformed in screen coordinates) will at most be star-convex, but not strictly so.
The region enclosed by B_2 might not be star-convex at all (see Fig.4B). However it is always possible to transform this region into a star-convex, but not strictly so. To do that one must consider, side by side, only the pixels whose centers are in angular order (see Fig.4C). Let B_3 be the polygon whose vertices are the centers of the pixels here considered. The pixels inside B_2 are also inside B_3; the pixels inside B_3 but not inside B_2 are boundary pixels for the region enclosed by B_2 (compare Fig.4A with Fig.4C). This is true because the following considerations are valid:
Definition: let us consider a straight line, r, in the plane XY, and two parallels equidistant from r. The distance between the two parallels is 1 on the X axis if the slope of r is ≥ 1, otherwise is 1 on the Y axis. The plane area delimited by the two parallel is then called **strip** associated with the straight line r (see Fig.5A).

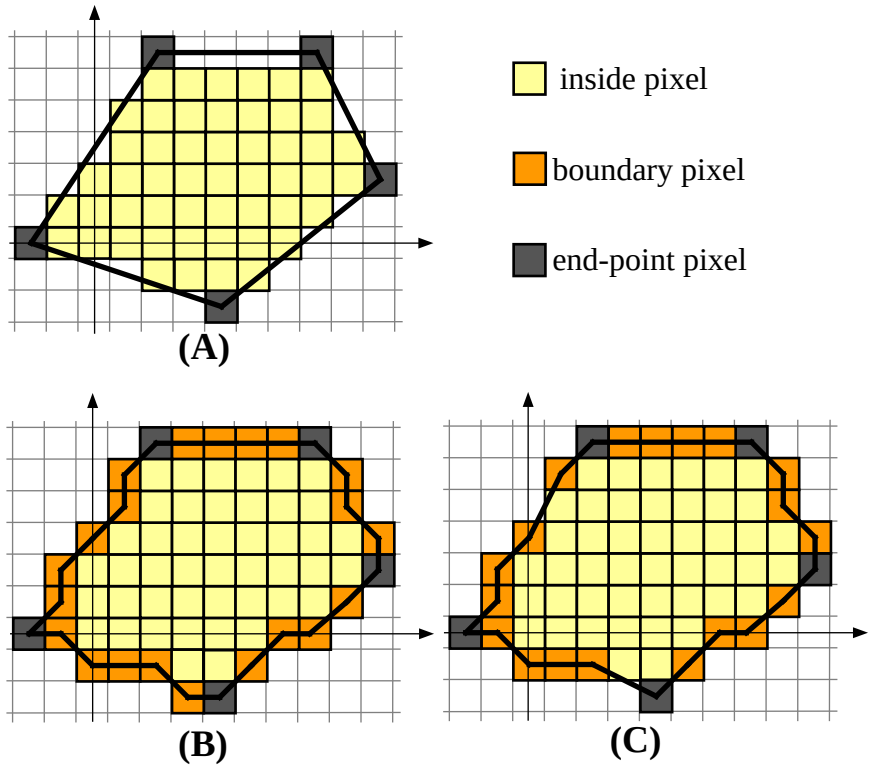

Figure 4: invisibility region with boundary B_1 (A), invisibility region with boundary B_2 (B), invisibility region with boundary B_3 (C)

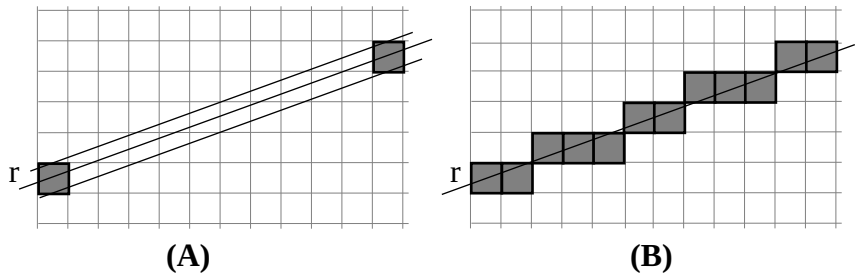

Figure 5: strip (A) and path (B) defined by line r through two integer coordinate points

Definition: given a straight line r in the plane XY passing through at least two integer coordinate points, the set of pixels generated by the Bresenham algorithm for r (see Fig.5B) is defined as the **path** associated with r.
Observation: given a straight line passing through at least two integer coordinate points, the pixels of the path are the only ones whose center is inside the half-open strip of the straight line.
Trivially, it turns out from the observation that any broken line, whose vertices are the pixels in the path, will be entirely contained in the half-open strip of the straight line r. Thus every pixel on one side with respect to the path (or the strip) will also be on the same side with respect to the broken line.
Given a segment $(x_A, y_A) - (x_B, y_B)$ in screen coordinates, let θ_A, θ_B be the transformed θ of the ends according to Γ. The new $G : [\theta_A \dots \theta_B] \to [0, \infty)$ is defined as follows: without loss of generality it is assumed that $\theta_A < \theta_B$. Using Bresenham's algorithm, the pixels (x_j, y_j) $j = 1, \dots, n$ of the segment path are generated, where $(x_1, y_1) \equiv (x_A, y_A)$ and $(x_n, y_n) \equiv (x_B, y_B)$; the transformed coordinates of (x_j, y_j) $j = 1, \dots, n$, are defined as (θ_j, ρ_j) $j = 1, \dots, n$, according to Γ. Therefore, we have the following:

$$
\begin{aligned}
&G(\theta_1) = \rho_1 \\
&G(\theta_n) = \rho_n \\
&G(\theta_j) = \begin{cases} \rho_j & if \;\; \theta_{j-1} < \theta_j \\ \max(\rho_{j-1}, \rho_j) & if \;\; \theta_{j-1} = \theta_j \\ nothing & if \;\; \theta_{j-1} > \theta_j \end{cases} \\
&for \;\; j = 2, \dots, n-1 \;\; and \;\; \theta_j \in [\theta_1 \dots \theta_n]
\end{aligned}
$$

