Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Advancing Digital Twins: Accelerating and Enhancing Workflows in Bin Picking Applications.

Ph.D. candidate
Paolo Scremin

Advisor
Prof. Ruggero Carli

Director & Coordinator
Prof. Andrea Neviani

Ph.D. School in
Information Engineering

Department of
Information Engineering
University of Padova

2023

University of Padova
Department of Information Engineering

Università
degli Studi
di Padova

Ph.D course in: Information Engineering
Curriculum: Information and Communication Technologies
Cycle: XXXVI

Advancing Digital Twins: Accelerating and Enhancing Workflows in Bin Picking Applications.

Advisor: Prof. Ruggero Carli
Director: Prof. Andrea Neviani

Ph.D. candidate: Paolo Scremin

Year 2023

Abstract

In recent decades the fields of computer vision and industrial automation have been undergoing unprecedented evolution. As a result, applications that were merely theoretical hypotheses are now reality. Thanks to better controllers, sensors and computational power, robots are able to perform increasingly complex tasks without human intervention or supervision.

One such application is "random bin picking," that is the recognition and subsequent manipulation of objects whose placement is unknown in a bin. To obtain the data of the object to be recognized, expensive scanners or 2D cameras are used to generate datasets on real samples of it.

In recent years, to reduce the time and cost of generating datasets needed for model training and tuning of search algorithms, more and more is being invested in the generation of virtual datasets and the creation of digital twins.

In particular the necessary point clouds or 2D views necessary for those algorithms, will be generated starting from a 3D model of the object. However, the 3D models provided, often contains geometry that is not useful for the dataset generation. Such geometry should be removed to speed up the process and avoid training errors. Real cameras also contain imperfections in their lenses and alignment that must be simulated to generate valid datasets.

This work therefore aims to achieve two goals:

1. To provide a solution to simplify generic 3D models by removing invisible geometry from the outside;

2. To provide a solution for simulating the radial distortion of 2D camera lenses.

This research work was carried out in collaboration with the company Euclid Labs of Nervesa della Battaglia, which over the years has specialized in offering solutions for bin picking.

Sommario

Negli ultimi decenni i campi della computer vision e dell'automazione industriale stanno subendo una evoluzione senza precedenti. Come risultato di questi progressi applicazioni che erano solamente ipotesi teoriche si stanno concretizzando. Grazie a migliori controllori, sensori e potere computazionale i robot sono in grado di compiere compiti sempre più complessi e senza l'intervento o supervisione umana.

Una di queste applicazioni è il "Random bin picking" ovvero il riconoscimento e successiva manipolazione di oggetti la cui posa è sconosciuta in cassoni o altri contenitori. Per ottenere i dati dell'oggetto da ricercare o della scena generalmente vengono impiegati costosi scanner o telecamere 2D per generare dataset su esemplari reali dell'oggetto da ricercare.

Negli ultimi anni per ridurre i tempi e i costi della generazione dei dataset necessari al training dei modelli e tuning degli algoritmi di ricerca, si sta investendo sempre più sulla generazione di dataset virtuali e sulla creazione di gemelli digitali.

In particolare, partendo da un modello 3D del pezzo si cerca di estrarne la nuvola di punti o le viste 2D necessarie a tali algoritmi. Tuttavia i modelli 3D spesso forniti contengono geometria non utile ai fini pratici della generazione dei dataset, che andrebbe rimossa per velocizzare il processo ed evitare errori di training. Le telecamere reali inoltre contengono imperfezioni nelle loro lenti e nel loro allineamento che devono esser simulate per generare dei dataset validi.

Questo lavoro mira perciò a raggiungere due obiettivi:

1. Fornire una soluzione per semplificare modelli 3D generici, rimuovendo la geometria invisibile dall'esterno;

2. Fornire una soluzione per la simulazione della distorsione radiale delle lenti delle camere 2D.

Questo lavoro di ricerca è stato svolto in collaborazione con l'azienda Euclid Labs di Nervesa della Battaglia, che negli anni si è specializzata a offrire soluzioni per il bin picking.

Acknowledgements

I would like to thank Walter Zanette and Roberto Polesel and the rest of the Euclid Labs team for this opportunity.
I would also like to express my gratitude to my supervisor, Prof. Ruggero Carli, for overseeing my work and providing valuable advice throughout these three years.
I want to extend a special thanks to Alessandro Rossi and Marco Barbiero for their collaboration and friendship that we have shared.
Finally, I would like to thank my family and Alessandra for supporting me during these years and bet on me.

To Francesca, Arianna, Giorgia and Christian, the world's best nephews.

Contents

1 Introduction

1.1 The Bin Picking Problem and the Benefits of Digital Twins

In today's rapidly evolving world, automation has emerged as a driving force behind increased efficiency, productivity, and competitiveness.
Mass production of goods that are all the same is a paradigm of the past. Now each consumer demands a certain level of customization [1] and a lot of products change and improve very rapidly [2]. Therefore today the manufacturing industry has at its core the concept of *flexibility*. Today, many industries have embraced a production layout organized into islands, each consisting of smaller, versatile production cells. These cells can be swiftly reconfigured and adapted to manufacture diverse products on the fly.

The Bin Picking Problem tries to find a way for an automatic system to effectively and reliably locate, grasp, and manipulate given objects in unknown position in a scene. By solving this problem an automated method for the loading and unloading of these production units will be established, yielding additional benefits such as:

- Increase Efficiency: reduced cycle times, labor costs and errors in material handling.
- Enhance Safety: reducing manual labor in physically demanding and injury-prone tasks.

- Precise Manufacturing: loading and unloading machines in an accurate and repeatable way contribute to product quality and consistency.

Therefore this problem is at the intersection of the fields of robotics and computer vision. Tasks like image segmentation, pose estimation, collision detection, trajectory planning are all part of what engineers have to face to create a bin picking application.

In addition, the application needs to be as *flexible* as the production unit that it has to serve. Since down times in a production unit represent a cost, it is important that the system can be set in operation with minimal delay and maximum efficiency in the transitions between different production tasks or product types.

To prevent downtime and help identify issues in the productions digital twins of the system can be used. Digital twins through the simulation of the real system offers benefits such as:

- Algorithms optimization: through simulation it is possible to refine and optimize algorithms for object recognition, grasp and path planning.
- Flexibility: it can help planning modifications to the system to accommodate changes in the production. It can help identify potential issues with the new setup.
- Remote monitoring and control: digital twins can be accessed remotely, allowing real-time monitoring and control from anywhere in the world.
- Cost Savings: digital twins do not need physical prototypes. This help to minimize the number of failed attempts, reducing material wastage, equipment damages and to minimize the downtime of the system.

ROS [3] and Gazebo [4] are well-known open-source software employed in the field of robotics and bin picking applications. These platforms offer flexible frameworks for robot development, simulation, and control.
In the recent years Euclid Labs, the company I collaborated with during my PHD years, has developed proprietary solutions for such tasks by creating :

- the free software Vostok [5], aimed for students and robotics enthusiasts.
- MARS[6], commercial product for the automation industry.

Both software are capable of simulating a digital twin for a cell, with MARS being the more comprehensive option. In particular MARS is capable of:

- Simulating: It is possible to simulate 2D cameras, 3D scanners and robot movements with collision avoidance.

- Aiding to develop and test a cell: It is possible to develop a digital twin of a cell. Defining also localization algorithms, cycle logic and path planning solutions.
- Comparing solutions: It is possible to compare the developed localization algorithms, generated paths, collision checking and path planning.
- Remote monitoring and control: connecting the real hardware to the program, the digital twin will reflect the real system's status, allowing for comprehensive monitoring and analysis. MARS is also designed to control the real Hardware.

In figure 1.1 the digital twin of a real working cell is presented.

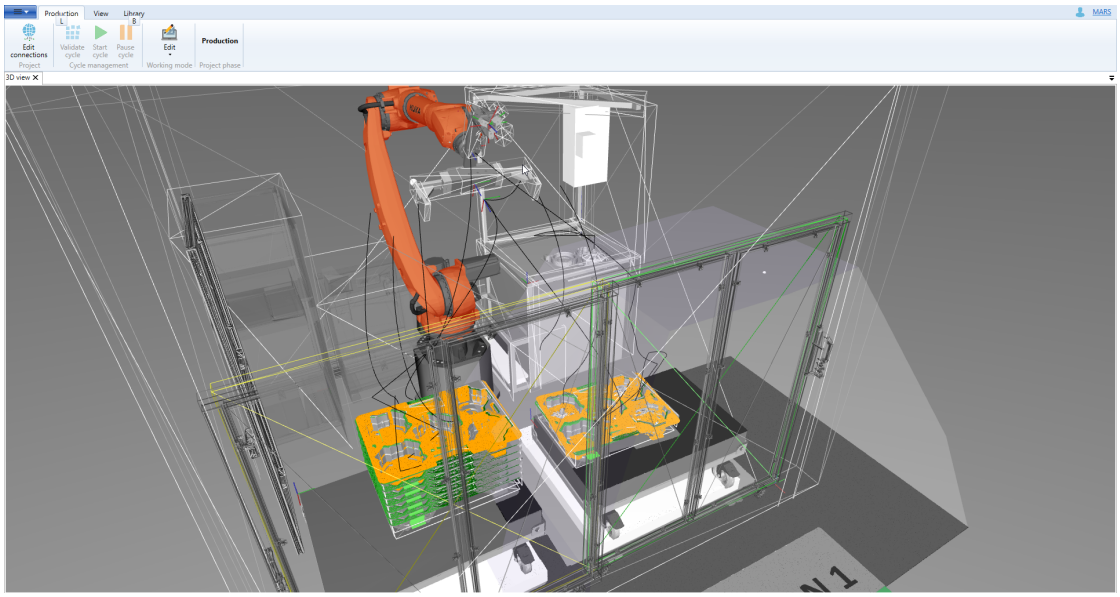


Figure 1.1: MARS simulating a workcell. The robot fills the right green blister with clutches. After filling the blister a new one is taken from the left stack. The blisters are not stationary and their pose are dynamically estimated using a simulated point cloud, highlighted in orange.

The results of our research, presented in this thesis, have been implemented in both software to improve the robustness of the recognition algorithms and camera simulation.

1.2 Manuscript Overview

This dissertation explores the optimization of bin picking workflows, with a specific focus on improving algorithms from 3D models to enhance speed and precision and the improvement of 2D camera simulation.

In the context of bin picking applications, training algorithms to perform tasks such as object recognition and manipulation is a fundamental step. These algorithms heavily rely on the accuracy and completeness of the 3D models that represent the objects to locate. However, the efficiency of these algorithms can be compromised when dealing with intricate 3D models that include non-visible or unnecessary details.

Chapter 2 faces a problem encountered during the simulator development. Our research aims to optimize the models used for training, ensuring that they contain only relevant information, eliminating non-visible components, and ultimately resulting in more streamlined and efficient bin picking operations. Visible-surface determination methods provide one of the most common solutions to the visibility problem. Our study presents a robust technique to address the global visibility problem in object space that guarantees theoretical convergence to the optimal result. We propose a strategy, based on the ambient occlusion estimate, that in a finite number of steps determines if each face of the mesh is globally visible or not. The proposed method is based on the use of Plücker coordinates that allows it to provide an efficient way to determine the intersection between a ray and a triangle. This algorithm does not require pre-calculations such as estimating the normal at each face: this implies the resilience to normals orientation. This technique leads to better performance in terms of data efficiency, as confirmed by experimental results. The contributions of Chapter 2 are based on Rossi, Barbiero, Scremin, Carli [7].

The adoption of 3D scanners in the automotive industry is increasingly common. However, the usage of 2D cameras remains prevalent in tasks like quality control and assembly line monitoring, and it is also possible to use them in a bin picking application. The choice is made primarily due to their cost-effectiveness and ease of integration.

Chapter 3 presents an efficient and robust approach for simulating the lens distortion parameters of 2D cameras. Starting from Brown's model on radial distortion [8], we investigate the limits of the inversion of it and we provide a comprehensive solution to the problem.

2

Robust Visibility Surface Determination in Object Space via Plücker Coordinates

2.1 Introduction

Nowadays, computer vision plays an increasing role in the industrial robotics area [9]. With the transition to Industry 4.0, robotic systems must be able to work even more independently, without the constant supervision of a human operator. Those systems must be able to see and perform decisions based on what they perceive [10]. For example, consider to grab an object, randomly placed within a box, with a robotic manipulator. Typically, a scanner captures a three-dimensional image of the box and then a matching algorithm compares it with the 3D model of the object to find correspondences. By means of complex algorithms, a compatible robot path is then planned. The problem of those algorithms comes with their complexity which is proportional to mesh dimension, i.e., the number of its triangles. At once, they must be efficient in order to satisfy the imposed cycle time and the best practice to reduce it is by removing unnecessary triangles. Especially in the industrial field, many models are characterised by detailed interiors which are invisible. Most of the faces are hidden inside the model and, therefore, this information is redundant for recognition purposes. In some cases, it can be a considerable part of the total mesh, leading to a waste of system memory

and a significant decrease in performance. In this scenario, an algorithm is essential for reducing the number of triangles without losing useful information. In literature, there exist several different techniques to remove hidden surfaces. They are commonly known as visible-surface determination (VSD) algorithms. There exist two main approaches: image space and object space. The former exploits rasterization rendering techniques to convert objects to pixels. Visibility is decided at each pixel position on the projection plane and, therefore, the complexity and the accuracy depend on the resolution of the view image. Image space techniques quickly provide a result, even if it is bound to a particular view. In other words, surfaces removed for a viewpoint may be visible from another one. This implies the need to run the algorithm every time the object pose or view changes. These features make those methods suitable for rendering optimizations, e.g., to limit the number of primitives to draw achieving better frame rates. This procedure is handled differently by several algorithms: z-buffering [11], binary space partitioning [12], ray tracing, Warnock [13] and Painter algorithms [14] are the most popular ones. In some scenarios, it is preferable to have a global result, which is view independent, even if at a higher computational cost. Actually, object space methods compare each mesh face with all the others to determine which surfaces, as a whole, are visible. Given the dependence on the number of faces those methods are slower but more precise. Some examples have been proposed in [15, 16]. There also exists an hybrid approach that exploits an image space technique with different views to merge the results obtaining an approximated global result [17].

This study presents a robust technique to address the global visibility problem in object space. The aim of this work is to develop an algorithm that guarantees theoretical convergence to the optimal result. In particular, we prove that, with a proper sampling technique, it is possible to correctly classify all the visible faces of a mesh using a finite number of computations. In addition, the proposed method does not require pre-calculations such as estimating the normal at each face, which is necessary information for algorithms relying on image space. These properties make this algorithm suitable for offline evaluation to optimize a 3D model by reducing its size. A typical background of interest of this application is the industrial one where the user needs to recognize an object to plan its manufacturing. Usually, such object does not deform but continuously changes its pose. In this case, it would be enough to pre-process the 3D model once using our method before running recognition algorithms.

