POLITECNICO DI TORINO
Repository ISTITUZIONALE

Occlusion points identification algorithm

Original
Occlusion points identification algorithm / DE VIVO, F., Battipede, M., Gili, P.. - In: COMPUTER AIDED DESIGN. - ISSN 0010-4485. - ELETTRONICO. - 91:(2017), pp. 75-83. [10.1016/j.cad.2017.06.005]

Availability:
This version is available at: 11583/2677046 since: 2017-09-22T15:51:33Z

Publisher:
Elsevier

Published
DOI:10.1016/j.cad.2017.06.005

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 September 2026

Occlusion points identification algorithm

F. De Vivo[a,*], M. Battipede[a], P. Gili[a]

[a] *Polytechnic of Turin, Department of Mechanical and Aerospace Engineering, Corso Duca degli Abruzzi, 24 - 10129 Torino, ITALY*

Abstract

In this paper a very simple and efficient algorithm is proposed, to calculate the invisible regions of a scene, or shadowed side of a body, when it is observed from a pre-set point. This is done by applying a deterministic numerical procedure to the portion of scene in the field of view, after having been projected in the observer reference frame. The great advantage of this approach is its generality and suitability for a wide number of applications. They span from real time renderings, to the simulation of different types of light sources, such as diffused or collimated, or simply to calculate the effective visible surface for a camera mounted on board of an aircraft, in order to optimise its trajectory if remote sensing or aerial mapping task should be carried out. Optimising the trajectory, by minimising at any time the occluded surface, is also a powerful solution for a search and rescue mission, because a wider area in a shorter time can be observed, particularly in situations where the time is a critical parameter, such as, during a forest fire or in case of avalanches. For its simplicity of implementation, the algorithm is suitable for real time applications, providing an extremely accurate solution in a fraction of a millisecond. In this paper, the algorithm has been tested by calculating the occluded regions of a very complex mountainous scenario, seen from a gimbal-camera mounted on board of a flying platform.

Keywords: Occlusion points, rendering, camera, remote sensing, light source, aircraft

Nomenclature

$f(\cdot)$	Homogeneous Transformation (HT)
$\mathbf{P^I}$	Vector of point coordinates in the IRF
$\mathbf{T}_a^b$	HT matrix from reference frame a to b
$\mathbf{R}_a^b$	Rotational matrix from RF a to b
$\mathbf{d}_a^b$	Translational vector from RF a to b
$\mathbf{I}_{P_{ij}}$	Image projection of the point P_{ij}
$\mathbf{C}_{ij}$	Vector of projected values $x^c_{1:n_r j}$
$\mathbf{C}_{xy}$	Image vector of projected values $x^c_{1:n_r j}$
$\mathbf{I}_{1/0}$	Vector of boolean values
$\mathbf{i}_Q$	Vector of column indexes
x^I, y^I, z^I	Inertial reference frame coordinates
x^c, y^c, z^c	Camera reference frame coordinates
x^b, y^b, z^b	Body reference frame coordinates
x^g, y^g, z^g	Gimbal reference frame coordinates
f_c	Focal length
i_k, j_k	Matrix row and column indexes
$\delta_{p_{x,y}}$	Pixel dimensions
ϕ, θ, ψ	Angles of roll, pith, yaw
θ_{el}, ψ_{az}	Gimbal elevation and azimuth angles
$\theta_{FOV_{x,y}}$	Vertical and horizontal FOV angles

1. Introduction

1.1. Overview

One of the fundamental research problems in computer graphics is related to accurately represent a complex model. In order to do this, it is imperative that those surfaces normally invisible from a certain point, are also invisible in the computer generated image. This problem is called the *visible surface problem* or the *hidden-surface problem* [1]. Over time, different algorithms have been developed to takle the visibility problem: ray tracing [2, 3], beam tracing, cone tracing, frustum casting [4] and Binary Space Partitioning (BSP) trees [5–7]. These methods share the common idea of sweeping the scene in the direction defined by a certain set of rays. While the previous algorithms are object space approaches, a widely used and simple algorithm for visible surface determination in image space is the z-buffer.

*Corresponding author

Email addresses: francesco.devivo@polito.it (F. De Vivo), manuela.battipede@polito.it (M. Battipede), piero.gili@polito.it (P. Gili)

1.2. Object space approaches

Ray tracing, unlike the BSP tree, determines visible surfaces in an image operating pixel-by-pixel rather than primitive-by-primitive, making this algorithm relatively slow for scenes with large objects in screen space. Unlike the earlier ray-object intersection pseudocode, where the intersections where verified looping over all the objects [8], in advanced algorithms, they can be computed in sub-linear time using *divide* and *conquer* techniques [9]. A different solution in most intersection acceleration schemes is to compute the intersection of a ray with a bounding box. This differs from conventional intersection tests because the knowledge of the exact position in which the ray hits the box is not required. A similar approach is the hierarchical bounding box [10]. The beam tracing algorithm, designed by Heckbert and Hanrahan [11], has the aim to overcome some drawbacks related to the ray tracing technique, casting a pyramid (beam) containing many rays rather than using a single ray at a time. The problem of this algorithm is that the beams might become rather complex and its implementation is difficult. The cone tracing, proposed by Amanatides [12], unlike the previous two techniques, traces a cone of rays at a time instead of a beam or a single ray. In contrast to the beam tracing, the algorithm does not determine precise boundaries of visibility changes. An efficient visible surface algorithm was developed by Naylor [13] for rendering polygonal scenes using a BSP tree. This tree is obtained by projecting the visible scene polygons and it is used to cut-out the invisible ones. The drawback of this method is that it requires that the whole scene is represented using the BSP tree, posing a significant problem for large and dynamic scenes. A solution for the estimation of the occluded region based on the calculation of the gradient value of the Digital Elevation Map (DEM) has been proposed by Oliveira [14]. In this case, an abrupt variation of the gradient defines the starting point of the occluded region.

1.3. Image space approaches

The z-buffer technique, for each pixel on the display, keeps both a record of the depth of the closest object in the scene and a record of the object intensity value that should be displayed. When a new polygon is processed, a z-value and intensity are calculated for each pixel inside the polygon boundaries. If the z-value at a pixel indicates that the polygon is closer to the viewer than the z-value in the z-buffer, the z-value and the intensity values recorded in the buffers are replaced by the polygons values. After all polygons have been processed, the resulting intensity buffer is displayed [15]. Despite its simplicity, there are two main disadvantages related to this technique: the first one is related to the memory required for depth buffer and the second one is the computation effort wasted on drawing distant points that are drawn over with closer points, that occupy the same pixel.

1.4. Computational complexity

In the rendering literature there is good empirical evidence that at least some of the acceleration schemes achieve sub linear time complexity (for instance better than $O(N)$) but there is a lack of proofs to show what complexity they actually achieve and under what conditions [16]. Mark de Berg [17] has recently developed efficient ray shooting algorithms. He considered the problem for different types of objects (arbitrary and axis parallel polyhedra, triangles with angles greater than some given value) and different types of rays (rays with fixed origin or direction, arbitrary rays). His most general algorithm can shoot arbitrary rays into a set of arbitrary polyhedra with N edges altogether, with a query time of $O(\log N)$ and preprocessing time and storage of $O(N^{4+\epsilon})$, where ϵ is a positive constant that can be made as small as desired. The disadvantage of this algorithm is its high memory requirements, making it not suitable for practical situations. The beam tracing algorithm has been verified to be of $O(N^r)$ [18], with r the number of reflections in the environment and N surface planes. Finally the BSP algorithm developed by Naylor is $O(2^N)$ [19].