Let θ_k, $k = 1, \dots, l$ $(l \leq n)$ be the integers to which a value for G has been attributed; G is defined in the remaining $\theta \in [\theta_1 \dots \theta_n]$ as the following:

$$
\begin{aligned}
&G(\theta) = \min(\rho_K, \rho_{K+1}) \\
&\forall \theta \in (\theta_K \dots \theta_{K+1}) \;\; K = 1, \dots, l-1
\end{aligned}
$$

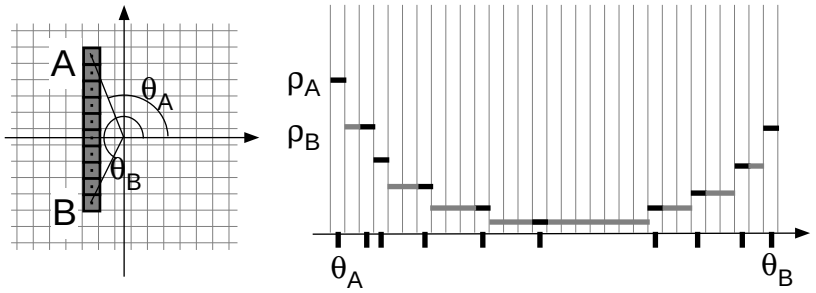

Figure 6: segment AB of pixels and its discrete function G; only the bold θ values are obtained by transforming pixels

In reference to Fig.4, letB_4 be the boundary of the invisibility region defined by the discrete function TOP obtained by starting from the new G functions. It is necessary to prove that B_3 and B_4 have the same inside pixels (see Fig.7 and compare with Fig.4C).
Main Result: boundary regions B_3 and B_4 have the same inside pixels where, by the term "inside pixel" we mean that its center is inside the region.
proof: proceed by proving the statement for every circular sector $\theta_K - \theta_{K+1} - \min(\rho_K, \rho_{K+1})$ and triangle $(0,0) - (\theta_K, \rho_K) - (\theta_{K+1}, \rho_{K+1})$.
Note that, because of how they are constructed, the circular sector is always contained in the triangle except where $\rho_K = \rho_{K+1}$, when the opposite occurs. Let us proceed by distinguishing the two following cases: points between (θ_K, ρ_K) and $(\theta_{K+1}, \rho_{K+1})$ which G must completed, either are or are not the transforms of adjacent pixels of the segment.
proof of the adjacent pixels case: trivially, whatever ρ_K and ρ_{K+1} are, the inside pixels at the circular sector $\theta_K - \theta_{K+1} - \min(\rho_K, \rho_{K+1})$ are the same as those of the vertices triangle $(0,0) - (\theta_K, \rho_K) - (\theta_{K+1}, \rho_{K+1})$ (see Fig.8A).
proof of the non-adjacent pixels case: this occurs for approximately radial pixel segments from which $\rho_K \neq \rho_{K+1}$ and the triangle $(0,0) - (\theta_K, \rho_K) - (\theta_{K+1}, \rho_{K+1})$ will contain the circular sector $\theta_K - \theta_{K+1} - \min(\rho_K, \rho_{K+1})$ (see Fig.8B). It will be sufficient to prove that the difference region does not contain pixels.
This is the aim of the following theorem and corollary.
theorem: given a segment of integer coordinate end-points (x_1, y_1) and (x_n, y_n) let θ_1 and θ_n be the corresponding angles. Without loss of generality let $((y_n - y_1)/(x_n - x_1)) > 1$ and $((x_n y_1 - x_1 y_n)/((x_n - x_1)) < 0$ (that is, the origin is to the left of the segment). Let (x_i, y_i) $i = 1, ..., n$ be the integer coordinates generated by the Bresenham algorithm for the pixels of the segment. Let (x_j, y_j) $j = l{+}1, \ldots, k{-}1$ be such that $\theta_j < \theta_l$ and $\theta_k > \theta_l$. Then pixels (x, y_j) with $x < x_j$ for $j = l{+}1, \ldots, k{-}1$ have the angle $\theta > \theta_k$.