Actually, we propose an algorithm based on the definition of ambient occlusion to determine the visibility of 3D model triangles. We exploit the ray tracing technique and, consequently, ray-triangle intersection algorithm. This process is further improved by

using a test based on Plücker coordinates instead of the widely known Möller-Trumbore algorithm [18]. Plücker coordinates have already been adopted in exact visibility culling methods [19]: despite this, it is difficult to find a comprehensive mathematical treatment and an algorithm that achieves the optimum in solving the visibility problem. Finally, this approach is numerically tested to validate both the choice of such intersection test and the performance with respect to a state-of-the-art VSD method.

The chapter is organized as follows. Section 2.2 summarizes the most recent works regarding VSD methods and the use of Plücker coordinates to speed up ray-triangle intersection tests. Section 2.3 shows the problem formulation in detail, focusing on the description of ambient occlusion and Plücker coordinates. Section 2.5 presents design and software implementation of the algorithm: special emphasis is placed on its limits, highlighting possible future improvements. In Section 2.6, the proposed solution is tested to analyse compression, quality and performance. The results are then compared to the ones obtained using the VSD method implemented in the MeshLab software. Finally, concluding remarks and possible extensions are presented in Section 2.8.

2.2 Related Works

Visibility computation is crucial in a wide variety of fields like computer graphics, computer vision, robotics, architecture and telecommunication. First visibility estimation algorithms aimed to determine visible lines or surfaces in a synthesized image of a three-dimensional scene. These problems are known as visible line or visible surface determination. There exist many different techniques to address the visibility problems, but we can identify two widespread algorithms: z-buffering for local visible surface determination and ray tracing for computing global visibility. The z-buffering and its modifications dominate the area of real-time rendering, whereas ray tracing is commonly used in the scope of global illumination simulations. Besides these, there is a plethora of algorithms to solve specific visibility problems. Sutherland et al.[14] provides a survey on ten traditional visible surface algorithms. Durand [20] gives a comprehensive multidisciplinary overview of visibility techniques in various research areas. Bittner et al. [21] provides a taxonomy of visibility problems based on the problem domain and provides a broad overview of visibility problems and algorithms in computer graphics grouped by the proposed taxonomy.

The visibility problem that this work aims to solve can be classified as global visibility, i.e., to identify which surfaces are invisible independently from the viewpoint of an observer placed outside the 3D model. The algorithms that address such problem aim to determine

a data structure able to yield the information about which parts of the geometry are invisible to an external observer. Developing efficient global visibility algorithms is still an open problem. A notable work in this field is presented by Durand et al. [16], where the authors propose a structure called visibility complex encoding all visibility relations in 3D space. Unfortunately, creating such structure is $\mathcal{O}(n^4 \log n)$, where n is the number of polygonal faces in the scene. Therefore, those methods are unfeasible for industrial applications and provide more information of what is really needed for the preprocessing purposes of this work. As explained in Section 2.3, the approach presented in this chapter is based on ambient occlusion computation through ray tracing, to estimate the visibility degree of the various faces to an external observer. Concerning ray tracing computations, several intersection algorithms have been developed over the years. According to Jiménez et al. [22], the most used algorithm to test ray-triangle intersection is the one introduced by Möller and Trumbore [18]. Then, slightly different versions have been proposed mainly aimed at taking advantage of specific hardware accelerations as done by Havel [23]. However, if the intersection point is not required, algorithms based on Plücker coordinates could be faster [24].

2.3 Problem Formulation

In this section, we describe the problem we aim to solve. We want to identify the visible faces of a 3D mesh by exploiting the concept of ambient occlusion in object space. As we will explain extensively in Section 2.4.1, we evaluate the visibility of a certain point on a face in relation to the amount of ambient light hitting that point. Our goal is to ensure theoretical convergence to the optimal result, i.e., all faces identified correctly. Moreover, we do not take any restrictive assumptions. Before going into theoretical and implementation details, it is useful to first introduce the core elements composing a 3D model and to provide a brief overview about the visibility problem.

2.3.1 Representation of 3D Objects

There are several ways to represent 3D objects in computer graphics. The most common one consists in considering an object as a collection of surfaces. Given the massive amount of different object types, several surface models have been developed over the years, with an increasing level of complexity according to the level of detail required. Figure 2.1 shows how a complex surface can be arbitrarily approximated using simple triangles.

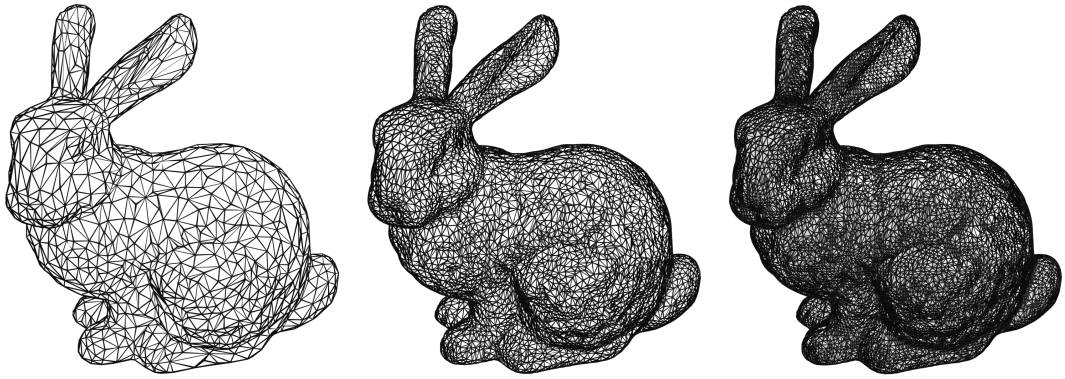


Figure 2.1: Different approximations of the "Stanford Bunny" using triangular meshes.

The resulting collection of surfaces is called polygonal mesh or simply mesh. Object rendering can be simplified and sped up using polygonal meshes since all surfaces can be described with linear equations. In particular, triangular meshes are preferred for their simplicity, numerical robustness and efficient visualization [25]. Polygonal meshes can be represented in a variety of ways by using the following core elements:

- vertex: a vector representing the position in 3D space along with further information such as colour, normal vector and texture coordinates;
- edge: the segment between two vertices;
- face: a closed set of edges, i.e., a triangle.

These elements are represented in Figure 2.2.

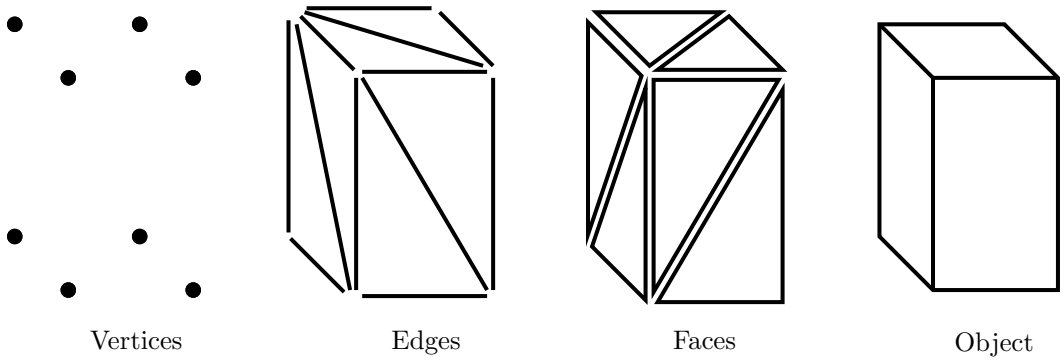


Figure 2.2: Representation of the core elements of a mesh.

This work aims to identify which faces of the 3D model are visible or not. By visible we mean there is at least one point outside the model for which there exists a segment, from that point to another on the triangle, that does not intersect any other.

2.3.2 Visibility Problem

The visibility problem has been one of the first major research topics in computer graphics and is the core of many state-of-the-art algorithms. This concept can be intuitively defined in terms of lines: two points A and B are mutually visible if no object intersects the line AB between them. From this simple notion, it is possible to address the visibility problem in different spaces regarding where points A and B lie [20].

- Image space: a 2D space for the visual representation of a scene. The rasterization process of converting a 3D scene into a 2D image works in this space. For this reason, the most common methods to solve the visibility problem perform their operations in 2D projection planes.
- Object space: a 3D space in which the scene is defined and objects lie. Methods developed within this space are computationally onerous and they are usually used to create proper data structure used to sped up subsequent algorithms. These acceleration structures are crucial for real time computer graphics applications such as video games.
- Line space: the one of all the possible lines that could be traced in a scene. Methods developed within this space try to divide the line space according to the geometry that a given line intercepts. Indeed, as stated at the beginning of this section, the visibility notation can be naturally expressed in relation to those elements.
- Viewpoint space: the one of all the possible views of an object. Theoretically, it could be partitioned into different regions divided by visual events. A visual event is a change in the topological appearance. For example, while rotating a coin vertically, at a certain point, one of its faces becomes visible while the other not. This process generates a structure referred in literature as aspect graph [26]. The latter has only a theoretical interest since it could have a $\mathcal{O}(n^9)$ complexity for general non convex polyhedral objects in a 3D viewpoint space, where n is the number of object faces. Nevertheless, a few visibility problems are defined and addressed in this space, such as viewpoint optimization for object tracking.

2.3.3 Problem Statement

In this work, the visibility of a certain point on a face is defined in relation to the quantity of ambient light hitting that point; this is the definition of ambient occlusion that we review in Section 2.4.1. Here, we address the visibility problem in object space and, without loss of generality, we consider triangular meshes. It is trivial to extend the results

presented in the following sections to a mesh composed of generic polygons. The problem we want to address can be formally cast as follows. Assume we have a 3D mesh $\mathcal{T}$ composed by N triangles, i.e., $\mathcal{T} = \{T_1, \ldots, T_N\}$. The triangle T_i is said to be visible, if there exist a ray starting from a point on T_i and proceeding to infinity that does not intersect any other triangle of the mesh. The goal is to determine, for $i = 1, \ldots, N$, if T_i is visible or not.

2.4 Proposed Approach: Ambient Occlusion, Visibility Index and Plücker Coordinates

The approach we propose to deal with the problem stated in Section 2.3.3 is based on the notion of ambient occlusion. Inspired by this definition, we introduce a visibility index which allows us to estimate if a triangle is visible or not. Basically, given a number of rays starting from points of a triangle, we try to determine if they intersect other triangles of the mesh. We will see that the intersection test can be done in an efficient way exploiting Plücker coordinates and the so-called side operator.

2.4.1 Ambient Occlusion

Ambient occlusion is a rendering technique used to improve the realism of a scene. It is an empirical technique introduced for the first time by Zhukov et al. to approximate the effects produced by global illumination systems [27]. The ambient occlusion models how a certain surface is illuminated by indirect light caused by the reflections of direct light over the various surfaces of the scene. Figure 2.3 shows the improvements that such technique brings to a rendered image.

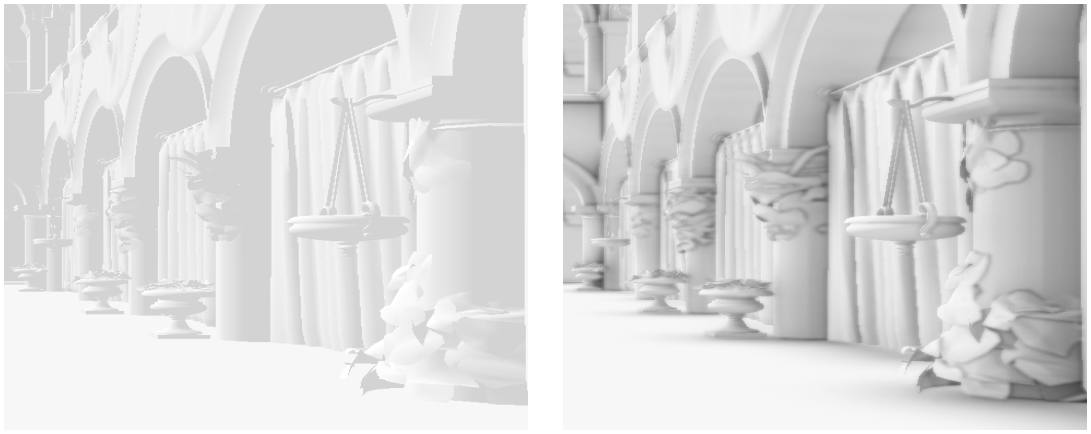


Figure 2.3: Visual comparison of a scene illuminated by ambient light. The image on the right shows how ambient occlusion estimation increases the realism of the scene. In particular, note that this method generates soft global shadows that contribute to the visual separation of objects.

The indirect light is approximated by considering an omnipresent, omnidirectional light source with fixed intensity and colour which is called ambient light. Then the exposure of a certain surface to this light is computed by looking at the geometrical properties of its surrounding. In other words, ambient occlusion is an approximation of the darkening which occurs when the ambient light is blocked by nearby objects. Formally, such term is defined for a point on the surface as the cosine-weighted fraction of the upper hemisphere where incoming light cannot reach the point. To be more precise, the ambient occlusion term for a point P on a face with normal n corresponds to the integral

$$AO(P) = \frac{1}{\pi} \int_{\omega \in \Omega} V(P, \omega) \cos\theta d\omega \tag{2.1}$$

where:

- P is the surface point;
- Ω is the upper hemisphere generated by the cutting of a sphere centred in P by the plane on which the surface lies;
- ω is a point of the hemisphere Ω and identifies the incoming light direction (with a slight abuse of language, in the following we will sometimes refer to ω as the ray direction);
- $V(P, \omega)$ is a function with value 1 if there is incoming ambient light from direction ω and 0 if not;

- $\frac{1}{\pi}$ is a normalization factor;

- θ is the angle between direction ω and the surface normal n (note also that $\cos\theta = \omega \cdot n$).

In Figure 2.4, a visual representation of the problem is presented.

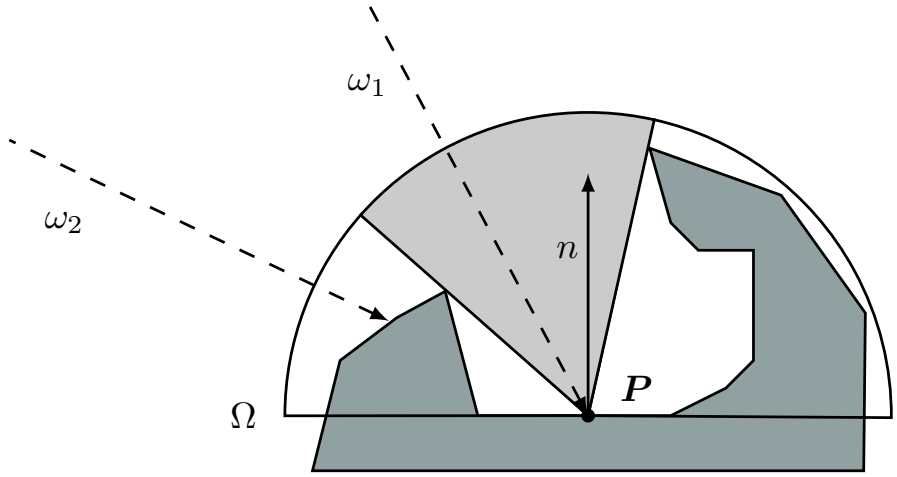


Figure 2.4: Visual representation of the ambient occlusion for a point P. For the rays in example, the function $V(P,\omega)$ has the following results: $V(P,\omega_2) = 0$, while $V(P,\omega_i) = 1$, $\forall\omega_i$ in the grey area of the hemisphere Ω, in particular for ω_1.