The proposed algorithm has a computational complexity of $O(N)$ and is based on a totally new approach with respect to those mentioned before. It determines invisible points exploiting the difference between their relative positions in the 3D reference frame and on the projected image plane. The algorithm has been developed to efficiently estimate the invisible points of a 3D model or a DEM in order to be integrated with the navigation system of an autonomous flying platform (UAV). In the following paragraphs, the algorithm is extensively explained, showing all practical aspects and presenting both a coarse and fine approach to numerically implement it. The case study presented in this paper refers to a camera mounted on board of an aircraft flying above a mountainous region. The moving camera can also be interpreted as a moving light source or a sequence of different points of view in a rendering process.

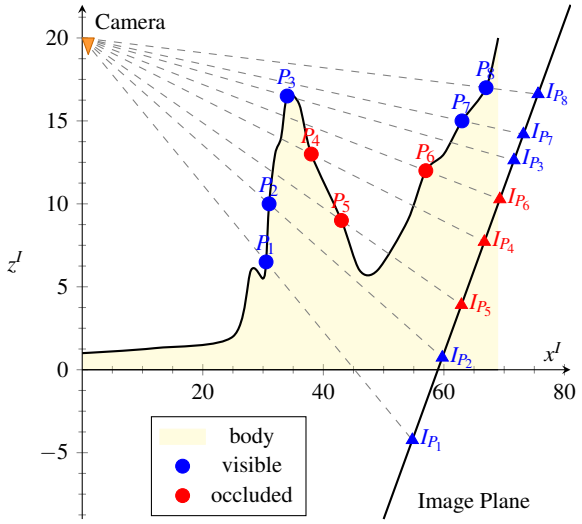

Figure 1: Different order assumed by points in real world and on the image plane

2. Methodology

2.1. Graphical solution

The basic idea behind this new approach is sketched in Figure 1. For simplicity a bi-dimensional problem or a section of a 3D volume is considered here. A mountain profile, or more in general, a 3D body, is represented by the coloured region. In blue and red are points taken on its surface, indicated using a P-letter and numbered from 1 to 8. On the top-left corner of the image, there is the point of view, the camera or the light source. The rightmost oblique line represents a section of the image plane, on which the points $P_1 \dots P_8$ are projected. Each projection is $I_{P_{ij}} = (x^c_{P_{ij}}, y^c_{P_{ij}})$. As a single column is processed in this example, for simplicity of notation, the second index j is omitted. x^c and y^c are obtained using Equation 2 introduced in Section 2.2.1. The blue points in Figure 1 are those visible for an observer at the camera position, instead, the red ones are the occluded points, lying in the shadow of the body. In order to calculate an occluded point, there are different methods, as introduced in the previous paragraph. A first solution could be to intersect the ray (gray dashed line) with the body surface and to check if it is intersected more than once (ray-object intersection). Another solution could be to calculate the peak (point P_3) and regard as occluded each point that is below the ray passing from P_3 and features at a greater horizontal distance from the camera. The solution proposed in this paper is to consider the differences between the relative positions assumed by the points in the Inertial Reference Frame (IRF) and on the image plane. In order to understand how the algorithm works, the problem can be approached firstly from a graphical point of view and then numerically. In Figure 2 the graphical method is schematically presented.

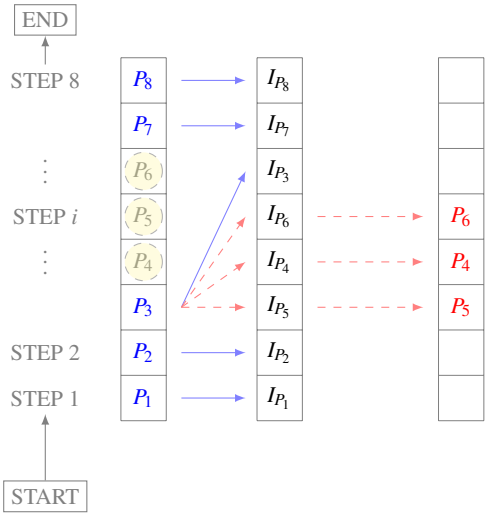

Figure 2: Graphical approach to calculate visible and occluded points

This method compares the vector with the points $P_{i=1\dots8}$, as they appear in the Inertial Reference Frame (IRF), and the vector containing the image projections $I_{P_{i=1\dots8}}$. At step 1, point P_1 is compared with the relative element in the second vector; if it corresponds to the projection I_{P_1}, it means that point P_1 is visible and it is possible to move to step 2. This procedure is repeated up to step 3, where, in the corresponding position of point P_3, there is the projection I_{P_5}. In this case, P_5 is an occluded point and is moved from the first to a third vector. P_3 is sequentially compared with the remaining elements of the second vector until the projection I_{P_3} is found. In these intermediate steps, also P_4 and P_6 are occluded points like P_5 and are moved from the first to the third vector, containing all the invisible points. When the last vector element is reached, the algorithm ends.

From a practical point of view, it is not straightforward to know the order of the projections I_{P_i} in the second vector without using a feature matching algorithm. For this reason, the graphical procedure can not be directly translated to code. Solution to this problem is presented in the next subsection.

2.2. Numerical implementation

In order to overcome the difficulties related to the graphical approach, two different algorithm implementations