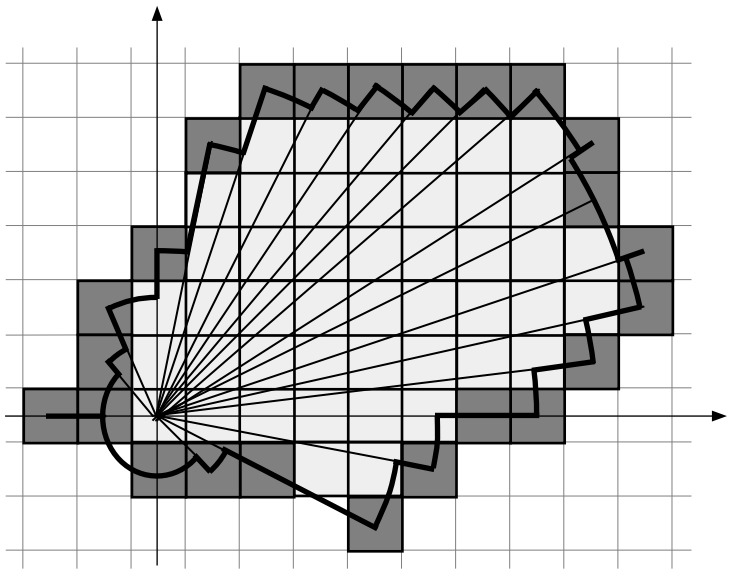

Figure 7: invisibility region with boundary B_4

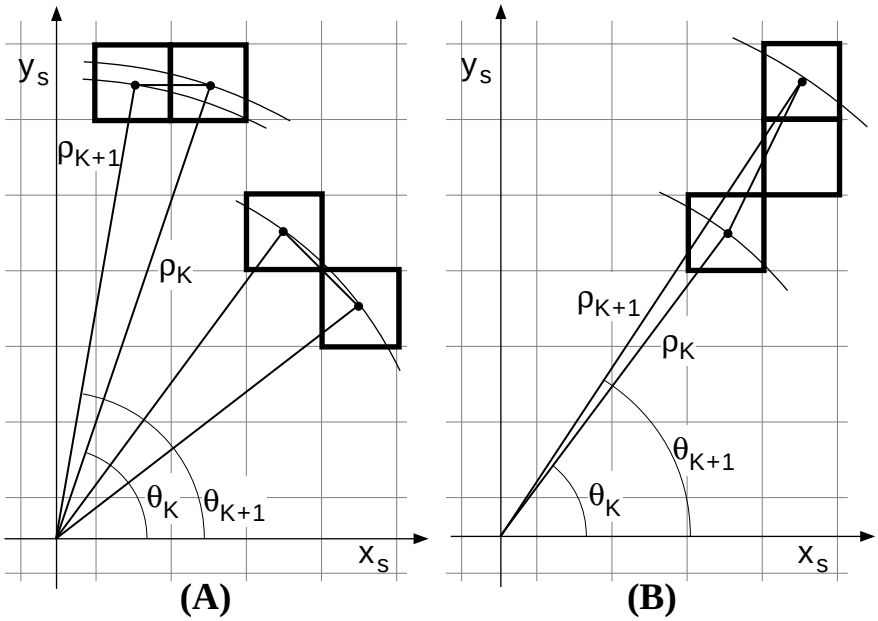

Figure 8: case of adjacent pixels (A) and non-adjacent (B)

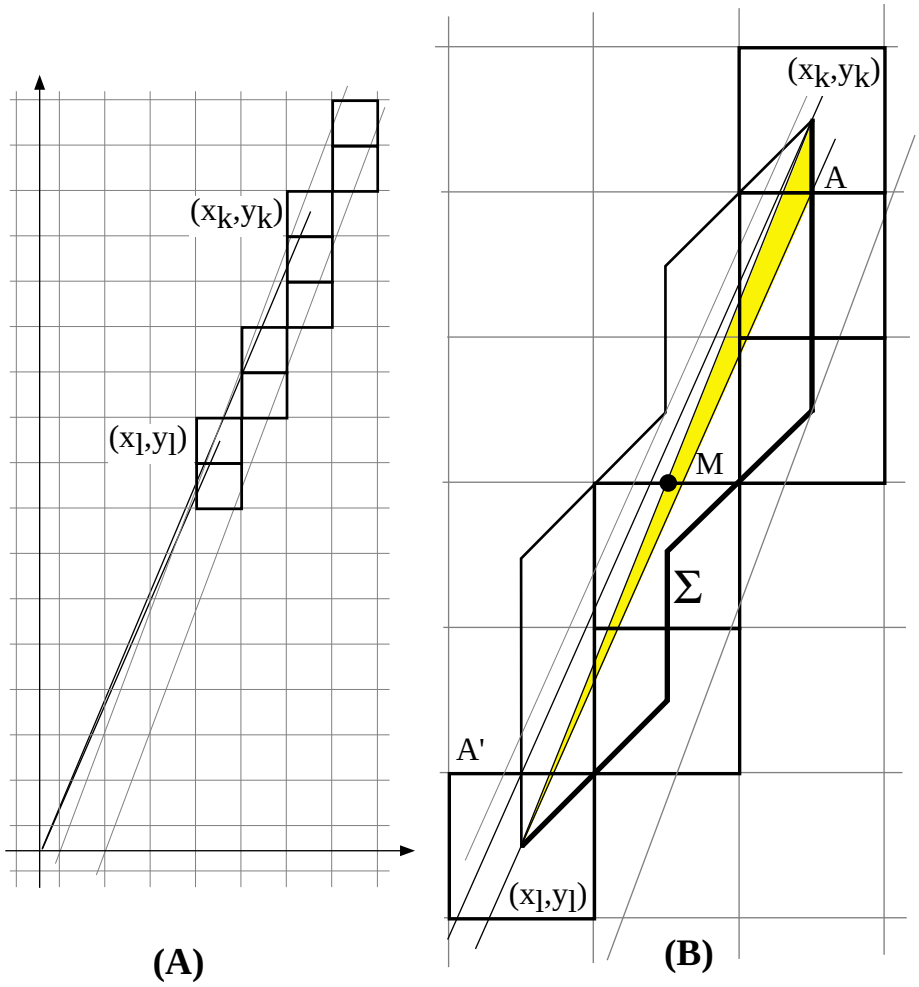

Figure 9: segment with endpoints $(4,8)$ $(7,16)$. (B) is a magnification of (A)

proof: consider the polygon Σ of vertices (x_j, y_j) $j = l, \ldots, k$. Since these vertices belong to the path of segment $(x_1, y_1) - (x_n, y_n)$ they are, due to observation previously made, inside the segment strip, therefore polygon Σ has no inside pixels. From the hypothesis the half-line $(0,0) - (x_l, y_l)$ will intersect polygon Σ only at one other point, A, beyond (x_l, y_l). It follows that triangle $(x_l, y_l) - A - (x_k, y_k)$, being in Σ, has no inside pixels. Now let M be the mid-point of segment $(x_l, y_l) - (x_k, y_k)$. It follows that triangle $(x_l, y_l) - A^{'} - (x_k, y_k)$, symmetrical with respect to M of $(x_k, y_k) - A - (x_l, y_l)$, has no inside pixels either. This proves the theorem. □

corollary: Based on the hypothesis of the previous theorem let (x_j, y_j) $j = l + 1, \ldots, k-1$ be such that $\theta_j < \theta_l$ or $\theta_j > \theta_n$ and $\theta_k > \theta_l$. Pixels (x, y_j) with $x < x_j$ for $j = l+1, \ldots, k-1$ then have angle $\theta > \theta_k$.

proof: the proof is perfectly analogous to that of the previous theorem. Here the

polygon Σ has vertices (x_l, y_l), (x_k, y_k) and only (x_j, y_j) $j = l+1, \ldots, k-1$ so that $\theta_j < \theta_l$. $\square$

5 Algorithm analysis and comparison

5.1 Space complexity

The memory allocation consists of two arrays of 32 bit unsigned integers for the function TOP in pseudopolar coordinates and a copy of it respectively. Dimension m of these arrays is given by:

$$m = \frac{circle}{\Delta\theta}$$

where $\Delta\theta$ and $circle$ depend on a chosen pseudoangle function. Quantity $\Delta\theta$ is such that, in the Γ transformation into pseudopolar coordinates, there are distinct integers θ_i at every pixel on the maximum circumference. This allocation depends only on the viewport-size. This is unlike the one proposed by Suffern [13] in which the boundary of the invisibility region is stored on a list of coordinate floating point items dynamically allocated. This allocation depends on: the represented surface, the grid and on the viewpoint.