The integral in (2.1) is usually evaluated using Monte Carlo techniques by sampling the upper hemisphere in K points and, for each one, cast a ray to test for occlusions. Therefore, the integral is approximated by:

$$\widehat{AO}(P) = \frac{1}{K}\sum_{i=0}^{K-1} V(P,\omega_i) \tag{2.2}$$

with ω_i sampled with probability density function

$$p(\omega_i) = \frac{\cos(\theta_i)}{\pi} = \frac{\omega_i \cdot n}{\pi}\,. \tag{2.3}$$

Notice that normal n shall be oriented correctly towards the outside of the mesh for (2.3) to be a probability density.

Several different algorithms can be adopted to estimate the ambient occlusion: [28] reviews exhaustively the evolution of ambient occlusion techniques in recent years.

2.4.2 Visibility Index

We describe the strategy used to determine if a point P of a triangle of the mesh is visible or not. Suppose K rays, say $r_0, \ldots, r_{K-1}$, starting from point P have been generated. Notice that each ray r_i is well identified by its direction ω_i. Then, based on the notion of ambient occlusion and, in particular, on its Monte Carlo estimate in Equation (2.2), the visibility score of a point P can be expressed as

$$PV_{score}(P) = \frac{1}{K} \sum_{i=0}^{K-1} V(P, \omega_i) \tag{2.4}$$

where $V(P, \omega) = 1$ if the ray does not intersect any other triangle and proceed infinitely. Therefore, the point visibility score PV_{score} of a point P is the fraction of light-rays which are un-occluded. The choice for this visibility score seems well posed since an observer along the direction of such un-occluded rays is able to see the point P. By testing enough points $\{P_1, \ldots, P_M\}$ on a face T it is possible to estimate its visibility score SV_{score} as the mean of its points scores, giving

$$SV_{score}(T) = \frac{1}{M} \sum_{i=0}^{M-1} PV_{score}(P_i)\,. \tag{2.5}$$

After the computation of such score for all the faces of the object it is possible to select only the most visible ones applying a global threshold, effectively removing the internal geometry of the model. Actually, a zero threshold allows to select only the visible faces.

The most costly part of the algorithm is to determine the value of the function $V(P, \omega)$. This translates in determining if any given ray intersects any other triangle of the mesh along its path. In this study, this process has been developed using Plücker coordinates and tested against the widely known Möller-Trumbore ray-triangle intersection algorithm. In our implementation every ray is tested against all the other object faces until a first intersection is found.

2.4.3 Plücker Coordinates

Firstly introduced by the mathematician Julius Plücker in 19th century, these coordinates provide an efficient representation of oriented lines. Any real line can be mapped to the Plücker space. Since coordinates are homogeneous, any two points on the same oriented line will have the same Plücker coordinate up to a scaling factor. Let $P = (P_x, P_y, P_z)$ and $Q = (Q_x, Q_y, Q_z)$ be two distinct points in $\mathbb{R}^3$ which define an oriented line l that goes from Q to P. This line corresponds to a set l of six coefficients, called

Plücker coordinates of the line l

$$l = (l_0, l_1, l_2, l_3, l_4, l_5)$$

where the first three represent the direction of the line. Actually,

$$l_0 = P_x - Q_x, \quad l_1 = P_y - Q_y, \quad l_2 = P_z - Q_z,$$

while the other three components are given by the cross product between P and Q, giving

$$l_3 = P_y Q_z - P_z Q_y, \quad l_4 = P_z Q_x - P_x Q_z,$$
$$l_5 = P_x Q_y - P_y Q_x \,.$$

An important aspect of these coordinates, which has been crucial to this study, is the so-called side operator [29]. Such function characterizes the relative orientation of two lines. Given $l = (l_0, l_1, l_2, l_3, l_4, l_5)$ and $r = (r_0, r_1, r_2, r_3, r_4, r_5)$, the side operator is defined as:

$$\text{side}(l, r) = l^T W r \tag{2.6}$$

where

$$W = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ I_{3\times3} & 0_{3\times3} \end{bmatrix}.$$

Therefore, the side operator can also be written as

$$\text{side}(l, r) = l_3 r_0 + l_4 r_1 + l_5 r_2 + l_0 r_3 + l_1 r_4 + l_2 r_5 \,. \tag{2.7}$$

Two oriented lines l and r can interact in space in three different ways:

1. if l intersects r, then $\text{side}(l, r) = 0$;

2. if l goes clockwise around r, then $\text{side}(l, r) > 0$;

3. if l goes counter-clockwise around r, then $\text{side}(l, r) < 0$.

Such cases are highlighted in Figure 2.5.

Plücker coordinates can be defined in other less intuitive ways and possess other properties that are not used in this work. Also note that not all the Plücker points define a real line in 3D space. To do so the Plücker point l have to satisfy the condition $\text{side}(l, l) = 0$.

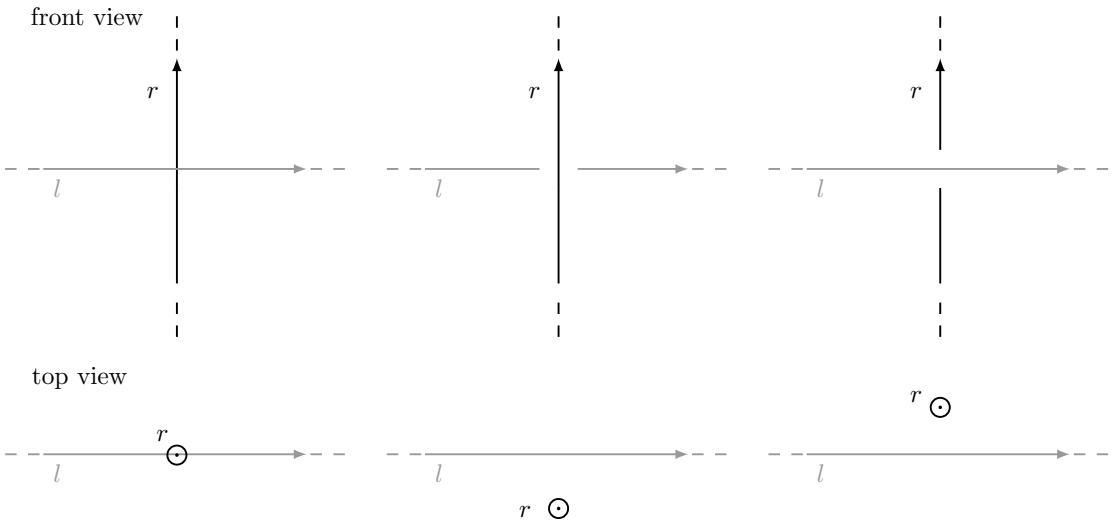


Figure 2.5: Possible line configurations in space: on the left l and r intersect, in the middle l goes clockwise around r while on the right l goes counter-clockwise around r.

2.5 Visibility Algorithm Based on Plücker Coordinates

This section starts with an overview of the main steps of the algorithm and, then, describes the implementation details. First, we want to emphasize that typically, in computer graphics, it cannot be assumed that triangles normals are always directed outwards the mesh. Actually, if only the upper hemisphere is considered, like in Figure 2.4, the procedure may create false occlusions. To avoid such scenario, it is sufficient to consider the whole sphere instead. A false occlusion example is presented in Figure 2.6.

The algorithm may be divided in three major steps for each mesh triangle T_i for which to compute the visibility.

Initialization

We select a set of M points, say $\mathcal{P} = \{P_1, \ldots, P_M\}$, on T_i. For each point $P_i \in \mathcal{P}$, we generate K rays as follows: we first create a sphere centred in P_i, then we sample K points on the surface of this sphere and, finally, we draw the K rays starting from P_i and passing through the previously sampled points. Notice that we generate in total MK rays. How selecting the points on T_i and sampling the sphere is described in Sections 2.5.1 and 2.5.2, respectively.

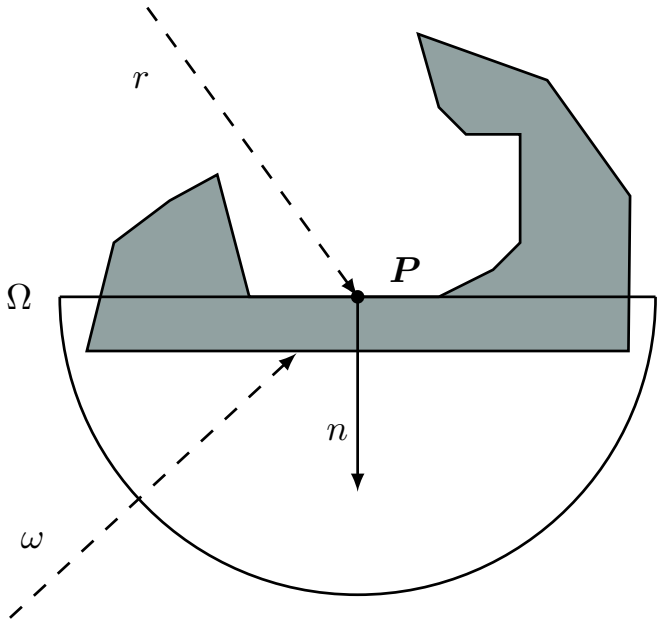


Figure 2.6: Example of a wrongly oriented normal that results in a false occlusion. By considering only the hemisphere Ω oriented along the normal n, the classification gives a wrong result: the ambient occlusion term is zero. Indeed $\forall \omega \in \Omega$ we have an occlusion but there is at least one un-occluded ray r that reaches P.

Score computation

Each triangle $T_j \neq T_i$ is tested against all the rays generated at the previous step to see if an intersection occurs. The ray-triangle intersection test is performed in Plücker space as it will be explained in Section 2.5.3. If a ray intersects a triangle T_j, then it is removed from the set of generated rays. Once the above intersection testing phase has been completed, the visibility score of T_i is computed as

$$SV_{score}(T_i) = \frac{\text{remaining rays}}{MK}\,. \tag{2.8}$$

Classification

If $SV_{score}(T_i) > 0$ then T_i is classified as visible, otherwise as invisible.

A couple of remarks are now in order.

Remark 1. *Typically, in computer graphics, meshes consist of a large number of triangles that are very small compared to the size of the mesh itself. Based on this fact, in order to reduce the computational burden of the proposed algorithm, in the initialization step we might consider only the barycenter of T_i, in place of selecting M points; by doing that,*

we generate only K rays per triangle instead of MK. Our numerical tests show that such choice does not affect the overall result.

Remark 2. *Note that it is possible to select triangles that are barely visible using a non-zero threshold $\delta \geq 0$ on the score. Formally, T_i is invisible or barely visible if $SV_{score}(T_i) \leq \delta$.*

Next, we explain more in detail how to select the M points on a face of the mesh, how to generate K rays on a sphere and how to compute the intersections of these rays with mesh triangles.

2.5.1 Sampling Points on Triangle

The points on a triangle T_i can be efficiently selected by taking random points at minimum distance or uniformly distributed.

Uniform distribution

Arvo [30] presents a procedure for deriving an area-preserving parametrization from a unit square $[0,1] \times [0,1]$ to a given bounded 2-manifold $\mathcal{M} \subset \mathbb{R}^n$. Using those results it is possible to uniformly sample any bounded surface, in particular a triangle, by having two independent random variables uniformly distributed over the interval $[0,1]$.

Minimum distance

Random samples are generated on a rectangle containing the triangle T_i using the Fast Poisson Disk Sampling method [31]. Then, following the rejection-sampling approach, only points that belong to that triangle are kept. The benefit of this algorithm is to apply a constraint on the minimum distance between the generated points, solving the typical clustering issue of uniform sampling methods.

The uniform distribution method is certainly easier to implement and faster to compute, although it tends to generate clusters of points. On the other hand, the minimum distance approach provides a more distributed set since it imposes a constraint on mutual distances. Therefore, to generate a few points, we recommend using the second method.

2.5.2 Rays Generation through Sphere Sampling

A key feature for a VSD algorithm is the ability to uniformly explore the surrounding space. It is, in fact, very important to select points on the surface of a sphere as uniform

as possible. The two most common methods to generate uniformly distributed points on a spherical surface are the following two.

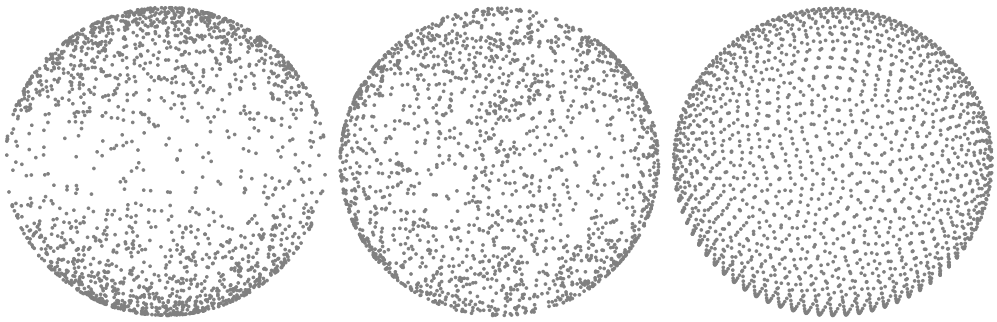
Uniform distribution

An easy method to uniformly pick random points on an n-dimensional sphere is presented in [32]. It is sufficient to generate an n-dimension vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^T$ whose x_i elements are independent and identically distributed samples of a Gaussian distribution. Then, a point P on the sphere is given by $P = \frac{\boldsymbol{x}}{||\boldsymbol{x}||_2}$.

Fibonacci lattice distribution

By using the notion of golden ratio and golden angle, both deriving from the Fibonacci sequence, it is possible to generate a set of samples at the same distance from their neighbours [33, 34]. In fact, the golden angle optimizes the packing efficiency of elements in spiral lattices.