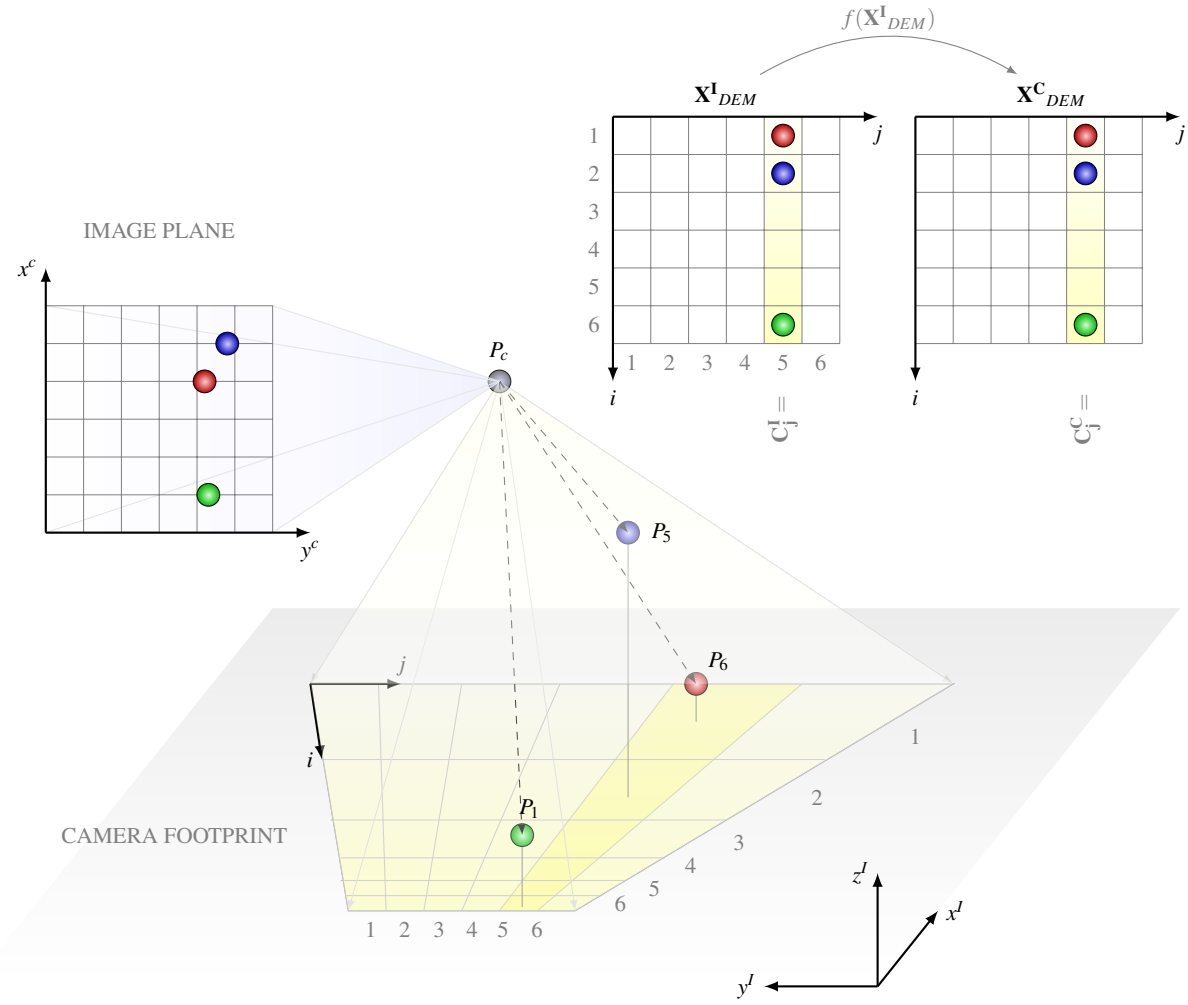

Figure 3: Mesh geometry based on camera footprint and image formation by means of homogeneous transformation f. The figure also shows the image plane $x^c - y^c$, the DEM matrix with the coordinate in the IRF ($\mathbf{X^I}_{DEM}$) and CRF ($\mathbf{X^C}_{DEM}$)

are possible. In this paper, for completeness, both of them are presented.

2.2.1. *Points projection*

The first step of the algorithm is to project the 3D scene in the Camera Reference Frame (CRF). The 3D model is provided in raster form, which means that a matrix containing the geographical coordinates and elevation values is used. One of the basic hypotheses, in order for the algorithm to work, is that the IRF body coordinates have to be in increasing or decreasing order along the x^I and y^I axes. For instance, a body that does not fulfil this hypothesis is the sphere because its surface points can not be uniquely represented by an increasing coordinate. If the sphere is cut horizontally along the equatorial plane, all points of the northern hemisphere respect this hypothesis and the algorithm applies. Of the entire 3D map, only the points in the camera Field of View (FOV) have to be processed. Using simple geometrical relations and camera parameters, the camera footprint can be easily calculated. A point P lies inside the FOV if the following relation is verified:

$$\arccos\left(\mathbf{z^c} \cdot \frac{\mathbf{P} - \mathbf{P_c}}{\|\mathbf{P} - \mathbf{P_c}\|}\right) = \theta_P < \theta_{FOV}, \qquad (1)$$

where $\mathbf{P_c}$ is the camera viewpoint, $\mathbf{z^c}$ is the camera optical axis and θ_{FOV} is the FOV. The occluded point determination can be strongly simplified if the bi-dimensional problem can be reduce to a mono-dimensional one. In order to achieve this result, it is sufficient to take the DEM points, lying in the FOV, on the trapezoidal grid, representing the projection of the

image pixel matrix on the 3D surface, as shown in Figure 3. This particular spatial distribution ensures that any matrix column $\mathbf{C_j}$, where the subscript j indicates the column index, is independent from the others. Using this geometry, a point can be occluded only by another point belonging to the same column and not from those of the adjacent ones. The matrix containing these DEM points in the IRF is $\mathbf{X^I}_{DEM}$ whereas a column of this matrix is $\mathbf{C^I_j}$. In order to project these points in the CRF, the homogeneous transformation $f : I \mapsto c$ is applied

$$\begin{bmatrix} x^c \\ y^c \\ z^c \\ 1 \end{bmatrix} = f(\mathbf{P^I}) = \mathbf{T}^c_I \begin{bmatrix} \mathbf{P^I} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}^c_I & -\mathbf{d}^c_I \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} x^I \\ y^I \\ z^I \\ 1 \end{bmatrix}, \quad (2)$$

where $\mathbf{R}^c_I$ and $\mathbf{d}^c_I$ are respectively the rotational matrix and translational vector (resolved in the image reference frame) which take into account the different orientation and origins of the two reference frames. x^c, y^c and z^c are the transformed coordinates in the CRF while x^I, y^I and z^I are the coordinates of the point P in the IRF. The matrix that contains the projected coordinates is $\mathbf{X^C}_{DEM}$. From Equation 2 it follows

$$\mathbf{C^C_j} = f(\mathbf{C^I_j})$$

where $\mathbf{C^C_j}$ is the j-column containing the projected points of the j-column $\mathbf{C^I_j}$. For example, the three points P_1, P_5 and P_6 in Figure 3, are taken from the column $j = 5$ of the matrix $\mathbf{X^I}_{DEM}$, and their projection in the CRF is qualitatively shown on the image plane $x^c - y^c$. As the point P_6 is occluded by the point P_5, $x^c_{P_6} < x^c_{P_5}$ whereas $x^I_{P_6} > x^I_{P_5}$.

2.2.2. *Suboptimal implementation*

The suboptimal implementation of the algorithm makes use of the matrix row and column indexes $i - j$ to discriminate the occluded points. This implementation comes straightforward if the main idea of exploiting the different order of the points between the IRF and CRF is considered. The matrix to be processed is $\mathbf{X^C}_{DEM}$. The information about the points order in the IRF is intrinsically available from the way in which the matrix $\mathbf{X^C}_{DEM}$ has been populated. Each column of this matrix is representative of a strip in the IRF as shown in Figure 3. For this reason, the points order in the IRF is the same they have in the vector $\mathbf{C_j}$. This property enables to operate directly on the matrix index i instead of the matrix values. The next step is to verify if there are points on the image plane that are differently ordered from those in the IRF. To do this, the x^c coordinates of the vector $\mathbf{C^C_j}$ have to be arranged in decreasing order and the vector associated to the sorted indexes has to be taken. This vector is called $\mathbf{I_s}$ (the capital I stands for the row index i), while the one associated to the unsorted elements is $\mathbf{I_u}$. If the three points example of Figure 3 are considered, $\mathbf{I_u} = \begin{bmatrix} 1 & 2 & 6 \end{bmatrix}^T$, while $\mathbf{I_s} = \begin{bmatrix} 2 & 1 & 6 \end{bmatrix}^T$ because on the image plane P_6 and P_5 are switched. The points moved by the sorting algorithm are only the occluded ones and those from which they are occluded. A vector $\mathbf{I_{1/0}}$ of boolean variables contains the value 1 for these points and 0 otherwise