5.2 Time complexity

The complexity of the proposed algorithm may be structured in:

$$C_{TOT} = C_0(facets) + C_1(facets, viewportsize) + C_2(facets, viewportsize)$$

where

$C_0(facets)$ summarizes: the relative complexity in calculating the function $P = F(\Theta, \Phi)$, the perspective projection and the transformation into screen coordinates of the grid points, determination of an OCFO, and a simple visibility test of the facets. All these steps lead to a complexity that is a linear function of the single facets;

$C_1(facets, viewportsize)$ considers: the complexity of the transformation into pseudopolar coordinates of the invisibility test, the possible drawing and determination of belonging to the star-convex, but not strictly so, boundary, of every pixel of the sides of the facets. It is a linear function in the number of processed pixels so that:

$$C_1(facets, viewportsize) = const \cdot Nprocessedpixels(facets, viewportsize)$$

	$ViewportSize$											$Suffern$
$No.of$	400^2		500^2		600^2		800^2		1000^2		T_0	T_{TOT}
$facets$	T_{TOT}	T_1	T_{TOT}	T_1	T_{TOT}	T_1	T_{TOT}	T_1	T_{TOT}	T_1		
25^2	0.10	0.03	0.12	0.04	0.14	0.05	0.19	0.06	0.23	0.08	0.02	0.05
50^2	0.25	0.06	0.28	0.07	0.32	0.09	0.40	0.12	0.47	0.14	0.08	0.20
100^2	0.66	0.12	0.72	0.15	0.80	0.18	0.95	0.23	1.10	0.29	0.33	0.86
150^2	1.26	0.19	1.35	0.23	1.47	0.27	1.68	0.34	1.90	0.43	0.73	2.16
200^2	2.05	0.29	2.18	0.33	2.32	0.39	2.62	0.49	2.88	0.61	1.29	4.28
250^2	3.08	0.37	3.27	0.45	3.43	0.51	3.75	0.61	4.10	0.74	2.03	7.39
300^2	4.31	0.50	4.48	0.55	4.71	0.66	5.10	0.76	5.50	0.90	2.93	11.64

Table 1: Execution times [sec] of Suffern and our algorithms for the sphere surface

where $Nprocessedpixels$ is a function that is at most linear in the facets and less than linear in the viewport-size. When the facets or the the viewport-size quadruplicates for a smooth function, then the pixels double. For an irregular function when quadruplicating the facets, the pixels quadruplicate, but when quadruplicating the viewport-size, they double;

$C_2(facets, viewportsize)$ considers the completion phase of the G function and the updating of the TOP function. This turns out to be linear in the worst case both in the facets and in the viewport-size.

Therefore C_{TOT} is linear in the worst cases with respect to the facets and less than linear with respect to the viewport-size.

5.3 Performances

The proposed method was implemented in a C language program on a Silicon Graphics Indigo, with a XZ4000 processor running at 100MHz. We also developed versions for PC computers, using both Pascal and C language. The program has been tested on many surfaces. Performance results of our method and of Suffern's algorithm are shown in table 1 and 2 for the sphere and random functions respectively. These were generated on uniform grids varying fineness on the interval $[0, \pi] \times [0, 2\pi)$ and varying viewport-sizes (applied only to our method). T_{TOT}, T_0 and T_1 are the execution times respectively for C_{TOT}, C_0 and C_1. T_2 can be evaluated by $T_2 = T_{TOT} - T_0 - T_1$. The execution times T_{TOT}, graphically represented as functions of the facets, are shown in Fig.10 and Fig.11.
The grid surfaces, with values on 101×101 grid points (100×100 number of facets) from the spherical view-point $(100, 1, 1)$ are shown in Fig.12.
Fig.13 and Fig.14 show the graph of function $Nprocessedpixels(facets, viewportSize)$ for the sphere and random surfaces, which respectively represent the most and the

	$ViewportSize$											$Suffern$
$No.of$	400^2		500^2		600^2		800^2		1000^2		T_0	T_{TOT}
$facets$	T_{TOT}	T_1	T_{TOT}	T_1	T_{TOT}	T_1	T_{TOT}	T_1	T_{TOT}	T_1		
25^2	0.16	0.07	0.19	0.09	0.22	0.11	0.28	0.14	0.35	0.18	0.02	0.63
50^2	0.52	0.30	0.62	0.37	0.72	0.44	0.93	0.58	1.13	0.72	0.09	0.33
100^2	1.78	1.14	2.12	1.40	2.45	1.67	3.14	2.21	3.81	2.73	0.34	2.19
150^2	3.86	2.59	4.59	3.21	5.31	3.81	6.75	5.02	8.20	6.23	0.76	6.71
200^2	6.65	4.57	7.88	5.65	9.10	6.70	11.54	8.83	13.89	10.92	1.34	15.45
250^2	10.33	7.17	12.16	8.79	14.04	10.44	17.65	13.63	21.32	16.88	2.1	32.23
300^2	14.72	10.36	17.30	12.68	19.98	15.10	25.45	19.70	30.55	24.41	3.04	56.17

Table 2: Execution times [sec] of Suffern and our algorithms for the random surface