Applying one of the following algorithms on a unitary sphere centred in the axis origin is equivalent to generate a collection of rays directions. Recalling that the visibility score for a given face T_i is defined as the fraction of un-occluded rays over the total rays created, it is important to notice that the choice of one algorithm in place of another will change the value and the meaning of that visibility score. The uniform distribution is certainly easier to implement than Fibonacci's one but tends to generate clusters when the number of sampled points is small. On the other hand, Fibonacci lattice provides greater uniformity even in the case of a few points. Figure 2.7 shows the sampling results obtained using the two methods described above. The figure also shows the result obtained with the cosine-weighted distribution [30]. The latter is the only one that allows to correctly estimate the ambient occlusion value as defined is Section 2.4.1. However, we can observe how this choice is not suitable for our aim since it tends to accumulate points near the poles and, consequently, rays are not well distributed. Notice that we are interested to evaluate the occlusion determined by homogeneously distributed rays rather than mathematically estimate the ambient occlusion term. Therefore, from our tests, we conclude that Fibonacci lattice provides the best distribution for our goal.

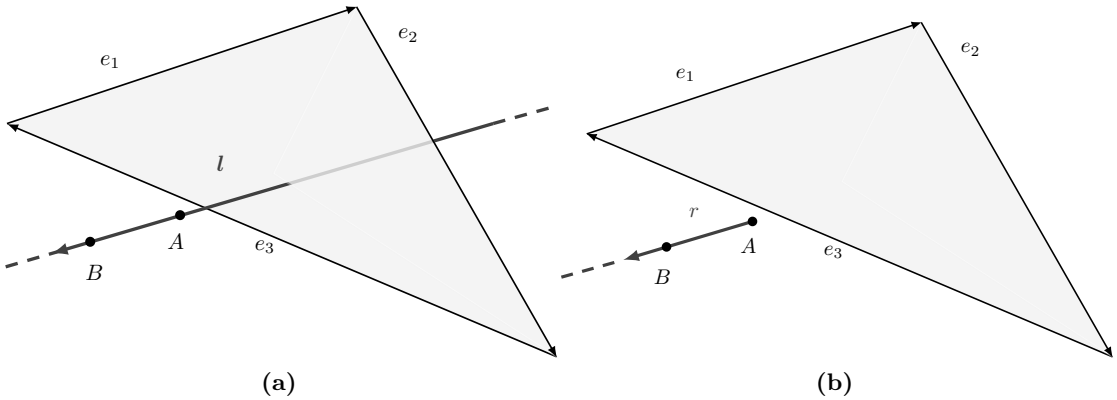


(a) **(b)** **(c)**

Figure 2.7: Comparison of three different points distributions on the surface of a sphere: (**a**) Cosine-weighted, (**b**) Uniform, (**c**) Fibonacci.

2.5.3 Ray Intersection Algorithm via Plücker Coordinates

We are now interested to determine the intersection between a ray and a triangle in Plücker space. This is not trivial since, in this space, it is not possible to define rays but only lines. Recall that a line extends infinitely in both directions while a ray is only a portion of it, starting from a point and going to infinity. This implies the inability to directly use a line-polygon intersection test to search for intersections between a ray and a polygon. Figure 2.8 shows the difference between the results obtained using line and ray intersection tests. In this section, we first describe Plücker line-polygon intersection test and, then, we modify it to support ray-polygon intersection.



(a) **(b)**

Figure 2.8: Comparison between line and ray intersection tests. (**a**) Considering the line passing through A and B, there is an intersection with the polygon. (**b**) Considering the ray starting in A and passing through B, there is no intersection.

As introduced in Section 2.4.3, Plücker coordinates can be used to define an efficient line-polygon intersection test thanks to the side operator. Actually, a line hits a polygon

if and only if it hits one of the polygon edges, or goes around clockwise or counter-clockwise to all the edges [35]. Figure 2.9 shows two examples of such test. The convention chosen is to define a polygon as a sequence of vertices $\{X_1, \ldots, X_N\}$ in such a way that the direction of the edges e_i is defined from X_i to X_{i+1}; in particular, the last edge goes from X_N to X_1.

Consider the edges e_i of a convex polygon with $i = \{1, \ldots, m\}, m \in \mathbb{N} \geq 3$ and a line l. The line l intersects the polygon if and only if

$$\forall e_i,\ \text{side}(l, e_i) \geq 0 \quad \text{OR} \quad \forall e_i,\ \text{side}(l, e_i) \leq 0\,.$$

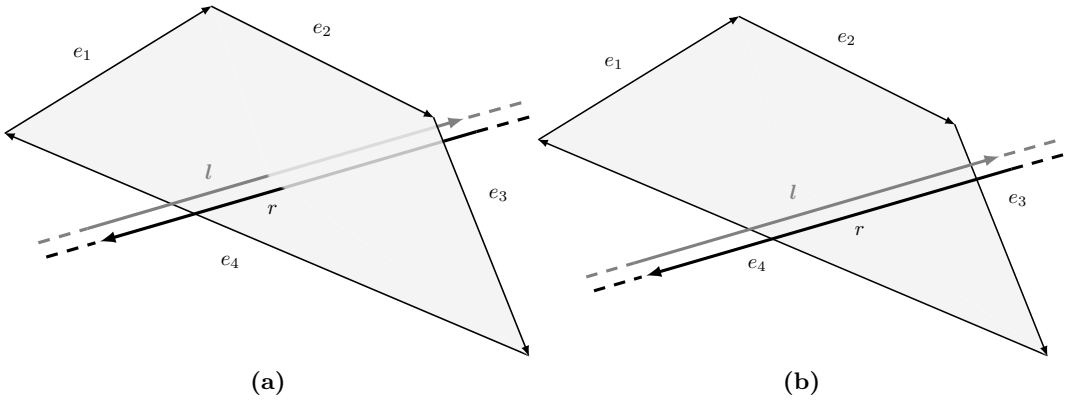


Figure 2.9: (**a**) Lines stabbing a convex polygon in 3D space. Note that the line $\boldsymbol{l}$ goes counter-clockwise around all the edges $\boldsymbol{e}_i$, while $\boldsymbol{r}$ clockwise. (**b**) Lines missing a convex polygon in 3D space. Note that the line $\boldsymbol{l}$ goes counter-clockwise around the edges $\boldsymbol{e}_1$ and $\boldsymbol{e}_4$, while clockwise around $\boldsymbol{e}_2$ and $\boldsymbol{e}_3$. The opposite happens with $\boldsymbol{r}$.

After describing the line-polygon intersection test method, we show how it is possible to use it as a ray intersection test [36, 37].

First of all, consider a plane π containing the ray origin P. Observe that this plane divides the 3D space in two half-spaces H_f and H_b, where we assume the former contains the ray trajectory. Now, let n be the versor orthogonal to π starting from P and contained in H_f. Then, the two half-spaces are formally defined as:

$$H_f = \{x \in \mathbb{R}^3 : n \cdot (x - P) > 0\}$$
$$H_b = \{x \in \mathbb{R}^3 : n \cdot (x - P) < 0\}\,.$$

As shown in Figure 2.10, ray-polygon intersection test is equivalent to the line-polygon one when considering only the geometries in H_f. Figure 2.10 also shows the challenging situation in which the vertices of a polygon may not belong entirely to one half-space.

In this condition, the polygon shall be clipped into two sub-polygons, each one belonging to the corresponding half-space. For instance, with reference to Figure 2.10, triangle T_4 is split into the trapezoid T_4^f and the triangle T_4^b. Only the trapezoid T_4^f will be processed in the ray intersection test. This clipping procedure is explained in Section 2.5.3.

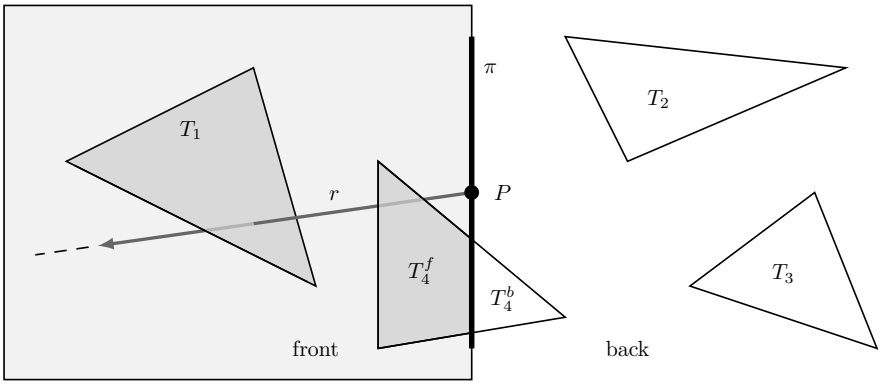


Figure 2.10: A ray-polygon intersection can be seen as a line-polygon one by considering only front geometries. Notice the case of the polygon T_4 that overlaps the plane π: in this case a clipping procedure is needed to subdivide it in T_4^f and T_4^b.

Clipping

The aim of this procedure is to determine the intersection points between the edges of the triangle and the plane that divides it. As we can see in Figure 2.11, to identify the polygons T_4^f and T_4^b in which the triangle is divided, we need to compute the intersection points I_1 and I_2 with the plane π.

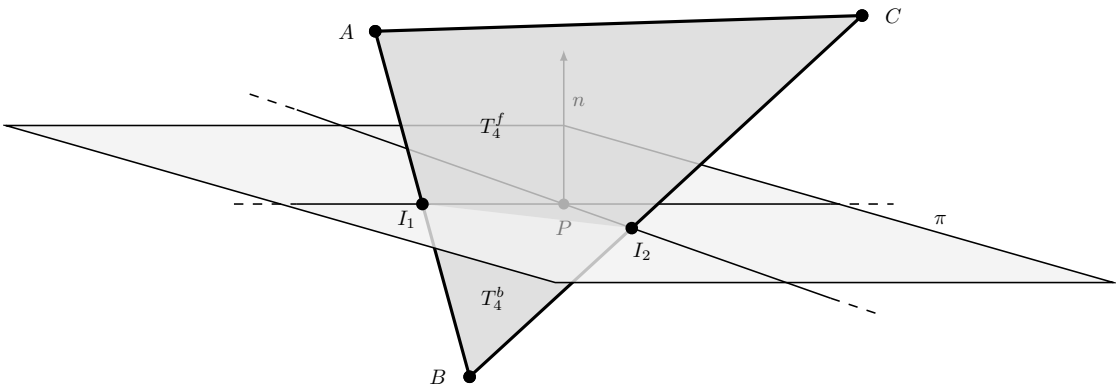


Figure 2.11: A triangle clipping example. The points I_1 and I_2 represent the intersections between the edges of the triangle ABC and the plane π. The trapezoid of vertices AI_1I_2C is considered in front of the plane, while the triangle I_1BI_2 behind it.

Consider the intersection point I_1. With reference to Figure 2.12, it is possible to

classify the points A and B to be either in front, back or on the plane π defined by the normal n and a point P.

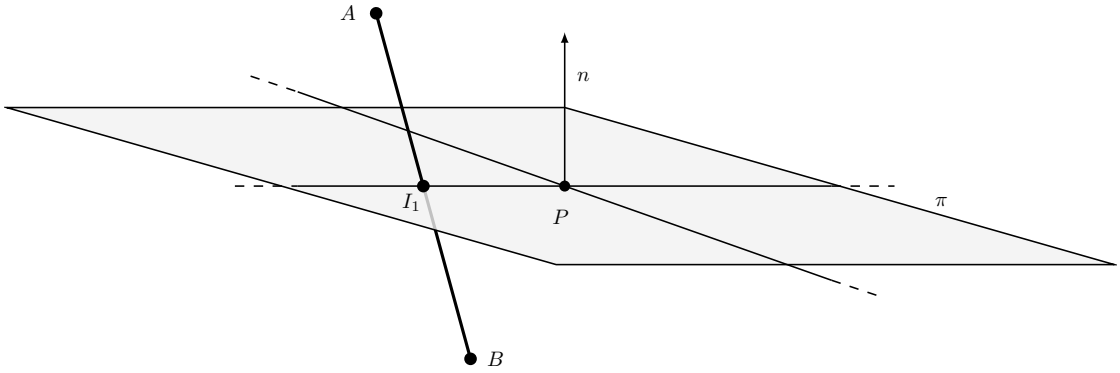


Figure 2.12: Segment clipping example. The point I_1 is the intersection between the segment AB and the plane π. The segment AI_1 is considered in front of the plane, while BI_1 behind it.

The classification can be done by looking at the signs of d_A and d_B, which are the scalar products between n and the vectors $A - P$ and $B - P$, respectively:

$$d_A = n^T(A - P)\,, \quad d_B = n^T(B - P)\,.$$

If sign $(d_A) > 0$, the point A is in front of the plane while, if sign $(d_A) < 0$, on the back and, if $d_A = 0$, the point lies on the plane. From those observations, it is clear that an intersection between the segment AB and the plane π is only possible if the following condition holds:

$$\text{sign}(d_A) \neq \text{sign}(d_B) \quad \text{OR} \quad \text{sign}(d_A) \cdot \text{sign}(d_B) = 0\,.$$

Note that the intersection point I_1 between segment AB and plane π can be expressed as

$$I_1 = A + t(B - A),$$

where $t \in [0, 1]$. It is possible to compute the value of t by observing that the vector $I_1 - P$ lies on the plane and, therefore, $n^T(I_1 - P) = 0$. We obtain:

$$\begin{aligned} 0 &= n^T(I_1 - P) = n^T(A - P) + t\, n^T(B - A) \\ &= n^T(A - P) + t\, n^T(B - P + P - A) \\ &= d_A + t(d_B - d_A)\,. \end{aligned}$$

This implies:

$$t = \frac{d_A}{d_A - d_B}\,.$$

Finally, we obtain:

$$I_1 = A + \frac{d_A}{d_A - d_B}(B - A)\,.$$

In a similar way we can compute the intersection point I_2. By using the above procedure, it is possible to divide any polygon w.r.t. an arbitrary plane π in 3D space into its front and back geometries. For example, by assuming that the polygon has three vertices, the only case in which clipping is required is when a vertex belongs to a different half-space of the others. Without loss of generality, we assume that $B \in H_b$, while $A, C \in H_f$. The clipping procedure can be summarized with the following points:

- Compute intersection points I_1 and I_2 of segments AB and BC with the plane π using the procedure described above.
- Generate two sub-polygons: the triangle $I_1BI_2 \in H_b$ and the trapezoid $AI_1I_2C \in H_f$.

Figure 2.11 shows the result of this procedure.

In our implementation, given a point P_j on T_i and a pencil of K rays starting from P_j, the clipping procedure is applied using the same plane for all the rays; specifically the plane used is the one containing the triangle T_i.

2.5.4 Proposed Approach Implementation and Its Convergence Properties

The implementation of the strategy we described in the previous sections, is reported in Algorithm 1.

Observe that, based on the proposed procedure, a non-visible triangle will be always classified as non visible, while a visible triangle might be erroneously classified as non-visible. However, it is possible to see that by increasing M and K, i.e., the number of points randomly selected on any triangle and the number of rays generated starting from any selected point, respectively, the misclassification error decreases. This fact is formally stated in the following propositions.