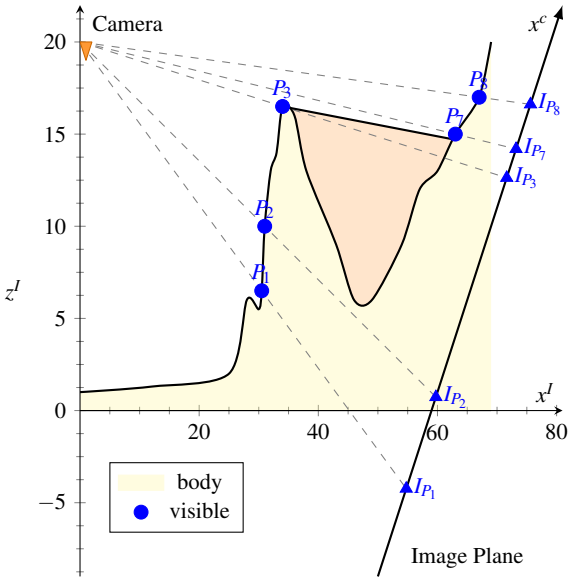

Figure 4: Every point in the scene is visible from the camera viewpoint. The vector $\mathbf{I_{1/0}} = \mathbf{0}_{n_r \times 1}$

$$\mathbf{I_{1/0}} = \begin{cases} 0 & \text{if } i^k_u = i^k_s \\ 1 & \text{if } i^k_u \neq i^k_s \end{cases} \qquad \forall k \in \{1 \dots n_r\}. \quad (3)$$

i^k_u and i^k_s in Equation 3 are respectively the elements at the position k of the vectors $\mathbf{I_u}$ and $\mathbf{I_s}$ and n_r is the number of rows of the matrix $\mathbf{X^C}_{DEM}$. For the example of Figure 3, this vector is $\mathbf{I_{1/0}} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^T$ because only the last element 6 is retained in its original position. In case in the scene there are not occluded points, or for points far from the occlusion region, the values in the boolean vector $\mathbf{I_{1/0}}$ are zero, and they do not require to be processed in the remaining part of the algorithm. The case with not occluded points in the scene and $\mathbf{I_{1/0}} = \mathbf{0}_{n_r \times 1}$ is shown in Figure 4. The surface is the same shown in Figure 1, but the invisible region has

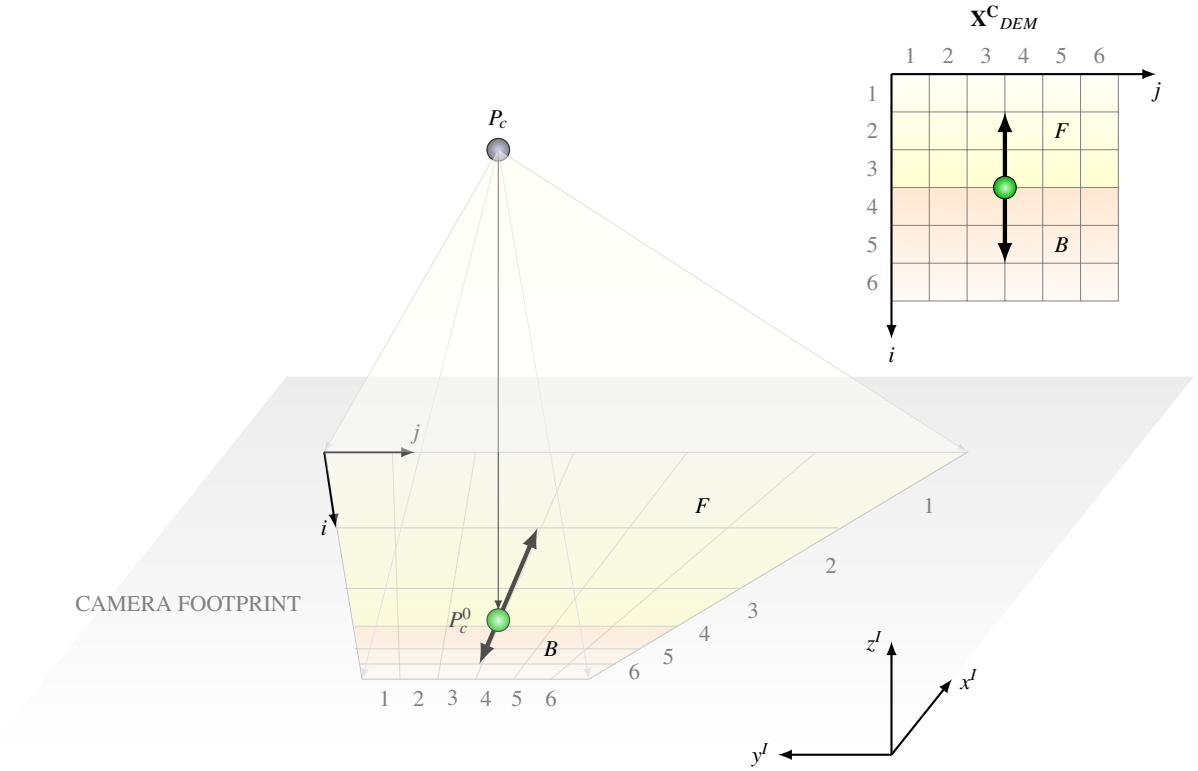

Figure 5: Subdivision of the matrix $\mathbf{X^C}_{DEM}$ and directions in which the subcolumns have to be processed

been filled with the orange volume, eliminating all the occluded points.

The last step of the algorithm discriminates the occluded points among those indexed by the value 1 of the vector $\mathbf{I_{1/0}}$. For the case of Figure 3 only the sub-vector $\mathbf{I_s} = \begin{bmatrix} 2 & 1 \end{bmatrix}^T$ is processed. The maximum value of the reduced vector $\mathbf{I_s}$ is searched and the boolean value, representative of its position is switched from $1 \rightarrow 0$. The procedure is iterated until the maximum is found in the first element vector. The row indexes of the vector $\mathbf{I_s}$ relative to the value 1 in the updated vector $\mathbf{I_{1/0}}$ indicate the occluded points. For the considered example, the maximum value is 2 and the occluded point is the one relative to $i = 1 = P_6$.

2.2.3. *Optimized implementation*

The second approach is much easier to implement than the first one, as it requires a lower number of operations and avoids sorting steps and searching for maximum values. For this reason, it has been preferred to the first one. This algorithm processes directly the values in the columns $\mathbf{C_j^C}$ of the matrix $\mathbf{X^C}_{DEM}$ and does not consider the positional index as the previous one. Before giving the description of the algorithm, it is important to analyse the effect of the camera attitude on the footprint. If the camera is pointing forward, all the points in the FOV fore-run the camera projection P_c^0 (green point in Figure 5) on the plane $x^I - y^I$. If the camera has a high negative elevation angle, or it is pointing downward, also the points that are behind P_c^0 are part of the footprint. This situation is depicted in Figure 5. The yellow region, labelled with the letter F, is the part of the footprint that fore-runs the camera projection P_c^0, whereas the orange region B is the part behind this point. This regions separation is necessary in order to correctly apply the algorithm. The black arrows departing from the point P_c^0 indicate in which direction the two sub-matrices have to be processed. For the example of Figure 5, the region F is processed from $i = 3 \rightarrow 1$, whereas the region B from $i = 4 \rightarrow 6$. The basic idea of the algorithm is to scan the sub-vector of $\mathbf{C_j^C}$ and to sequentially store in a variable k_{max} the maximum value of the projection x^c up to that point. If the value of the next element in the vector is lower than k_{max}, this is an occluded point. The value of k_{max} is retained until the condition $x^c > k_{max}$ is verified again. For simplicity, the case of Figure 6 is considered. As in Figure 5, the yellow region is the F

space and the orange is the B space. If the F region is processed, the algorithm starts by the point P_8 and the variable k_{max} is initialized with $k_{max} = x^c_{P_8}$. At the next step, k_{max} is compared to $x^c_{P_9}$: if

$$x^c_{P_9} > k_{max} \Rightarrow \begin{cases} k_{max} = x^c_{P_9} \\ \mathbf{I^i_{1/0}} = 0 \end{cases} \tag{4}$$

otherwise if

$$x^c_{P_9} < k_{max} \Rightarrow \begin{cases} P_9 \quad occluded \\ \mathbf{I^i_{1/0}} = 1 \end{cases} . \tag{5}$$

This procedure is repeated until the point P_{14} is reached. The superscript i in $\mathbf{I^i_{1/0}}$ indicates the row index of the vector $\mathbf{C^C_j}$ relative to the position of x^c being processed (in this case $x^c_{P_9}$). At the end of the procedure, the occluded points are those relative to the value 1 of the vector $\mathbf{I_{1/0}}$. The same algorithm is applied to the B region, starting from P_7. The variable k_{max} is initialized with $x^c_{P_7}$ and compared to $x^c_{P_6}$. Also in this case the two conditions of Equation 4 and 5 hold. The algorithm ends when P_1 is reached. According to the camera elevation, FOV and platform attitude, one of the two regions (F or B) could be reduced and eventually disappear. In this case, the point P^0_c lies outside the footprint and the entire column $\mathbf{C^C_j}$ is processed upwards or downwards, according to the position of P^0_c. In the case of a forward looking camera, the point P^0_c is behind the footprint and the vector is processed upwards.