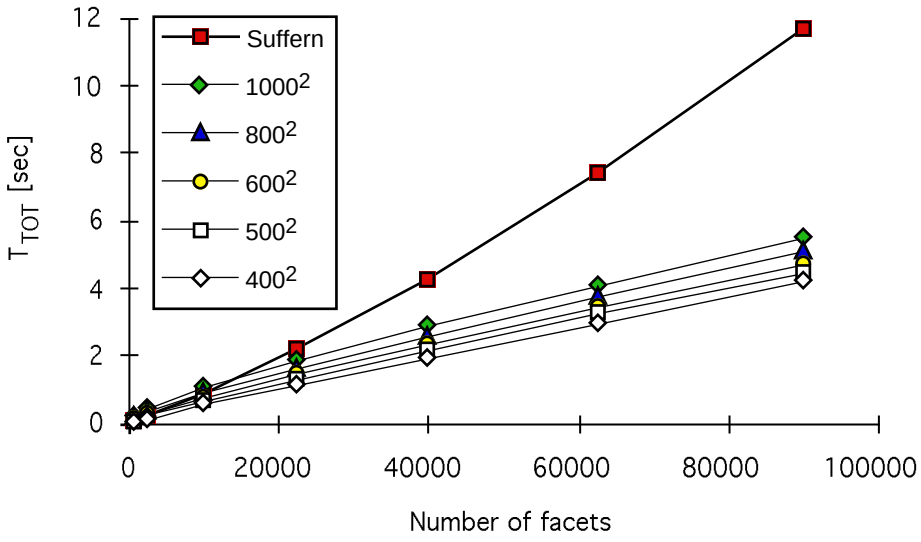

Figure 10: total execution times of Suffern and our algorithms as functions of the number of facets for the sphere surface

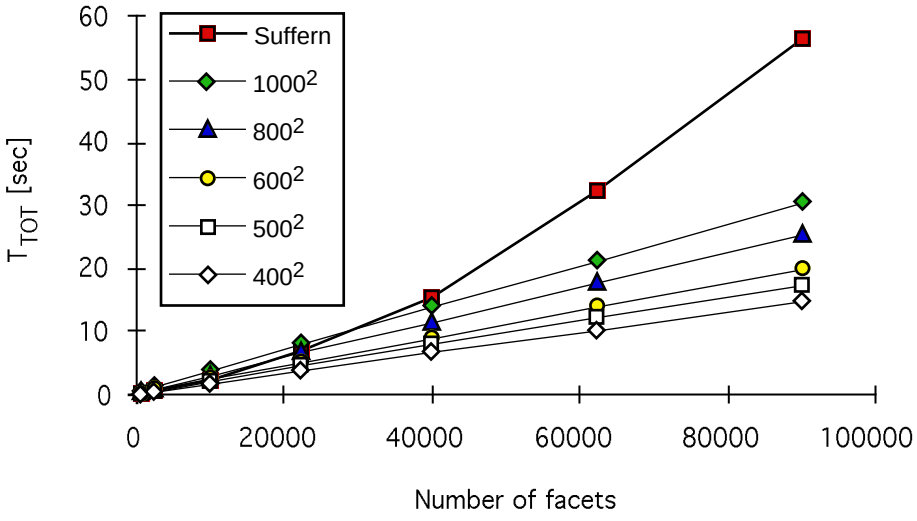

Figure 11: total execution times of Suffern and our algorithms as functions of the number of facets for the random surface

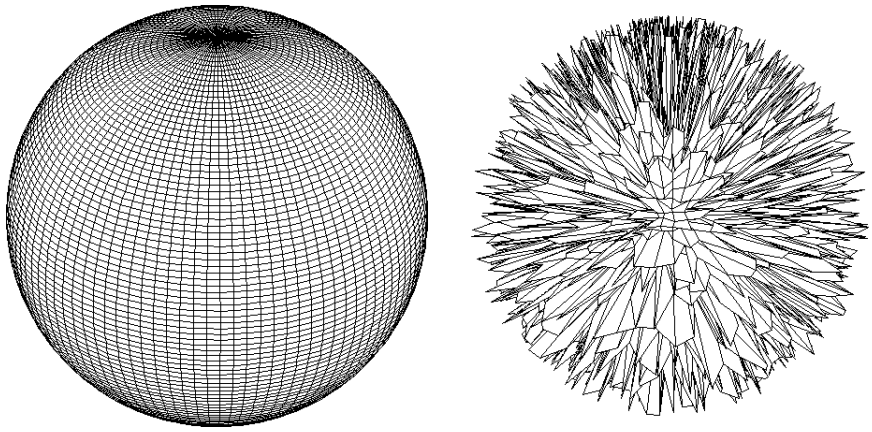

Figure 12: Plot of the functions $F(\Theta, \Phi) = 50$ and $F(\Theta, \Phi) = 25 + 2$ (16 *random*) where *random* is a random variable uniformly distributed in $[0, 1]$

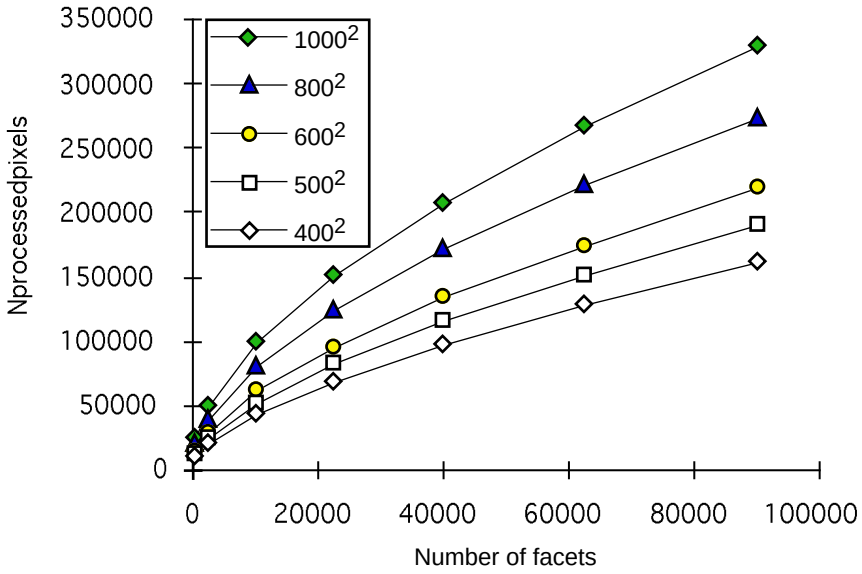

Figure 13: *Nprocessedpixels* as functions of the number of facets for the sphere surface