Algorithm 1 VSD based on Plücker coordinates

Input: 3D mesh $\mathcal{T} = \{T_1, \dots, T_N\}$

Implementation: For each T_i, the following actions are performed in order:

1. select M points, $P_1, \dots, P_M$, on T_i (see Section 2.5.1);
2. for each point P_h, $h = 1, \dots, M$, generate K rays (see Section 2.5.2);
3. for each of the MK rays generated at the previous point, perform the ray intersection test against all the triangles $T_j \neq T_i$ (see Section 2.5.3); if a ray intersects T_j, then it is removed from the set of generated rays;
4. compute the visibility score of triangle T_i as in (2.8).

Classification: For $i \in \{1, \dots, N\}$, triangle T_i is classified as visible is $SV_{\text{score}}(T_i) > 0$, otherwise as non visible.

Proposition 2.5.1. *Consider Algorithm 1. Assume that*

- *the M points in step (1) are selected either uniformly random or adopting the minimum distance approach; and*
- *the K rays in step (2) are generated uniformly random.*

Then, for M and K going to infinity, that is, $M, K \to \infty$, the probability that a visible triangle is classified as non visible goes to 0.

Proof. Let $\mathcal{S}$ be a sphere that contains the entire mesh. Consider a visible triangle T_i. The fact that T_i is visible means that there exists a point Q_1 on T_i and a point Q_2 on $\mathcal{S}$ such that the segment connecting Q_1 to Q_2 does not intersect any other triangle of the mesh. Let $\ell_{Q_1Q_2}$ be the line passing through Q_1 and Q_2 and let $\mathcal{C}^r_{\ell_{Q_1Q_2}}$ be the infinitely long cylinder of radius r having as axis the line $\ell_{Q_1Q_2}$. Accordingly, define

- $\mathcal{I}_1^r$ to be the region obtained by intersecting $\mathcal{C}^r_{\ell_{Q_1Q_2}}$ with T_i; and
- $\mathcal{I}_2^r$ to be the region obtained by intersecting $\mathcal{C}^r_{\ell_{Q_1Q_2}}$ with $\mathcal{S}$ and containing Q_2.

Notice that, for $r > 0$, both $\mathcal{I}_1^r$ and $\mathcal{I}_2^r$ are connected regions of non-zero measure.

Since T_i is visible, it follows that there exists $\bar{r} > 0$ such that the portion of $\mathcal{C}^{\bar{r}}_{\ell_{Q_1Q_2}}$ included between $\mathcal{I}_1^{\bar{r}}$ and $\mathcal{I}_2^{\bar{r}}$ does not intersect any triangle of the mesh; in turn, this fact is true also for any segment connecting a point in $I_1^{\bar{r}}$ to a point in $I_2^{\bar{r}}$.

Since M goes to ∞, the probability of randomly picking a point P_j on T_i belonging to $\mathcal{I}_1^{\bar{r}}$ goes to 1 (either using the uniform selection or the minimum distance approach).

Moreover, since also K goes to ∞, the probability of selecting a ray starting from a given point P_j that crosses $\mathcal{I}_2^{\bar{r}}$ goes to 1. Hence, for $M, K, \to \infty$, T_i will be classified as visible with probability that goes to 1. This concludes the proof. □

Proposition 2.5.2. *Assume that*

- *the M points in step (1) are selected either uniformly random or adopting the minimum distance approach; and*
- *the K rays in step (2) are generated according to the Fibonacci lattice distribution.*

Then, there exists $\bar{K} > 0$, such that, if $K \geq \bar{K}$ and $M \to \infty$, then the probability that a visible triangle is classified as non visible goes to 0.

Proof. Consider a visible triangle T_i. Let $\mathcal{I}_1^{\bar{r}}$ and $\mathcal{I}_2^{\bar{r}}$ be as in the proof of Proposition 2.5.1. As observed previously, since M goes to ∞, the probability of randomly picking a point P_j on T_i belonging to $\mathcal{I}_1^{\bar{r}}$ goes to 1 (either using the uniform selection or the minimum distance approach). Moreover, adopting the Fibonacci lattice distribution, there exists $\bar{K}_i$, such that, if $K \geq \bar{K}_i$, then, for any point P_j selected on T_i, Algorithm 1 will generate at least one ray starting from P_j that crosses the region $\mathcal{I}_2^{\bar{r}}$. This implies that, for $K \geq K_i$ and $M \to \infty$, triangle T_i will be correctly classified with probability 1. To conclude the proof, it suffices to define

$$\bar{K} := \max \{K_i \, : \, T_i \text{ is visible}\} \, .$$

□

Based on Remark 1, we may provide a modified version of Algorithm 1 where step (1) can be simplified by considering only the barycentre of the triangle instead of sampling it. Specifically, step (1) can be substituted by the following step: (1') Compute the barycenter of T_i. Clearly, Proposition 2.5.1 and Proposition 2.5.2 are not still valid when implementing step (1') instead of step (1). However, in our numerical tests described in Section 2.6, we have implemented this modified version of Algorithm 1 and the obtained results show that this choice does not affect the overall result. This fact is not surprising since we have considered the typical case where the size of each triangle is very small as compared to the size of the entire mesh.

2.6 Results

This section presents a series of numerical tests to evaluate the proposed algorithm efficiency and robustness. First we compare the performance of the ray-triangle intersection

test obtained using our Plücker space-based approach with the use of the Möller-Trumbore algorithm [18], that represents the state-of-the-art in this scenario. We then compare our solution with a state-of-the-art method, based on ambient occlusion estimate in the image space, which is implemented in MeshLab; see [17] for all the implementation details. To carry on all of these tests, we used a midrange PC equipped with an Intel i3-6100 CPU, 8 GB DDR4 RAM, and a AMD RX-480 4 GB GPU with 2304 stream processors. In particular, we implemented all the algorithms using high-level shading language (HLSL). This allows us to take full advantage of the GPU power which is made specifically for computer graphics tasks.

2.6.1 Intersection Algorithms Comparison

The algorithm described in Section 2.5 is heavily based on the intersection test between rays and triangles. Therefore, the computation complexity strongly depends on the optimization of this test. For that reason, we decide to compare the performance of Plücker-based ray-triangle intersection algorithm with the Möller-Trumbore one. To compare the two methods we designed a numerical test that can be summarized in the following steps.

1. Generate a set of triangles $\mathcal{T}$ randomly within a cubic volume of $1\,\mathrm{m} \times 1\,\mathrm{m} \times 1\,\mathrm{m}$.
2. Create a set of points $\mathcal{S}$ by sampling the surface of a sphere, with radius $2\,\mathrm{m}$, centred in the cubic volume, using the Fibonacci distribution.
3. For each point $S \in \mathcal{S}$, generate a ray starting from the volume centre and passing through S, thus generating a set of rays $\mathcal{R}$.
4. For each ray in $\mathcal{R}$, check if it is occluded by at least one triangle in $\mathcal{T}$.

Finally, the algorithm yields the following set:

$$\mathcal{V} = \{r \in \mathcal{R} : \exists\, t \in \mathcal{T} : \texttt{intersects}(r, t)\}$$

where $\texttt{intersects}(r, t)$ is either computed with Möller-Trumbore method or the Plücker-based one. Of course, the two methods produce the same set $\mathcal{V}$ but the number of operations they require is different. To provide a more reliable result, we reproduced all the tests 10 times by averaging the partial results. Moreover, since there are two parameters in the test algorithm, i.e., the number of sampled points in the sphere surface ($|\mathcal{S}|$) and the number of generated triangles ($|\mathcal{T}|$), we decide each time to fix one parameter and change the other to allow a more detailed and clear reading. Figure 2.13 shows

the time spent to run the two algorithms and the ratio between them obtained using 10,000 triangles and a different number of rays. We have chosen this particular value because it allows all GPU cores to work full-throttle. Actually, Figure 2.14 shows that, by using a small number of triangles, some GPU cores remain in idle state. This figure, indeed, shows a performance comparison obtained using 1000 rays and a different number of triangles.

From the results, it is clear how Plücker-based algorithm always offers better performance than the Möller-Trumbore one in the cases tested. Indeed, the latter is about 3 times slower at the same load. An intuitive reason is related to the nature of the two algorithms. At each call of the intersection test, Möller-Trumbore must perform four dot products, three cross products, and three vectorial subtractions [18]. On the other hand, the method based on Plücker coordinates only needs 2 dot products and 1 vectorial subtraction. Although Plücker's algorithm needs a challenging initialization, it allows to reuse some valuable information about the triangle that in Möller-Trumbore must be calculated on-the-fly at each iteration. Therefore, since we compare the same triangle with many different rays, the pre-processing done by Plücker's algorithm allows us to save valuable time in the long run. Finally, we want to emphasize that Möller-Trumbore also computes the intersection point between a ray and a triangle. However, this information is totally irrelevant for our purposes.

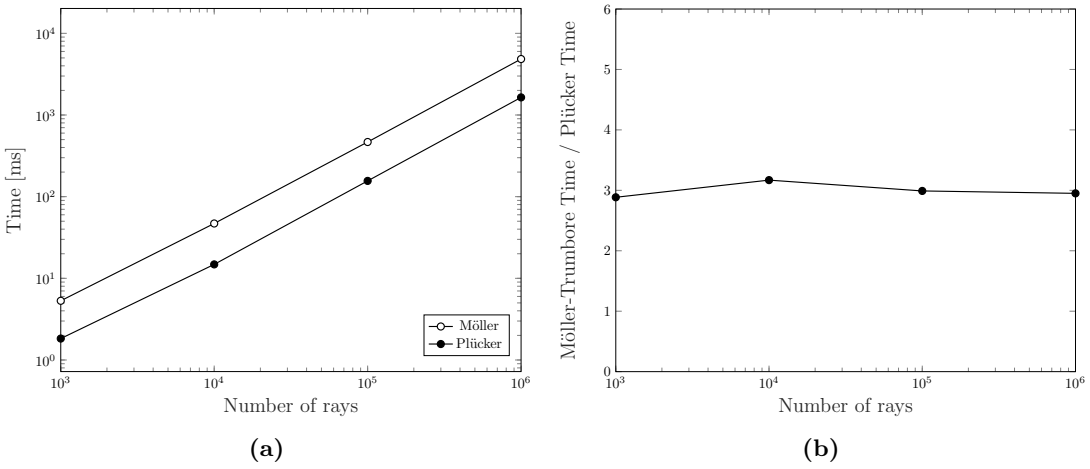
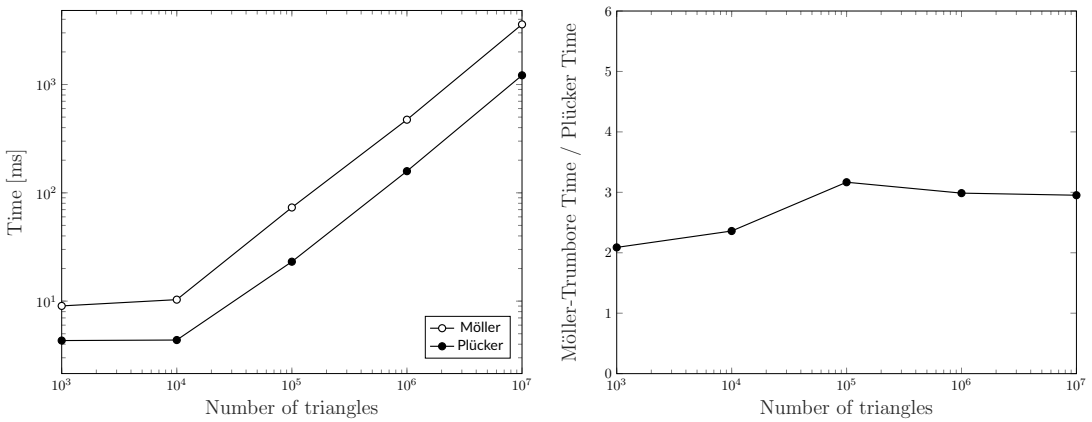


Figure 2.13: Computation time comparison for intersecting 10,000 triangles with an increasing number of rays using Möller-Trumbore and Plücker-based algorithms. (**a**) Computation time, (**b**) Ratio between the computation times of the two algorithms.



(a) **(b)**

Figure 2.14: Computation time comparison for intersecting an increasing number of triangles with 1000 rays using Möller-Trumbore and Plücker-based algorithms. (**a**) Computation time, (**b**) Ratio between the computation times of the two algorithms.

2.6.2 Comparison with a State-of-the-Art Method

In this section, we compare the results obtained using the proposed algorithm with the ones provided by a state-of-the-art VSD method. The latter is based on the ambient occlusion estimate in image space and it is implemented within an open-source software called MeshLab [17]. MeshLab is widely used in computer graphics for processing and editing 3D triangular meshes. Among the many operations that this software offers, it provides visibility surface determination based on a custom version of the screen-space ambient occlusion (SSAO) [38]. This technique is typically used to improve the realism of a scene during rendering. Since it works in image space, it is characterized by a lower computational complexity. On the other hand, the dependence on the rendered image resolution limits the result quality. This limitation does not occur by estimating the ambient occlusion in object space using a ray tracing technique as presented in this work. The VSD method provided by MeshLab consists of the following steps.

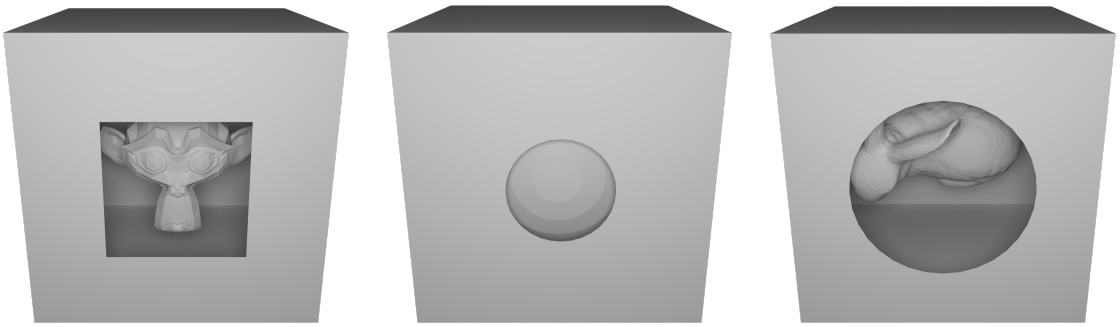
1. Given a number of views V, V points are sampled on the surface of a sphere that fully embrace the triangular mesh using the Fibonacci distribution. The set $\mathcal{R}$ of camera directions contains all the rays starting from each sampled point and directed to the sphere origin.

2. For each direction $r \in \mathcal{R}$, the depth map of the corresponding camera view is computed. By using this map the ambient occlusion term is computed for each pixel. Each value is then added to the correspondent projected triangle vertex.

3. For each vertex, the partial results obtained for each direction r, i.e., for each view, are averaged to obtain a unique global value.