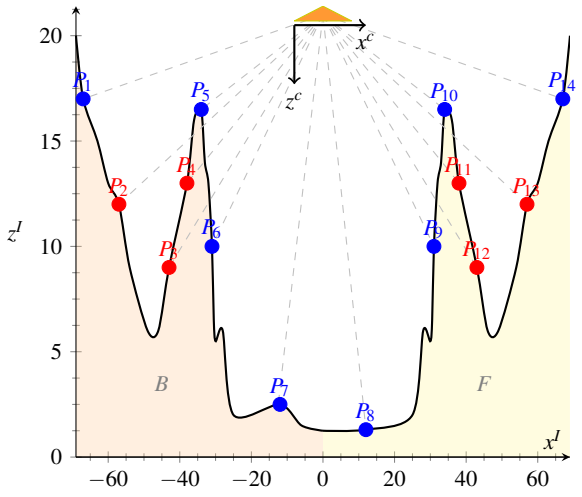

Figure 6: Specular behaviour of the projected points located behind the camera position P^0_c in the IRF

2.3. Numerical example

In this section a simple numerical example of a forward looking camera is presented. As the camera points forward, the vector is processed upwards. The column vector $\mathbf{C^C_j}$ is

$$\mathbf{C^C_j} = \begin{bmatrix} 11 & 2 & 6 & 9 & 10 & 7 & 4 & 3 \end{bmatrix}^T$$

and the occluded values are those relative to the row index $i = 2, 3, 4$. The variable k_{max} is initialized with the last vector value $k_{max} = 3$ and then compared sequentially to the remaining elements. In Figure 7 all algorithm steps are shown schematically.

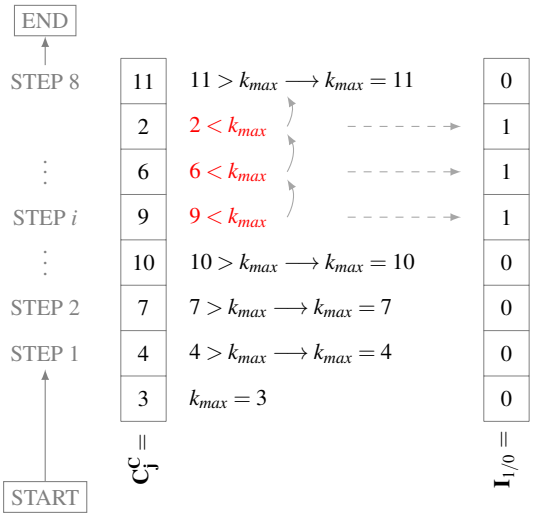

Figure 7: Optimized approach to find the occluded points

3. Model validation

The model proposed in this paper has been validated with the *ray tracing* algorithm [8]. An efficient vectorized implementation of the ray tracing algorithm proposed by Möller and Trumbore has been used.

3.1. Ray-Triangle intersection algorithm

This algorithm verifies if there is an intersection between a ray $\mathbf{r}(t)$ and a triangle defined by its three vertices V_0, V_1 and V_2. The ray $\mathbf{r}(t)$ is defined as

$$\mathbf{r}(t) = \mathbf{O} + t\mathbf{D} \tag{6}$$

where the vectors $\mathbf{O}$ and $\mathbf{D}$ in Equation 6 are respectively the origin of the ray and its direction. A point $\mathbf{P}_T(u, v)$ on a triangle, defined as a function of the barycentric coordinates (u, v) is given by the following equation

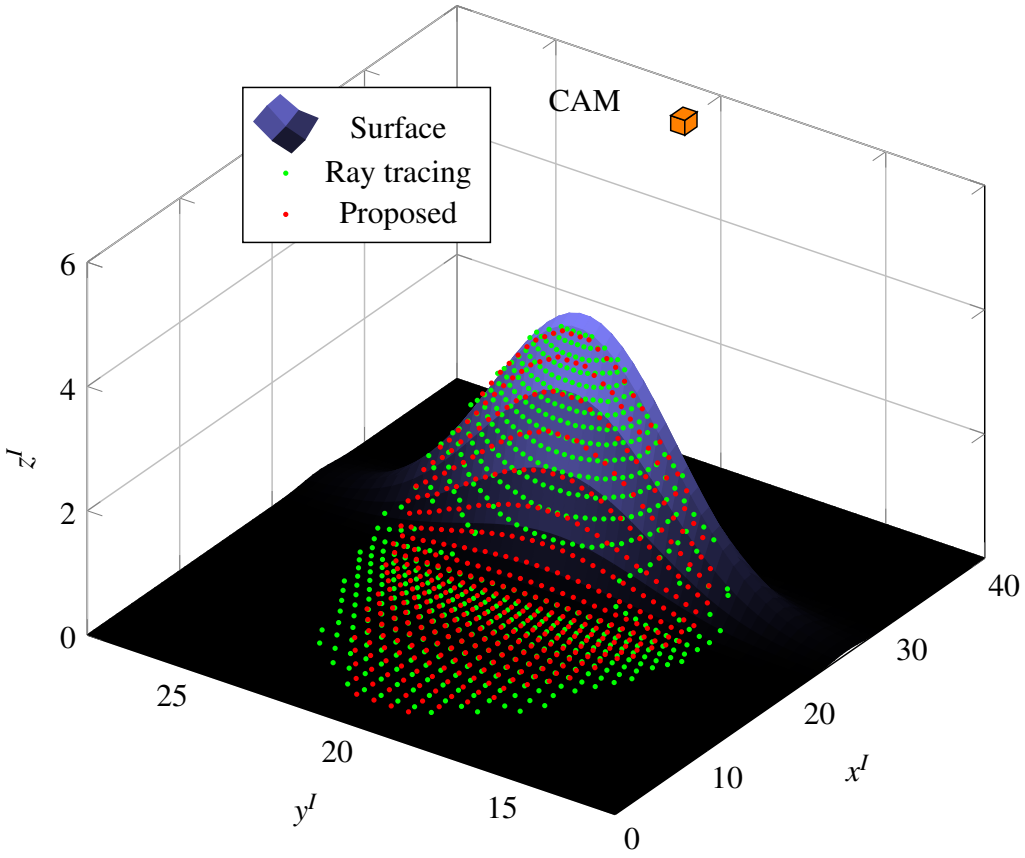

Figure 8: Comparison between the solution provided by the ray tracing and the proposed algorithm

$$\mathbf{P}_T(u,v) = (1-u-v)\mathbf{V}_0 + u\mathbf{V}_1 + v\mathbf{V}_2. \tag{7}$$

where $u \geq 0$, $v \geq 0$ and $u + v \leq 1$. The intersection between the ray and the triangle is given by $\mathbf{r}(t) = \mathbf{P}_T(u,v)$

$$\mathbf{O} + t\mathbf{D} = (1-u-v)\mathbf{V}_0 + u\mathbf{V}_1 + v\mathbf{V}_2. \tag{8}$$