least regular function.
These results confirm that the algorithms, as implemented, are approximately linear in time [13] and linear in time for the worst case (ours) as functions of a number of facets. Actually, for most of the functions, as shown in the examples of Fig.15 and Fig.16, the graph of the execution time is more like that of the sphere than the random surface.
All the tests carried out show that our proposal is highly competitive for fine grids, proposing itself as a valid tool for high quality rendering at low cost. For example, see Fig.17 where it is necessary to utilize a grid 360×720 in order to begin to recognize the shape of lands above sea-level. The execution time for this example is $13.43sec$ for our algorithm and $43.13sec$ for the Suffern algorithm with a viewportsize 800×800.

6 Some computational details

6.1 Simple visibility test

The proposed method makes use of a preliminary simple visibility test for all the facets that aim to improve performance. Consequently, the facets that are negative on testing do not require further elaboration. The test adopted is quite similar to that proposed by Anderson for Cartesian coordinates, and taken up by Suffern for spherical coordinates. Our version differs from the latter by working in screen

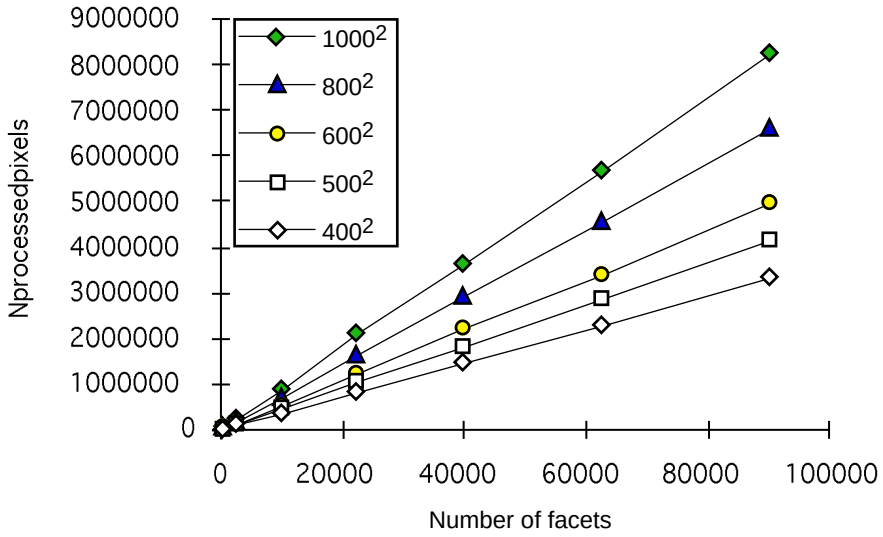

Figure 14: *Nprocessedpixels* as functions of the number of facets for the random surface

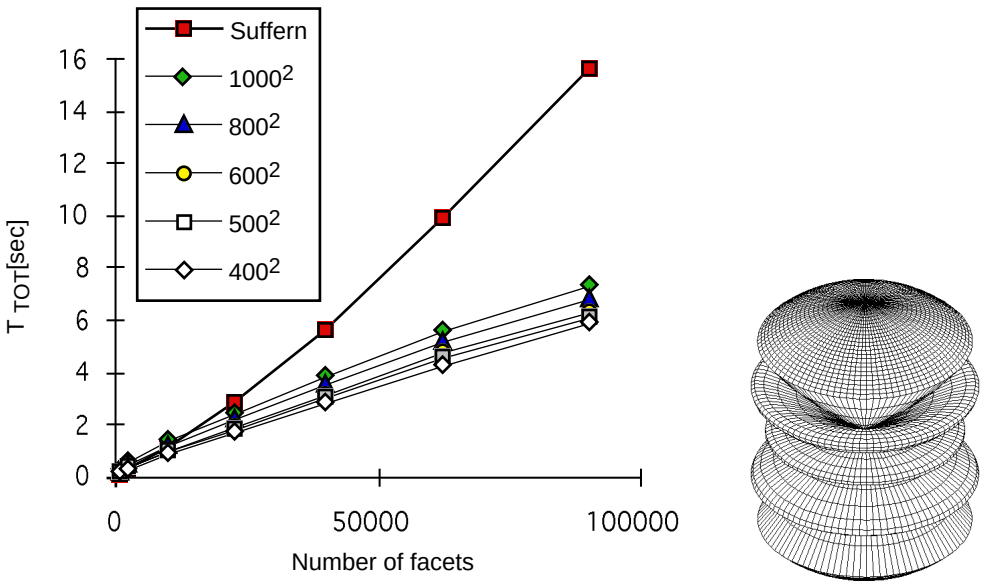

Figure 15: Total execution times for the surface $F(\Theta, \Phi) = abs((sin(2.5\pi(cos(g(\Theta)) - 1)))/(5.0sin(\pi/2(cos(g(\Theta)) - 1))))$ *with* $g(\Theta) = \Theta$ *if* $0.8 < \Theta < (\pi - 0.8)$ *or* $g(\Theta) = 0.8$ *otherwise*, shown on the right