After this process, it is possible to identify the invisible triangles by selecting those that have at least one occluded vertex of the three, i.e., with a zero ambient occlusion value. Obviously, the approximation improves as the number of views increases. With this brief description of the algorithm, that can be found at MeshLab GitHub repository [39], we want to highlight a couple of key points that distinguish the two approaches. As mentioned earlier, MeshLab VSD method works in the image space. This makes it significantly faster than the algorithm we propose but less robust, due to the dependence on the image resolution used when rendering depth maps. The second main limit regards the assumption that all triangles normals must be oriented towards the outside of the mesh. Actually, the method implemented in MeshLab is directly based on the notion of ambient occlusion presented in Section 2.4.1 and, therefore, it relies on triangle normals. The following numerical results show that, if triangles normals are flipped, the resulting estimate is totally meaningless. Actually, in the industrial domain, it may happen to observe models with some normals that are wrongly oriented. However, for the first tests, we have considered models with all the normal properly oriented.

To verify and compare performance in a reproducible way, we decided to create three different 3D models. As shown in Figure 2.15, a monkey head, a ball and a Stanford Bunny are respectively inserted in a box characterized by a hole in the front side. Since we want to test the ability to detect hidden triangles, we decided to design these ad-hoc models which perfectly fit an ideal scenario.



(a) **(b)** **(c)**

Figure 2.15: An overview of the 3D models used for numerical tests. (**a**) Sample 1 consists of 986 triangles, (**b**) Sample 2 of 1006 triangles and (**c**) Sample 3 of 9297 triangles.

We start by comparing the time needed for the two algorithms to estimate the visibility for each face of the sample meshes. From Figure 2.16, we can notice how the results are

fully expected due to the significant difference in complexity. Actually, we can infer that the computational complexity of MeshLab algorithm is $\mathcal{O}(VN)$, where V is the number of views and N the number of triangles. Indeed, the depth map calculation complexity is proportional to the number of triangles. As far as Algorithm 1 is concerned, we have implemented it by considering the modified step $(1')$ and the Fibonacci lattice distribution method; it can be seen that has a complexity of $\mathcal{O}(KN^2)$ where K is the number of rays tested for each triangle.

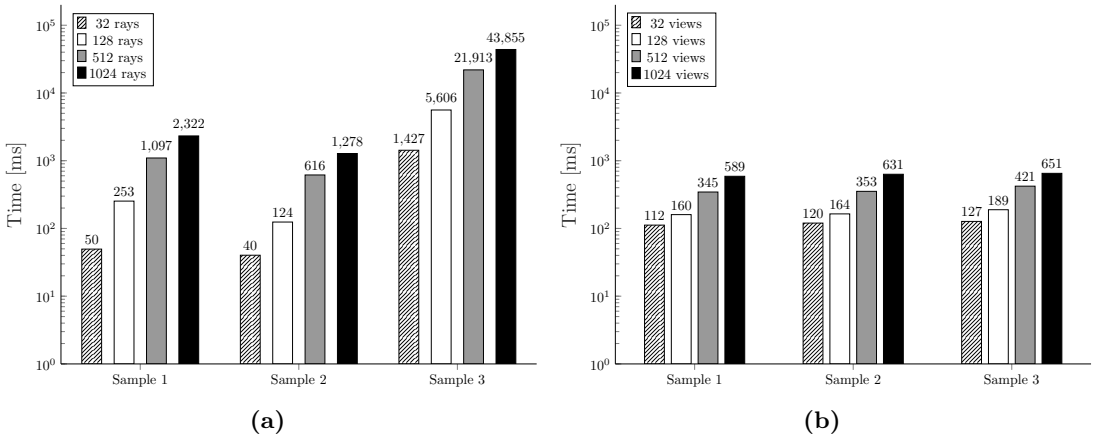


Figure 2.16: Computation time comparison of the two algorithms using different settings and samples. (**a**) shows the results obtained with Plücker VSD, (**b**) shows the results obtained with MeshLab VSD.

However, the time needed to perform VSD is not of particular interest for this work. In fact, recall that these algorithms are usually used in computer graphics only once to lighten 3D models. We aim for a robust method and, therefore, we are more interested in evaluating a performance index such as the number of incorrect classifications. Given the set of triangles $\mathcal{T}$, we consider as misclassification error the sum of false positives and negatives divided by the number of triangles. Since we test if a triangle is invisible, false positives correspond to visible triangles that the algorithm identifies invisible. Consider the set $\mathcal{I}_A$ of triangles classified as invisible by the algorithm and the set $\mathcal{I}_G$ of truly invisible triangles which is known a priori. The misclassification error ME can be written as:

$$\mathrm{ME} = \frac{|\{t \in \mathcal{I}_A : t \notin \mathcal{I}_G\}| + |\{t \in \mathcal{I}_G : t \notin \mathcal{I}_A\}|}{|\mathcal{T}|}\,. \tag{2.9}$$

Notice that, as discussed in Section 2.5.4, false negatives cannot exist by construction

of the algorithm, i.e., $|\{t \in \mathcal{I}_G : t \notin \mathcal{I}_A\}| = 0$. Therefore, (2.9) can be simplified obtaining

$$\text{ME} = \frac{|\{t \in \mathcal{I}_A : t \notin \mathcal{I}_G\}|}{|\mathcal{T}|} .$$

We want to emphasize our interest in achieving the maximum lossless compression in terms of number of triangles. Since the maximum compression is reached when $\mathcal{I}_A = \mathcal{I}_G$, by minimizing ME we are maximizing the lossless compression.

Figure 2.17 shows the misclassification rates obtained using a different number of rays, or corresponding views, for each of the designed mesh samples. From these plots we can observe how, as the number of rays or views increases, the misclassification rate decreases. This is widely expected since, using fewer views or rays per triangle, it is more likely to misidentify a visible triangle as invisible and, thus, increase the number of false positives. Intuitively, having more views or rays per triangle allows the algorithm to find even the most hidden triangles, getting closer to the optimum value.

From Figure 2.18 we can observe that our algorithm tends to the optimal faster than the method in comparison. Actually, as described previously, MeshLab VSD method operates in image space and it is constrained to a maximum resolution of 2048 $\times$ 2048 pixels [39]. As a result, even if the number of views increases infinitely, this method could never reach the optimal since sphere mapped points would have a finer quantization than the rendered image. This explains why the improvement in accuracy is so slow after a certain number of views. To be more precise, we report in Table 2.1 the evolution of the two misclassification rates with an increasing number of rays or views. In particular, we can notice that, using 10,000 rays, our algorithm is able to converge to the optimum while, MeshLab algorithm, stabilizes around 1% of error.

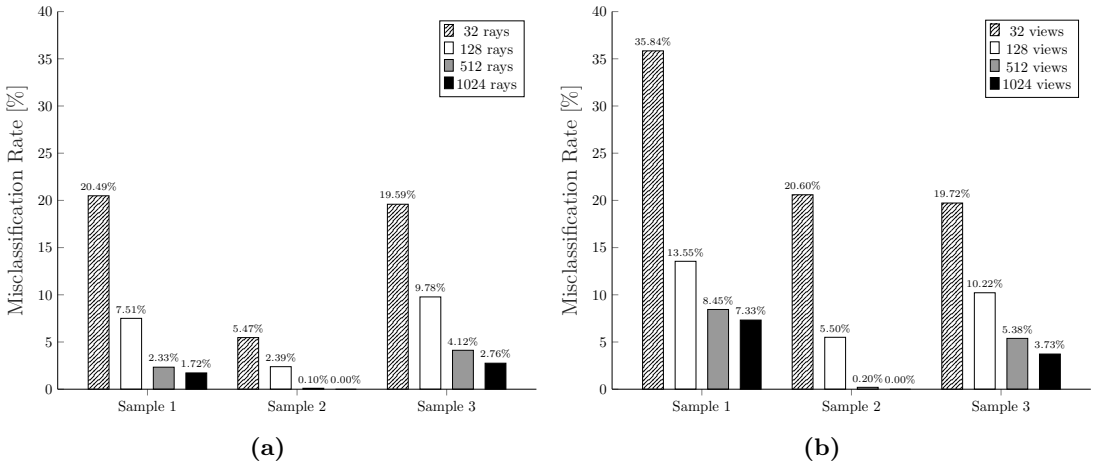


Figure 2.17: Misclassification rates comparison of the two algorithms using different settings and samples. (**a**) shows the results obtained with Plücker VSD, (**b**) shows the results obtained with MeshLab VSD.

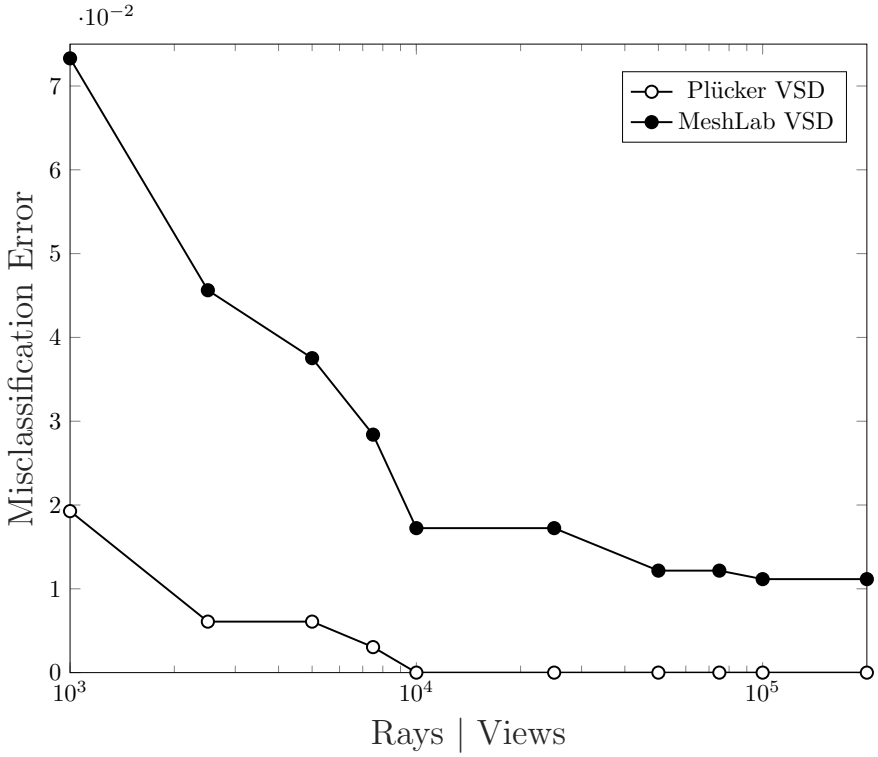


Figure 2.18: Convergence comparison of the misclassification error using the two algorithms on Sample 1.

Finally, we want to highlight another significant result. At the beginning of this section,

we stressed that one of the limitations of MeshLab algorithm regards the orientation of mesh triangles normals. Based on the notion of ambient occlusion, these normals must be oriented towards the outside of the mesh. This scenario is not so common in computer graphics: a triangular mesh may have normals flipped or it may not have normals at all. Indeed, as described in Section 2.3.1, depending upon the file format used to represent a mesh, there may or may not be encoded information on triangles normals. We then tried to flip all the normals of the third sample mesh and measuring the misclassification rate again as the number of views changed. Figure 2.19 shows how MeshLab algorithm is totally unable to solve the VSD problem in the case of incorrectly oriented normals. Our method, on the other hand, obtains the same results as before since it considers the entire sphere and not only the upper hemisphere according to the normal.

Table 2.1: Detail of times and misclassification rates of VSD performed on Sample 1 by the two algorithms.

Rays | Views	MeshLab VSD		Plücker VSD	
	Time [s]	M. E. Rate [%]	Time [s]	M. E. Rate [%]
1000	0.58	6.39	2.29	1.93
2500	1.27	4.56	5.70	0.61
5000	2.41	3.75	11.39	0.61
7500	3.67	2.84	16.99	0.30
10,000	4.77	1.72	22.66	0.00
25,000	11.74	1.72	44.31	0.00
50,000	24.23	1.22	114.20	0.00
75,000	34.79	1.22	170.50	0.00
100,000	48.51	1.12	226.92	0.00
200,000	91.08	1.12	456.20	0.00

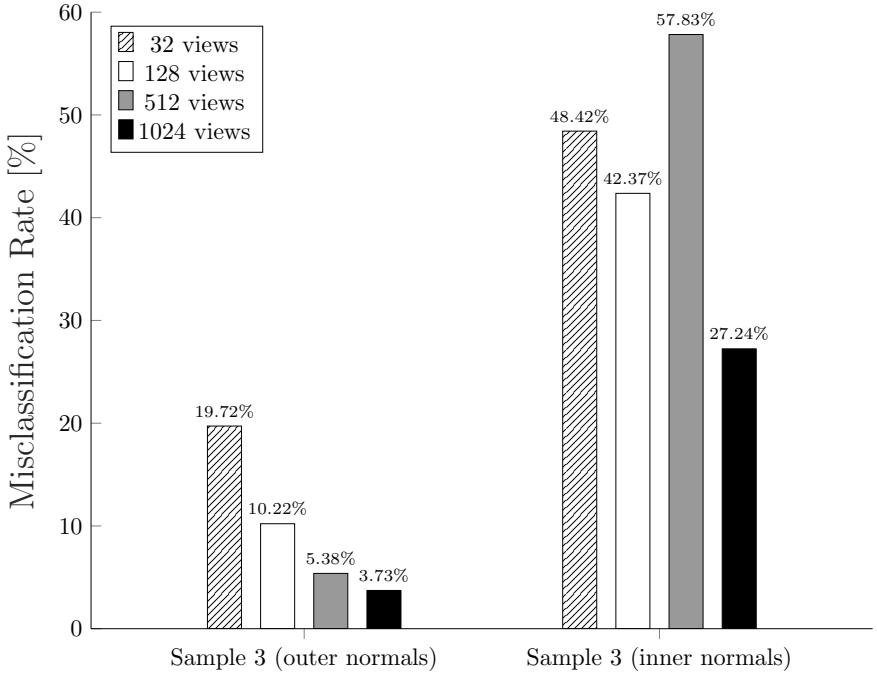


Figure 2.19: Misclassification rates comparison using MeshLab VSD algorithm on Sample 3 with outer and inner normals.

2.7 Hybrid approach

Results showed that our approach is more robust in terms of convergence to the maximum lossless compression. In addition, it is resilient to normals orientation, a fundamental characteristic for the industrial context.

However, from the results, it is evident that our algorithm exhibits high complexity and considerable computation time. To mitigate the computation time while maintaining classification optimality, we propose the following algorithm, referred to as the "Hybrid approach". This entails applying our Plücker algorithm only to triangles classified as non-visible by the MeshLab VSD algorithm. To carry on all of the test that follows, we used a midrange laptop equipped with an Intel i7-9750H CPU, 16 GB DDR4 RAM, and a NVIDIA GeForce GTX1650 4 GB GPU.