Rearranging this equation, the following linear system in the variables t, u, and v is obtained

$$\begin{bmatrix} -\mathbf{D} & \mathbf{V}_1 - \mathbf{V}_0 & \mathbf{V}_2 - \mathbf{V}_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \mathbf{O} - \mathbf{V}_0. \tag{9}$$

The solution of this system provides the intersection point coordinates. Defining the origin $\mathbf{O}$ as the camera viewpoint, and $\mathbf{D}$ the direction relative to each camera pixel, the intersection with each triangle of the body surface mesh, obtained by means of the Delaunay triangulation, can be found. If there is an occluded point along a given direction, the ray will intersect the surface at least twice. The occluded point is given by the intersection that lies at a greater distance from the camera, and so with a higher value of the variable t.

3.2. Validation solution

A comparison of the results provided by the proposed method and the *ray tracing* (or ray triangle) algorithm are shown in Figure 8. The surface used for validation is given by a bivariate Gaussian distribution. The camera has a pixel array of 100×100, and is located in one side of the body with an elevation angle of $\theta_{el} = -60$ deg with respect to the horizon. With this configuration, one part of the body will be in the shadow. The green solution is the one provided by the ray tracing algorithm, while the red points are those calculated with the proposed algorithm. The shadow region calculated with the two algorithms is practically the same, except for a very small region along the boundaries and at the base of the body. The loss of green points at the base of the body is related to a weakness of the ray tracing algorithm; if the triangular surface patch is parallel to the ray direction, there will be not intersection with the ray, which

results in a loss of information about the occluded point, while the proposed algorithm is able to correctly cover any part of the body. A second very important result is that while in the ray tracing case the solution accuracy is both a function of the body mesh and pixel resolution, in the proposed algorithm the solution is only related to the camera parameters and is not affected by the adopted body discretisation. A third relevant result is related to the time required to find a solution. In the case of the ray tracing, the algorithm calculates for each camera pixel the intersection with the surface mesh triangles, which means that the time is function of both the number of pixels and mesh resolution. For the proposed algorithm, the computational time is only related to the image formation e verification for occlusions. For this reason the algorithm depends only on the number of camera pixels.

4. Case study

This paragraph shows how the algorithm behaves when a more complicate scenario is presented. In this case a 3D Digital Elevation Map taken from a Geographic Information System (GIS) resource database [20] has been considered. It represents part of the Mount Mitchell in Yancey County (North Carolina), with a map accuracy of ≈30 m. This particular scenario has been chosen in order to highlight the power of the algorithm when a very complex scene is processed. The aim is to simulate what is seen from a gimbal-camera mounted on a moving platform. In this particular situation, if any point in the camera footprint is projected from the DEM matrix onto the image plane, both visible and invisible points will appear on it, as shown in the example of Figure 1. The first step, before calculating the invisible points, is to define the camera footprint, represented by the yellow points in Figure 9.

In order to calculate them, it is necessary to project the points defined in the IRF onto the image plane by applying Equation 2. Observing the Figure 10, the homogeneous transformation $\mathbf{T}_I^c$ is given by

$$\mathbf{T}_I^c = \mathbf{T}_g^c \mathbf{T}_b^g \mathbf{T}_I^b = \begin{bmatrix} \mathbf{R}_g^c & -\mathbf{d}_g^c \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_b^g & -\mathbf{d}_b^g \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_I^b & -\mathbf{d}_I^b \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \tag{10}$$

where $\mathbf{T}_I^b$ projects the points from the IRF to the body reference frame, $\mathbf{T}_b^g$ from body to gimbal and $\mathbf{T}_g^c$ from gimbal to camera.

The direction cosine matrix $\mathbf{R}_I^b$ is

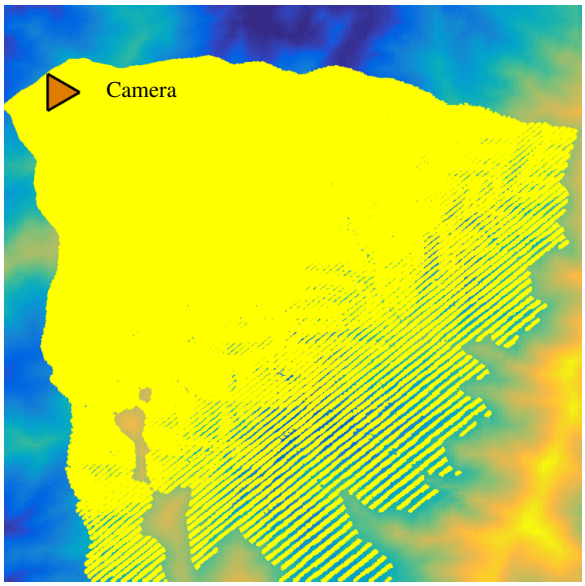

Figure 9: Camera footprint. Yellow points are those inside the camera field of view

$$\mathbf{R}_I^b = \begin{bmatrix} c_\phi c_\theta & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ s_\phi c_\theta & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix}$$

where $c_\bullet$ and $s_\bullet$ stand for $\cos(\cdot)$ and $\sin(\cdot)$, while ϕ, θ and ψ are respectively the Euler angles of roll, pitch and yaw. Matrix $\mathbf{R}_b^g$ is

$$\mathbf{R}_b^g = \begin{bmatrix} c_{\theta_{el}} c_{\psi_{az}} & c_{\theta_{el}} s_{\psi_{az}} & s_{\theta_{el}} \\ -s_{\theta_{el}} & c_{\psi_{az}} & 0 \\ -s_{\theta_{el}} c_{\psi_{az}} & -s_{\theta_{el}} s_{\psi_{az}} & c_{\theta_{el}} \end{bmatrix}$$

where θ_{el} is the gimbal elevation angle, defined in the $x^b - z^b$ plane, and ψ_{az} is the gimbal azimuth angle defined in the $x^b - y^b$ plane. They are used to rotate the camera with respect to the body reference frame. Matrix $\mathbf{R}_g^c$ is constant if the relative motion between camera and gimbals is neglected

$$\mathbf{R}_g^c = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

Once the map points have been projected on the image plane, to discriminate the points in the field of view from the ones outside, it is sufficient to verify which of them respects the following two conditions

$$\begin{cases} 0 \le x^c \le n_r \delta_{p_x} \\ 0 \le y^c \le n_c \delta_{p_y} \end{cases} \tag{11}$$

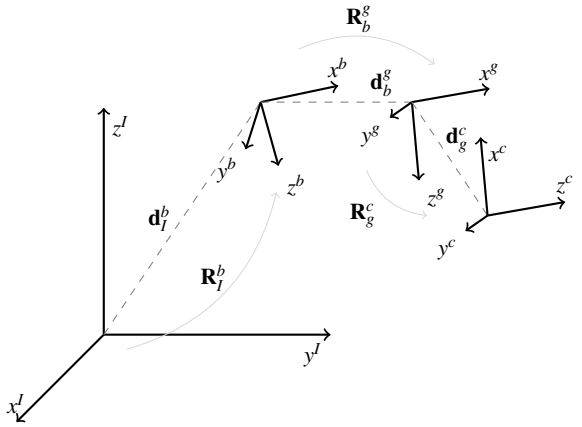

Figure 10: Rotations and translations required to pass from the IRF to the camera reference frame