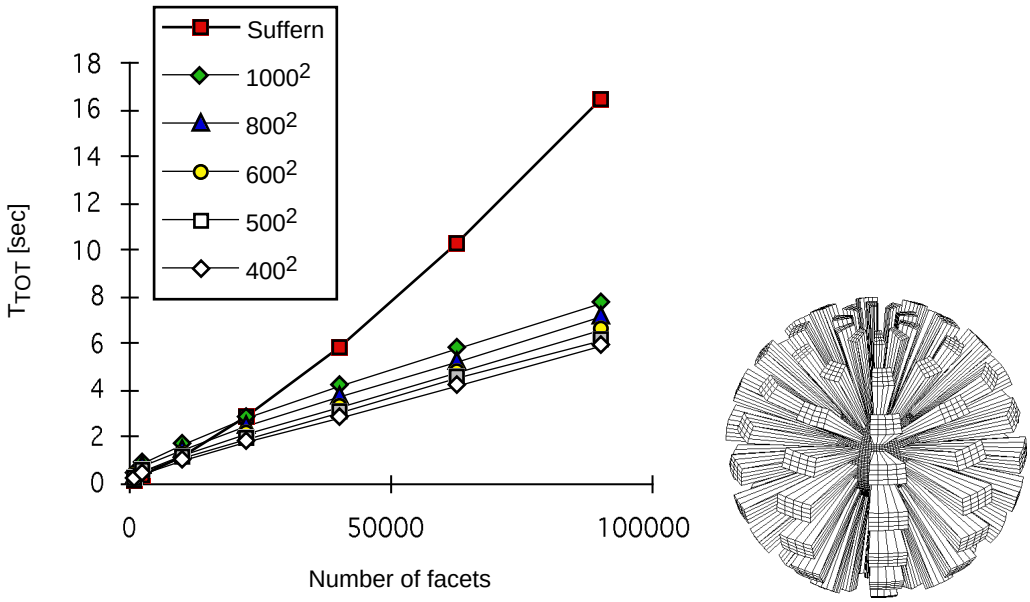

Figure 16: Total execution time for the surface $F(\Theta,\Phi) = 0.6$ *if* $(int)(\Theta/\Delta\Theta)$ *and* $(int)(\Phi/\Delta\Phi)$ *are both odd and* $F(\Theta,\Phi) = 0.2$ *otherwise with* $\Delta\Phi = \Delta\Theta = \pi/20$, *south pole* $= 0.2$ *or north pole* $= 0.6$, shown on the right

coordinates (integer) and also by managing the cases of projected facets with vertices in the origin or sides passing through the origin.

6.2 Pseudopolar transformation

The algorithm does not need to know the exact value of the angle or radius in polar coordinates corresponding to a pixel, but only the angular and radial order of the pixels. Thus a transformation Γ, in which the pseudoangular and pseudoradial functions are strictly increasing both in angle and radius respectively, is sufficient. The simplest pseudoradius ρ for a pixel with screen coordinates (x, y) is given by:

$$\rho = x^2 + y^2$$

A continuous pseudoangle function, less computationally expensive compared to $\alpha = arctan(y/x)$, may be the following:

$$\alpha(x,y) = \begin{cases} 1 - \frac{x}{x+y} & if \quad x > 0, \quad y \geq 0 \\ 1 + \frac{x}{x-y} & if \quad x \leq 0, \quad y > 0 \\ 3 - \frac{x}{x+y} & if \quad x < 0, \quad y \leq 0 \\ 3 + \frac{x}{x-y} & if \quad x \geq 0, \quad y < 0 \end{cases}$$