We start by comparing the time needed for the three algorithms to estimate the visibility for each face of the sample meshes. From Figure 2.20

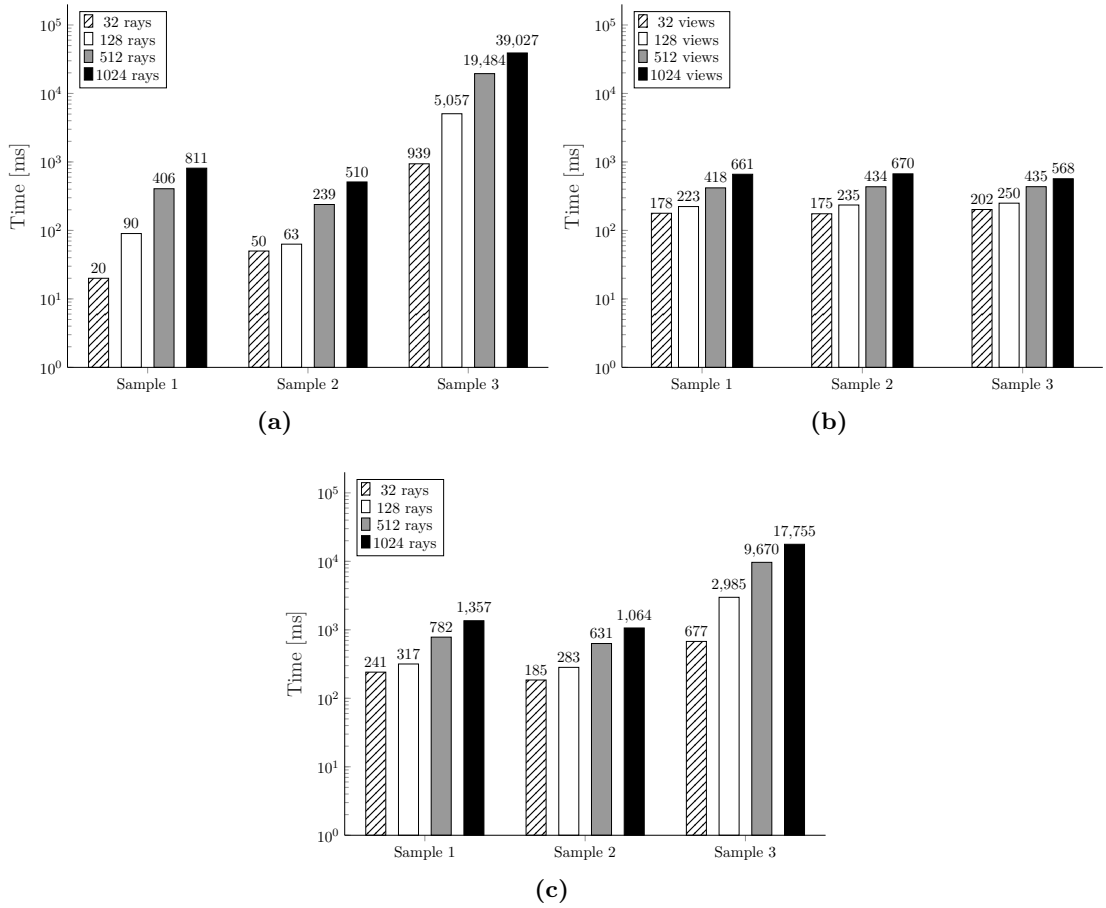


Figure 2.20: Computation time comparison of the two algorithms and the mixed approach using different settings and samples. (**a**) shows the results obtained with Plücker VSD, (**b**) shows the results obtained with MeshLab VSD, (**c**) shows the results obtained with the hybrid method.

Looking at Figure 2.20 it is clear that the computation time is higher compared to the MeshLab VSD algorithm and lower compared to our Plücker algorithm. As previously stated thePlücker method is the most computationally expensive being it $\mathcal{O}(N^2)$. Nonetheless, it can be observed in Figure 2.21 that the Hybrid approach it does achieve lower classification error compared to the Meshab VSD but is not precise as the the pure Plücker Method. This is the case since the Hybrid method method only re-checks the faces classified as invisible from the Meshlab VSD, therefore faces incorrectly classified as as visibile will not be corrected.

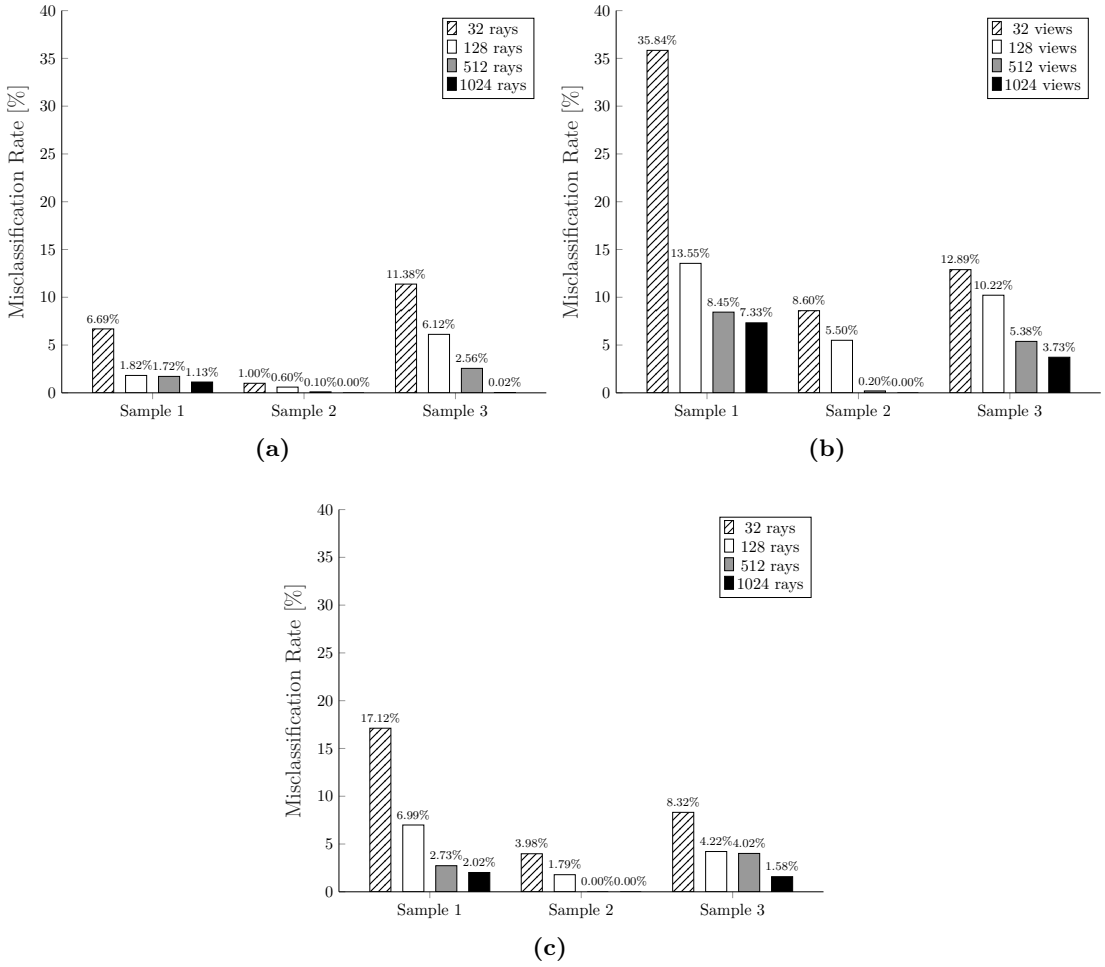


Figure 2.21: Misclassification rates comparison of the two algorithms using different settings and samples. (**a**) shows the results obtained with Plücker VSD, (**b**) shows the results obtained with MeshLab VSD, (**c**) shows the results obtained with the mixedhybrid method.

2.8 Conclusions

In this chapter, we address the visibility problem in object space. We started from the notion of ambient occlusion and we adapted it to solve the VSD problem. Thanks to the use of Plücker coordinates, we were able to speed up the ray-triangle intersection test. We compared the performance of our algorithm against a state-of-the-art one which exploits another approach based on image space. This allowed us to evaluate pros and cons of a solution in object space with another in image space. Results showed that our approach is more robust in terms of convergence to the maximum

lossless compression. In addition, it is resilient to normals orientation, a fundamental characteristic for the industrial context. Although the proposed solution is characterized by a high computational complexity, we stress that its impact is completely negligible since VSD techniques are typically used once per mesh. Anyway, there exist several acceleration techniques that can be adopted to speed up computations such as kd-trees, grids, bounding volume hierarchies [40]. Since the purpose of this work is to prove the result optimality, these improvements were not considered.

We then created an hybrid approach, where a first processing step is done using an image space VSD method and, then, a second step with our algorithm in order to give a more accurate result. In this case, our Plücker-based method will check only the invisible triangles recognized during the first step.

This hybrid approach proved to reduce the computational burden, but in some of our test cases, it also decreased the accuracy of the method since false visible faces will not be retested by our method.

The outcome of these experiments appears to be heavily influenced by the specific geometry involved. In some tests infact the hybrid method reached the optimal classification. Future developments may involve integrating our method with established space partitioning techniques like octree or kd-tree. Another possible route is refining the image space VSD by eliminating its quantization, leading to theoretical optimal classification.

3

Robust Simulation of Radial Optical Distortion

3.1 Introduction

Computer vision is playing a key role in the automation process due to the increasing use of recognition and pose estimation algorithms. Although, in recent years, the usage of three-dimensional scanners is increased, most recognition algorithms are still based on images [41]. Robots are required to autonomously operate based on information derived from those images. Due to the variability in this data, it is mandatory to test the safety, efficiency, and robustness of any new algorithm by means of realistic and reliable simulations. In fact, even the smallest artifact may cause the target not to be recognized correctly and, thus, the simulation of an ideal virtual camera is not sufficient for industrial purposes. Actually, many computer vision algorithms rely on the assumptions of a linear pinhole camera model, but the distortion caused by the optics is usually significant enough to violate this assumption. Obviously, it is not feasible to exactly replicate a real camera: noise, light conditions and lens physics deeply characterize the acquired image and modeling them is extremely complex. For this reason, in this work, we aim to provide an efficient and robust solution to simulate optical distortion for industrial cameras. In particular, we present a deep characterization of the common polynomial

distortion model and from there we develop a useful simulation tool that is embedded in Vostok[42]. Vostokis a free software developed by the company Euclidlabs, that also collabareted with the work here presented. Figure 3.1 shows an overview of the camera simulation toolkit implemented in Vostok which is based on this work.

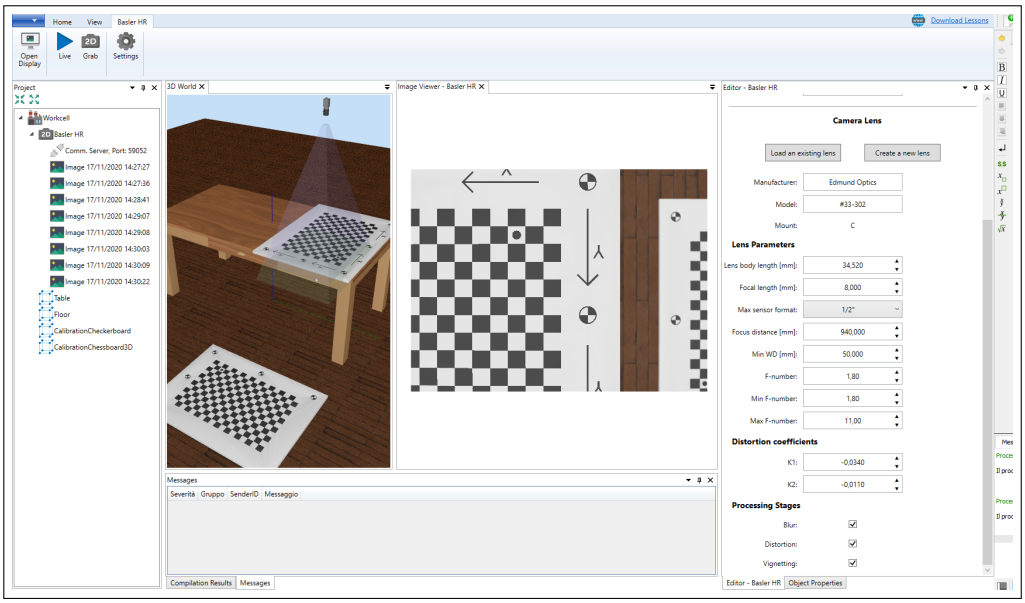


Figure 3.1: Overview of the camera simulation toolkit available in Vostok.

Decentering lens distortion was modeled by Conrady in 1919 [43], then improved by Brown in 1966 [44] and a definitive distortion model was proposed by Brown in 1971 [8]. The latter has been adopted by the computer vision community for several decades. Since then, many studies have focused on estimating the distortion coefficients of the polynomial models through several camera calibration techniques. Viala [45] compares different lens distortion models to define the one that achieves the best results under particular conditions. The author also introduces a novel technique to calibrate several models under stable conditions. Alvarez et al. [46] propose an algebraic approach for estimating optical distortion parameters based on the rectification of lines in the image. The authors present a new energy function, based on a statistical approach, to measure the distortion error and an efficient technique to minimize it. Bukhari [47] provides a novel method for automatic radial distortion estimation based on the plumb-line approach. That method only requires a single image and no special calibration pattern. However, although we have the equations to compensate the distortion, our interest is focused on applying it to ideal images to simulate the effects introduced by real lenses. Regarding the polynomial model, several solutions have been adopted to perform inverse radial

distortion: those can be divided in iterative and non-iterative ones. The former iteratively refine the distorted position until a convenient convergence is reached, while the latter try to find an approximated closed-form solution of the inverted model. For example, Drap et al [48] presents a formal calculus involving a power series used to deduce a closed-form solution. Iterative approaches give excellent results, but the processing time is drastically increased with respect to non-iterative methods. However, the use of the so-called look-up tables allow to store the mapping between ideal and distorted pixels: thus, the computation time becomes negligible. For this reason, we decided to focus on iterative methods because they offer a better accuracy. In the literature we can also find further simulation approaches that are based on a physical radiometric description of the lenses, i.e., how optical elements convert scene radiance in sensor irradiance [49, 50]. Those solutions are much more faithful but slower than those based on approximate polynomial models; the latter are then more suitable for real-time simulations.

In this chapter, to simulate optical distortion, we perform a comprehensive analysis of the inversion of the Brown's polynomial model: particular interest is placed on the determination of the limits of invertibility. Then, we describe an iterative algorithm to apply the distortion by exploiting the Newton's method. To evaluate the robustness of the proposed algorithm, we study its performance on synthetic images. In particular, Section 3.2 shows the problem formulation in detail. Section 3.3 presents design and software implementation of the simulation tool. In Section 3.4, the proposed algorithm is tested analyzing the robustness and quality of the results. Finally, concluding remarks and possible extensions are presented in Section 3.5.