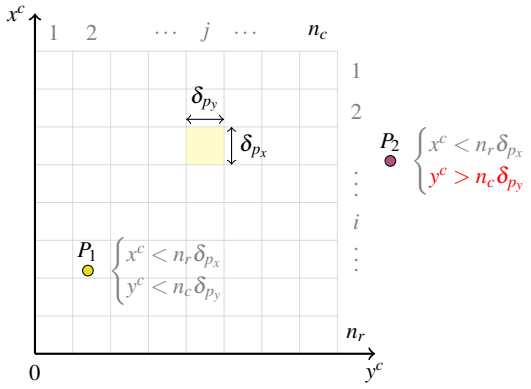

Figure 11: Image plane with points inside the field of view (P_1) and outside of it (P_2)

where x^c, y^c, n_r and n_c have been defined in Section 2.2.1, whereas δ_{p_x} and δ_{p_y} are the pixel dimensions along x^c and y^c respectively, as shown in Figure 11. In this figure there are also two points, P_1 and P_2; the first one is part of the footprint because it fulfils both conditions of Equation 11, whereas the second one is discarded because y^c exceeds the upper boundary along that direction. The camera parameters are the focal length $f_c = 0.035\,\text{m}$, the number or rows $n_r = 300$ and columns $n_c = 300$ and the pixel dimensions (assumed to be equals) $\delta_{p_x} = \delta_{p_y} = 0.36\,\text{mm}$. The focal length f_c is used to calculate the camera field of view θ_{FOV_x}, in the $x^c - z^c$ plane, and θ_{FOV_y}, in the $y^c - z^c$ plane

$$\theta_{FOV_x} = \arctan\left(\frac{n_r\delta_{p_x}}{f_c}\right) \tag{12}$$

$$\theta_{FOV_y} = \arctan\left(\frac{n_c\delta_{p_y}}{f_c}\right). \tag{13}$$

At this point, it is possible to apply the algorithm described in the previous section to discriminate between the visible points and the invisible ones.

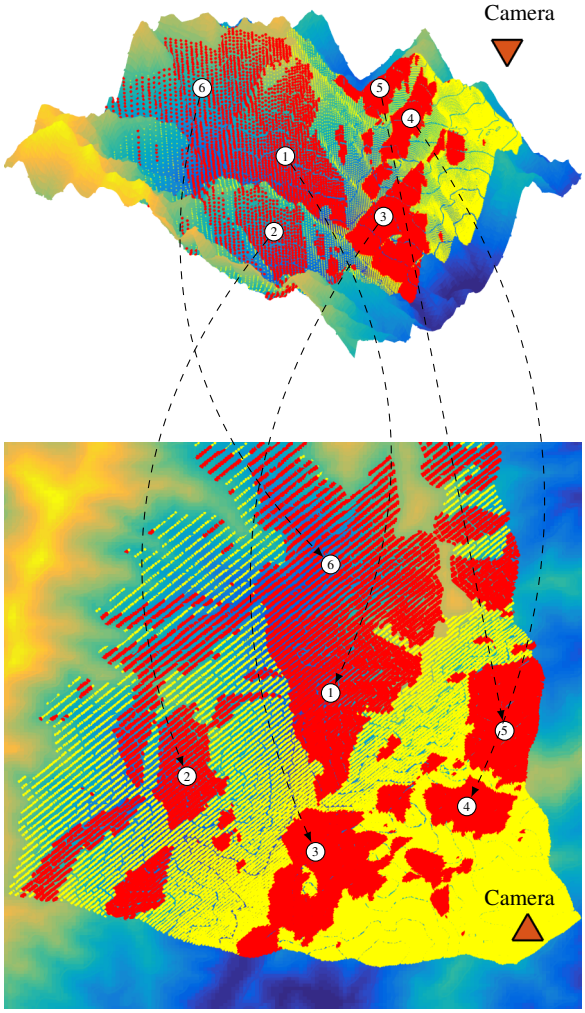

Figure 12: The visible region (blue points of Figure 1) are represented by the yellow points, occluded points are in red. The arrows associate the same regions on the 3D sight and its plain view

The red region of Figure 12 is characterized by not visible points from the camera position, or it represents the mountain shadow if it is assumed that the camera is replaced by a light source. In order to better visualize the results, both a 3D and the plain view of the scene

are provided. To simplify the association between the same regions on the two sights, numbered arrows have been added. The interesting result that emerges, is the complexity of the shape of the shadowed regions. This confirms the capability of the algorithm of managing extremely complex scenes, like mountains or spaces populated by a multitude of objects. Obviously, the solution accuracy is function of the number of pixels (or camera resolution) and distance from the camera. A further very important property of this algorithm is its intrinsic dependency by the camera parameters. This enables to simulate the effect of different light sources on the scene: if the focal length is increased, the FOV is reduced and the light is collimated; conversely, a diffusive light is realized by reducing f_c.

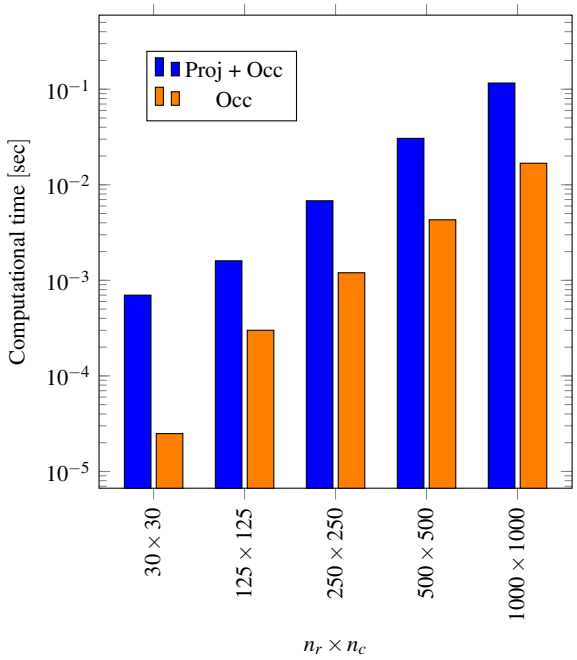

Figure 13: Algorithm computational time as a function of the number of pixels, averaged over 50 simulations

The algorithm has a computational complexity of $O(N)$, whith N the number of pixels. The results proposed in this paper have been obtained using the software MATLAB on an Intel Core i7-4710HQ 64 bit, 2.50 GHz with 16 GB RAM. In order to show the algorithm performance in the worst conditions, the matrix columns have been processed sequentially and not by parallelizing the code, and the FOV has been kept wider than usual in order to increase the number of points to process. From Figure 13, it is possible to appreciate the computational time as a function of the number of pixels, or equivalently n_r and n_c. Considering that the FOV is function