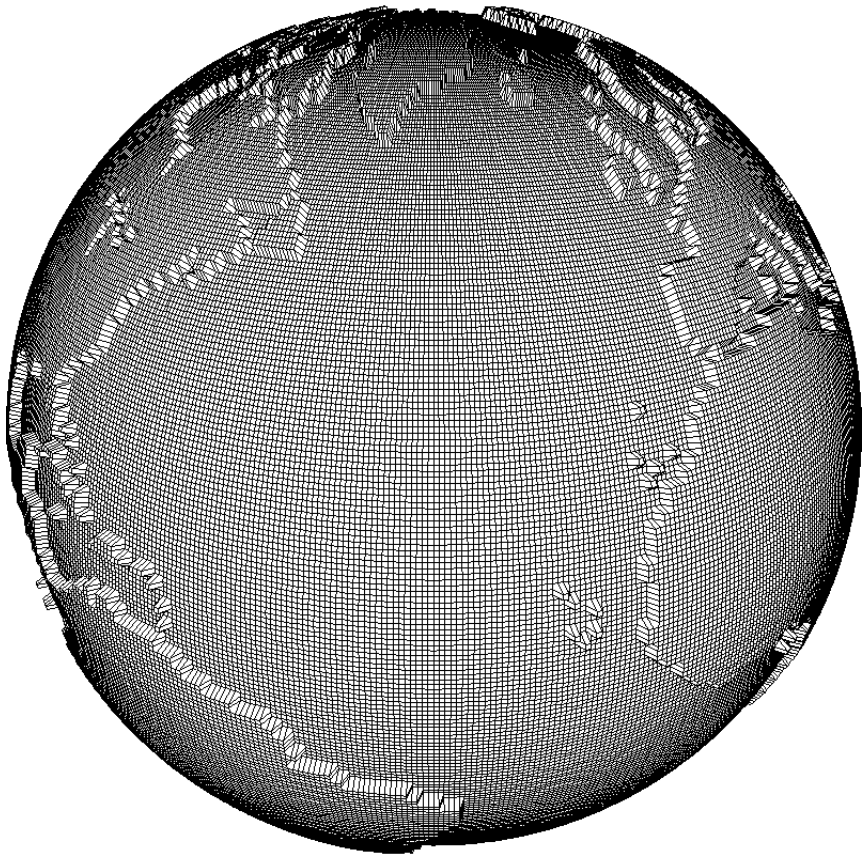

Figure 17: plot of the earth with grid 360×720

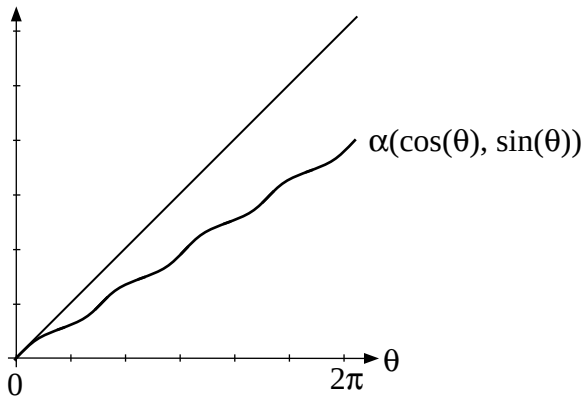

Figure 18: graph of function α

For our purposes, a discrete pseudoangle function is necessary so that pixels with the same ρ must have different pseudoangles. In the definition of Γ the function $arctan(y/x) : \mathbb{R}^2 \to \mathbb{R}$ restricted to a circumference, or thought of as a function of the angle θ, is the line of slope 1. The function $\alpha(x, y)$ given above, has the graph shown in Fig.18 for $0 \leq \theta < 2\pi$.
In the definition given of Γ it is possible to define m simply as the number of pixels on boundary C in order to satisfy the injectivity. For the function just given, the definition of m is more complex. Note that the function $\alpha(cos\theta, sen\theta)$ for $\theta \in [0, \pi/2)$ has its minimum slope in $\pi/4$. Here the function has a flex with the tangent line of equation $\alpha = ((4+\pi)/8 - \theta)/2$ which has values in the interval $[(4-\pi)/8, (4+\pi)/8]$ or range $\pi/4$. The sought after m will then be given by:

$$m = round\left(\frac{4\left(8\ round\ \left(\frac{\sqrt{2}}{2}r_s\right) + 4\right)}{\pi}\right)$$

6.3 Screen segment through the origin of the pseudopolar coordinates system

This case is frequently found due to the working in screen coordinates. Segments, which in the projection plane are only close to the origin, once transformed into screen coordinates will, in fact, pass through the origin. In particular, this case is also found in the conditions of $P = 0$ or V_p aligned with latitudinal and/or longitudinal edges.
The proposed method, which runs a pixel invisibility region that is star-convex, but not strictly so, manages to control this case in a simple manner. If a segment has an end-point in the origin, it will be radial with the pseudoangle determined by the

other end-point. In the updating phase it will, at most, modify a single value of the TOP function. The case of a segment aligned with the origin is managed in the same way, that is, the two end-points have the same pseudoangle.
Where the segment passes through the origin, it is broken into two segments, each one with an end-point in the origin. The segments are managed as in the above-mentioned case.

7 Conclusions

The proposed method is: general, pixel exact, and faster than the known algorithm on fine grids. For most of the functions the complexity is less than linear both in the facets and in the viewport-size.
The idea of the method has also been applied for functions in Cartesian coordinates. This implementation is more involved with respect to the presented one. The main complication is that two discrete functions are necessary, TOP and $BOTTOM$, to store the boundary of the invisibility region. Furthermore, the origin of the pseudopolar coordinate system, V_z (the intersection of the projection plane and the z-direction line passing through V_p), may be so far from the boundary of the invisibility region to require floating point arithmetic. V_z may also be out of floating point range or undefined (two vanishing points). In these cases we need to apply the two array mask algorithm that is correct and pixel exact [1, 3]. Moreover, our proposal outperforms the Anderson's proposal.
The idea is also successfully applicable for other types of representations, such as contours lines.

References

[1] Alvisi, L., Casciola, G. On the two array mask hidden-line algorithm. *Computer & Graphics* 13, 2 (1989), 193-206.

[2] Alvisi, L., Casciola, G. TAM rivisitato: un metodo rapido ed esatto per la rappresentazione prospettica di superfici, *Pixel*, 10 (1988), 15-25.

[3] Alvisi, L., Casciola, G. *Two and four array mask algorithms in practice.* Technical report, Department of Mathematics, University of Bologna, (February 1988).

[4] Anderson, D. P. Hidden line elimination in projected grid surfaces. *ACM Transaction on Graphics* 1, 4 (Oct. 1982), 274-288.

[5] Butland, J. Surface drawing made simple. *Computer Aided Design*, 11 (1979), 19-22.

[6] Chen Wang, S., Staudhammer, J. Visibility determination on projected grid surfaces. *IEEE Computer Graphics & Applications*, (July 1990), 36-43.

[7] Foley, J., van Dam, A., Feiner, S., Huges, J. *Computer Graphics, Principles and Practice (2 ed.)*. Addison Wesley, (1990).

[8] Lawson, D.L. C^1 Surface Interpolation for Data on a Sphere. *Rocky mountain J. Math.*, 14 (1984), 177-202.

[9] Nielson, G.M., Ramaraj, R. Interpolation Over a Sphere Based Upon a Minimum Norm Network. *Computer Aided Geometric Design*, 4, (1987), 41-57.

[10] Renka, R.J. Interpolation of Data on the Surface of a Sphere. *ACM Trans. Math. Softw.*, 10, (1984), 417-436.

[11] Sánchez-Reyes, J. Single-Valued Surface in Spherical Coordinates. *Computer Aided Geometric Design*, 11, (1994), 491-517.

[12] Skala, V. *An Interesting Modification to the Bresenham Algorithm for Hidden-Line Solution.* NATO ASI Series F17 in Earnshaw, R.E., (Ed.), Foundamental Algorithm For Computer Graphics, Springer-Verlag, New York, (1985), 343-347.

[13] Suffern, K. Perspective views of polar coordinate functions. *Computer & Graphics* 12, 3/4 (1988), 515-524.

[14] Watkins, S.L. Masked three-dimensional plot program with rotations. *Communication of ACM* 17, 9 (1974), 520-523.

[15] Williamson, H. Algorithm 420 - Hidden-line plotting program [j6]. *Communication of ACM* 15, 2 (Feb. 1972), 100-103.

[16] Wright, T.J. A two space solution to the hidden line problem for plotting function of two variables. *IEEE Transaction on Computer*, (Jan.1973), 28-33.