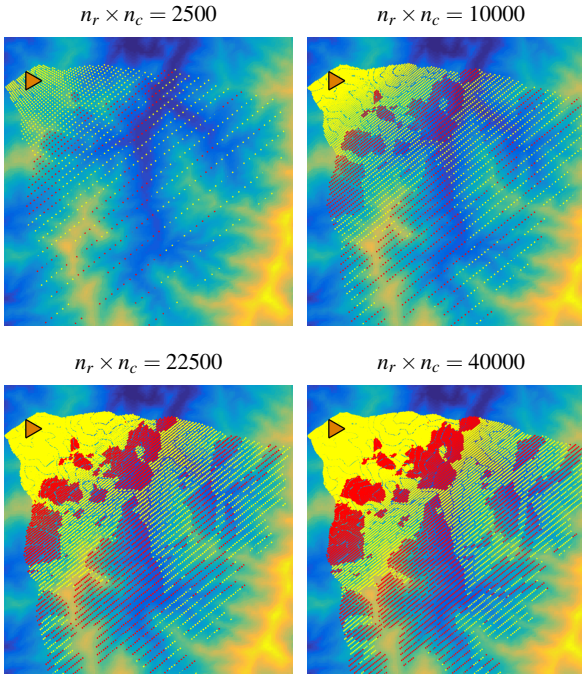

Figure 14: Level of detail of the visible and occluded regions by increasing the number of pixels

of the number of pixels, in order to maintain unchanged the observed region, the value of the focal length f_c has been increased case by case. The blue bars in Figure 13 represent the total computational time required to apply the homogeneous transformation f to every point in the field of view, plus the time required by the algorithm to evaluate the occluded points. Conversely, the orange bars are representative exclusively of the time required to calculate the occluded points. Figure 14 shows how the solution is affected by the number of pixels. What is really interesting, is that for $n_r \times n_c > 10000$, the occluded region boundaries are not affected by any variation. This means that an array of pixels approximatively of 100×100 and a computational time in the order of $O(10^{-4})$ seconds are sufficient to provide an excellent solution. What should be noticed, is that the found solution is punctual, but if any processed point is representative of a surface patch, as in the case of a surface-ray intersection algorithm, the entire patch (except for those along the boundaries of the occluded region) can be considered in the shadow. In this case, being the boundaries of the occluded region not strongly affected by the number of points, a relatively low number of points need to be processed, even in the case of complex geometries, without compromising the accuracy of the solution. Figure 15 shows the solution provided by the algo-

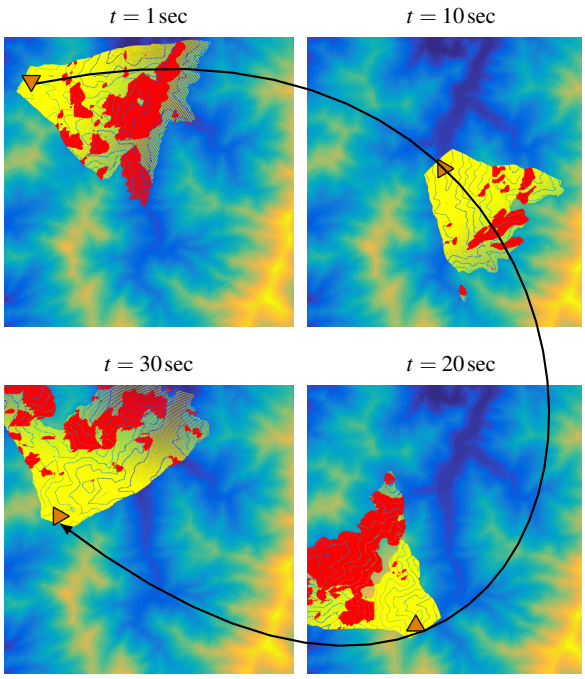

Figure 15: Algorithm solution at different time steps along the simulated trajectory (black curve)

rithm when the camera is driven by the platform along a given trajectory (black line). In this case, the FOV has been reduced in order to simulate a more realistic situation.

5. Conclusions

This paper presents a very simple and efficient algorithm $O(N)$ to determine the invisible regions of a generic 3D model, when it is observed from a given point of view. The solution is initially presented graphically, and then numerically, in order to reach readers with heterogeneous background, being the algorithm open to a very wide range of applications. The numerical simulations highlight the algorithm capability of providing extremely accurate solutions by using a limited number of points, also when the observed scene is very complex. This algorithm, as any other graphical tool, is suitable for parallel computing, because the processing of the columns in the pixel matrix is independent from the results on the other columns. This feature would be effective in decreasing the computational time, which is crucial in computer graphics. In addition, the implementation simplicity enables the algorithm to be integrated with other tools with minimum efforts.

Declaration of conflict of interest

The authors declare that there are no conflicts of interest.

References

[1] J. Bittner, Hierarchical techniques for visibility computations, Ph.D. thesis, Faculty of Electrical Engineering, Czech Technical University in Prague (2002).
[2] S. Teller, K. Bala, J. Dorsey, Conservative radiance interpolants for ray tracing, in: Rendering Techniques 96, Springer, 1996, pp. 257–268. doi:10.1007/978-3-7091-7484-5_26.
[3] A. S. Glassner, An introduction to ray tracing, Elsevier, 1989.
[4] S. Teller, J. Alex, Frustum casting for progressive, interactive rendering, Tech. rep., MIT, Cambridge, MA, USA (1998).
[5] B. F. Naylor, Interactive solid geometry via partitioning trees, in: Proc. Graphics Interface, Vol. 92, 1992, pp. 11–18.
[6] B. Naylor, Constructing good partitioning trees, in: Graphics Interface, CANADIAN INFORMATION PROCESSING SOCIETY, 1993, pp. 181–181.
[7] B. Naylor, Binary space partitioning trees as an alternative representation of polytopes, Computer-Aided Design 22 (4) (1990) 250–252. doi:10.1016/0010-4485(90)90055-H.
[8] T. Möller, B. Trumbore, Fast, minimum storage ray/triangle intersection, in: ACM SIGGRAPH 2005 Courses, ACM, 2005, p. 7. doi:10.1145/1198555.1198746.
[9] P. Shirley, R. K. Morley, Realistic ray tracing, AK Peters, Ltd., 2008.
[10] P. Bauszat, M. Eisemann, M. A. Magnor, The minimal bounding volume hierarchy., in: VMV, 2010, pp. 227–234. doi:10.2312/PE/VMV/VMV10/227-234.
[11] P. S. Heckbert, P. Hanrahan, Beam tracing polygonal objects, ACM SIGGRAPH Computer Graphics 18 (3) (1984) 119–127. doi:10.1145/964965.808588.
[12] J. Amanatides, Ray tracing with cones, SIGGRAPH Comput. Graph. 18 (3) (1984) 129–135. doi:10.1145/964965.808589.
URL http://doi.acm.org/10.1145/964965.808589
[13] B. F. Naylor, Partitioning tree image representation and generation from 3d geometric models, in: Proceedings of Graphics Interface, Vol. 92, 1992, pp. 201–212.
[14] H. Oliveira, A. Habib, A. Dal Poz, M. Galo, Height gradient approach for occlusion detection in uav imagery, The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 40 (1) (2015) 263. doi:10.5194/isprsarchives-XL-1-W4-263-2015.
[15] K. I. Joy, The depth-buffer visible surface algorithm, On-Line Computer Graphics Notes, Visualization and Graphics Research Group, Department of Computer Science, University of California-Davis.
[16] B. Walter, P. Shirley, Cost analysis of a ray tracing algorithm.
[17] O. v. K. M. de Berg M., Cheong, O. M., Computational Geometry, Algorithms and Applications, 3rd Edition, Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-77974-2.
[18] T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, J. West, A beam tracing approach to acoustic modeling for interactive virtual environments, in: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, ACM, 1998, pp. 21–32.
[19] S. F. Buchele, A. C. Roles, Binary space partitioning tree and constructive solid geometry representations for objects bounded by curved surfaces., in: CCCG, Citeseer, 2001, pp.

49–52. doi:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.2159&rep=rep1&type=pdf.

[20] webGIS (2009). [link].
URL http://www.webgis.com/terr_us75m.html