3.2 Problem Formulation

Before introducing the distortion model, we need to recall some basics of an ideal camera model that can be used for simulation purposes. From a geometrical point of view, the most common choice is to adopt the so-called pinhole camera model [51]. In this section, we start by introducing this framework and, then, we move onto the distortion caused by the curved nature of any lens.

3.2.1 Pinhole Model

The pinhole camera model is widely adopted to describe how the coordinates in a three-dimensional space are projected into the image plane, i.e., the two-dimensional pixel space. This model, represented in Figure 3.2, assumes that the camera aperture has the dimension of an Euclidean point and no lenses are present. This is a very simplified model

and, therefore, some corrections must be included to develop an accurate simulator.

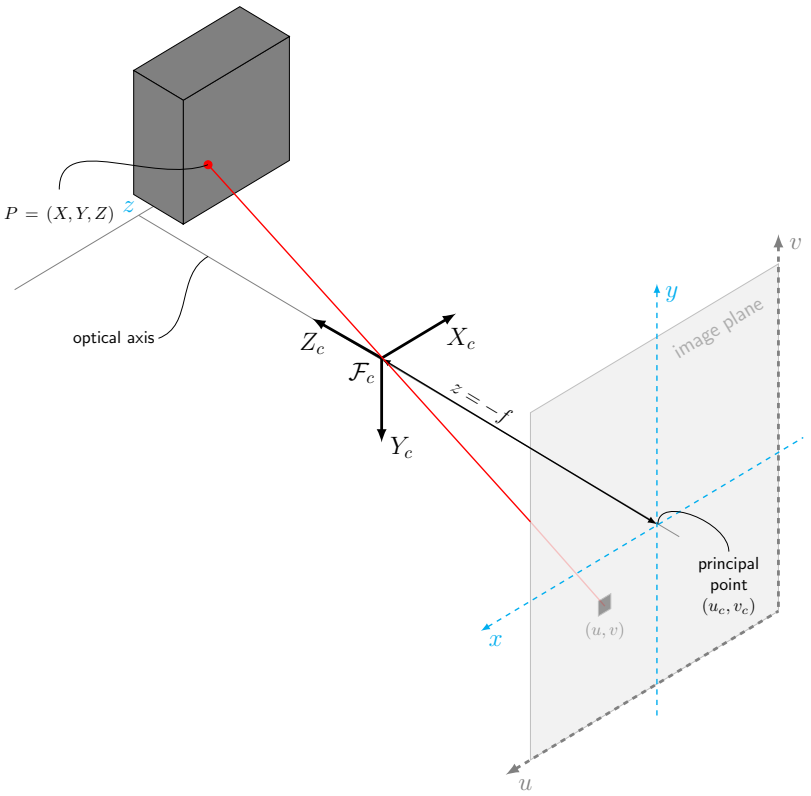


Figure 3.2: Representation of the pinhole camera model.

Any point $P = (X, Y, Z) \in \mathbb{R}^3$ in the camera's field of view can be mapped to a point (pixel) $(u, v) \in \mathbb{R}^2$ in the image plane by means of a transformation that takes into account camera intrinsic and extrinsic parameters. Actually, according to the pinhole camera model:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & \gamma & u_c & 0 \\ 0 & f_v & v_c & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The intrinsic parameters matrix K encodes information about the focal length f, the image sensor format and the principal point C. The latter represents the intersection between the optical axis and the image plane. In particular,

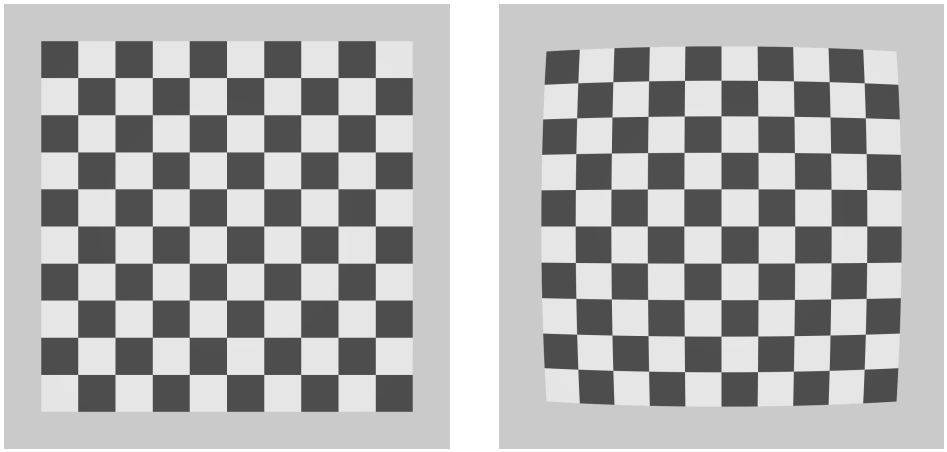
- $f_u = f/m_u$, where m_u is the pixel size width;
- $f_v = f/m_v$, where m_v is the pixel size height;
- γ is the skew coefficient: the pixels in a CCD sensor may not be perfectly squared, resulting in a small distortion along u and v directions;
- $C = (u_c, v_c)$ represents the principal point which is ideally the center of the image.

The model is also characterized by R and T, i.e., the rotation and translation matrices of the camera frame $(X_c, Y_c, Z_c, \mathcal{F}_c)$ with respect to the world frame.

Cameras are usually equipped with lenses because they allow us to gather more light and keep the image sharper. The pinhole camera model does not consider the presence of lenses, which makes it too simplified for our purposes.

3.2.2 Optical Distortion

The optical distortion occurs in an image when a deviation from the standard projection is present. This phenomenon can be modeled by two components: a radial and a tangential distortion. Radial distortion (Figure 3.3b,c) is caused by the spherical shape of the lens, whereas tangential distortion (Figure 3.3d) is caused by the decentering and non-orthogonality of the lens components with respect to the optical axis [52]. Radial distortions can be further classified as either barrel or pincushion distortions. The former happens when the lens' field of view is larger than the size of the image sensor. As a result, straight lines are visibly curved inwards, especially towards the extreme edges of the image. Pincushion distortion is the exact opposite: the field of view is smaller than the size of the image sensor and, consequently, straight lines appear to be curved outwards from the center.



(a) No effect

(b) Barrel distortion

(c) Pincushion distortion

(d) Tangential distortion

Figure 3.3: Overview of the main optical distortion effects by framing a checkerboard from the same position.

The traditional model of radial and tangential distortion, called Brown's model[8], gives the map between the position of the observed points $(\tilde{u}, \tilde{v})$ and the corresponding undistorted ones (u, v). The model, truncated at the fourth order, is the following:

$$\begin{aligned} u &= \tilde{u} + \overbrace{\Delta_u(k_1\tilde{r}^2 + k_2\tilde{r}^4)}^{\text{radial distortion}} + \overbrace{p_1(\tilde{r}^2 + 2\Delta_u^2) + 2p_2\Delta_u\Delta_v}^{\text{tangential distortion}} \\ v &= \tilde{v} + \Delta_v(k_1\tilde{r}^2 + k_2\tilde{r}^4) + 2p_1\Delta_u\Delta_v + p_2(\tilde{r}^2 + 2\Delta_v^2) \end{aligned}$$

where $\Delta_u = (\tilde{u} - u_c)$, $\Delta_v = (\tilde{v} - v_c)$, $\tilde{r} = \sqrt{\Delta_u^2 + \Delta_v^2}$, k_n is the n-th radial distortion coefficient, and p_n is the n-th tangential distortion coefficient. Although Brown's model

takes into account the tangential component, the radial one is the most significant in today's industrial cameras [53, 54]. For this reason, we decided to neglect the tangential component, obtaining the following simplified model:

$$u = \tilde{u} + \Delta_u(k_1\tilde{r}^2 + k_2\tilde{r}^4) \tag{3.1}$$
$$v = \tilde{v} + \Delta_v(k_1\tilde{r}^2 + k_2\tilde{r}^4)\,. \tag{3.2}$$

3.2.3 Inversion of Radial Distortion Model

In this section, we want to analyze the invertibility of the radial distortion model through a functional analysis. Our interest is primarily in studying the analytical bounds on the maximal residual distortion. Actually, we are interested in determining a priori the maximum radius r that we can distort, i.e., for which the model is invertible, given a particular pair of distortion coefficients (k_1, k_2). To do this, through a function study, we are going to identify the domain of existence of $\tilde{r}$ and, by using directly the Brown's model, we will be able to determine the maximum allowable radius r.

Let's start from the simplified model described by equations (3.1) and (3.2). To further simplify the notation, we can subtract both members of the equations for the corresponding principal point component, obtaining:

$$\begin{aligned}\overbrace{u - u_c}^{x} &= \overbrace{(\tilde{u} - u_c)}^{\tilde{x}}(1 + k_1\tilde{r}^2 + k_2\tilde{r}^4)\\ \underbrace{v - v_c}_{y} &= \underbrace{(\tilde{v} - v_c)}_{\tilde{y}}(1 + k_1\tilde{r}^2 + k_2\tilde{r}^4)\,.\end{aligned}$$

Therefore, a point $P = (x, y)$ can be described by means of a function of the corresponding distorted one $\tilde{P} = (\tilde{x}, \tilde{y})$ and the distance of $\tilde{P}$ from the principal point C. Formally, we obtain:

$$x = \tilde{x}(1 + k_1\tilde{r}^2 + k_2\tilde{r}^4) \tag{3.3}$$
$$y = \tilde{y}(1 + k_1\tilde{r}^2 + k_2\tilde{r}^4) \tag{3.4}$$

where $\tilde{r} = \|\tilde{P} - C\| = \sqrt{\tilde{x}^2 + \tilde{y}^2} \geq 0$. Those points can also be expressed in polar

coordinates, i.e.,

$$P = (x, y) = (r \sin\theta, r \cos\theta)$$
$$\tilde{P} = (\tilde{x}, \tilde{y}) = (\tilde{r} \sin\theta, \tilde{r} \cos\theta)\,.$$

By definition, radial distortion involves perturbation only on the radius of the pixel without modifying angles. Therefore, it is sufficient to inspect the radius only. By merging equations (3.3) and (3.4), we obtain:

$$\underbrace{x^2 + y^2}_{r^2} = \underbrace{(\tilde{x}^2 + \tilde{y}^2)}_{\tilde{r}^2}(1 + k_1\tilde{r}^2 + k_2\tilde{r}^4)^2$$

that leads to

$$\tilde{r} + k_1\tilde{r}^3 + k_2\tilde{r}^5 - r = 0\,. \tag{3.5}$$

Consequently, recalling that r is known, to obtain the position of the distorted point $\tilde{P}$, we need to find the positive real zero of equation (3.5) that is closer to r. Real and positive because $\tilde{r}$ represents a radius of a circumference, while closer to r because we want to discard unrealistic distortions.

Note that, by Abel-Ruffini theorem, there is no closed-form solution for the roots of a polynomial equation of degree five with arbitrary coefficients, but there exist several numerical techniques to estimate them. Starting from equation (3.5), we define (3.6) and (3.7), and resort to functional analysis to determine the domain in which the model is invertible.

$$f(\tilde{r}) := \tilde{r} + k_1\tilde{r}^3 + k_2\tilde{r}^5 \tag{3.6}$$
$$g(\tilde{r}) := f(\tilde{r}) - r \tag{3.7}$$

Note that $f(0) = 0$ and $r \in \mathbb{R}_0^+$. Therefore, $g(0) = -r \leq 0$. Then, we must identify the interval $\mathcal{D} = [0, R] \subset \mathbb{R}$, $R \in \mathbb{R}_0^+$ where the solution of the function $g(\tilde{r})$ exists and is unique. Since $g(\tilde{r})$ is continuous, this is equivalent to identify the interval in which $g(\tilde{r})$ is strictly monotonic. This analysis can be done directly on $f(\tilde{r})$ since r is constant. Within a neighborhood of zero, $f(\tilde{r})$ can be approximated to $f(\tilde{r}) \simeq \tilde{r} + O(\tilde{r})$, highlighting that it is clearly an increasing function in this area. From (3.6), we compute its derivative,

obtaining:

$$\dot{f}(\tilde{r}) = 1 + 3k_1\tilde{r}^2 + 5k_2\tilde{r}^4 \,.$$

We need to determine the interval where $\dot{f}(\tilde{r}) > 0$. Meanwhile, by exploiting the monotonicity, we prove the uniqueness of the solution. Let's compute the solutions of $\dot{f}(\tilde{r}) = 0$, i.e.,

$$(3k_1 + 5k_2\tilde{r}^2)\tilde{r}^2 = -1 \,. \tag{3.8}$$

The four roots of (3.8) are:

$$\tilde{r} = \pm\sqrt{\frac{\pm\sqrt{9k_1^2 - 20k_2} - 3k_1}{10k_2}} \,.$$

If $k_1 \geq 0$ and $k_2 \geq 0$ or if $k_1 < 0$ and $k_2 \geq \dfrac{9k_1^2}{20}$ then $\mathcal{D} \in \mathbb{R}_0^+$. In the other cases, the domain is $\mathcal{D} = [0, R]$, where

$$R = \sqrt{\frac{-\sqrt{9k_1^2 - 20k_2} - 3k_1}{10k_2}}$$

if $k_2 \neq 0$. Otherwise, if $k_2 = 0$, then

$$R = \sqrt{\frac{-1}{3k_1}} \,.$$

Table 3.1 summarizes all the intervals, depending on the distortion coefficients (k_1, k_2), in which the function $f(\tilde{r})$ is strictly monotonically increasing. Note also that R, when it exists finite, is also the maximum point of the function $f(\tilde{r})$ and, consequently, of $g(\tilde{r})$.

Table 3.1: Intervals of existence and uniqueness of the solution $\tilde{r}$.

	$k_1 < 0$	$k_1 \geq 0$
$k_2 < 0$	$0 \leq R < \sqrt{\frac{-\sqrt{9k_1^2-20k_2}-3k_1}{10k_2}}$	
$k_2 = 0$	$0 \leq R < \sqrt{\frac{-1}{3k_1}}$	$R \in \mathbb{R}_0^+$
$0 < k_2 < \frac{9}{20}k_1^2$	$0 \leq R < \sqrt{\frac{-\sqrt{9k_1^2-20k_2}-3k_1}{10k_2}}$	$R \in \mathbb{R}_0^+$
$k_2 \geq \frac{9}{20}k_1^2$	$R \in \mathbb{R}_0^+$	

3.3 Robust Radial Distortion Simulation

3.3.1 Newton-Raphson Method

As mentioned before, the Abel-Ruffini theorem states that there is no closed-form solution for the roots of a polynomial equation of degree five with arbitrary coefficients, but there exist several numerical techniques to estimate them. One of the most famous iterative methods is surely the Newton-Raphson method. The approximation process is repeated as

$$r_i = r_{i-1} - \frac{g(r_{i-1})}{\dot{g}(r_{i-1})}$$

until convergence is reached. To ensure the solution convergence, it is sufficient to verify that all the assumptions of the Newton-Raphson theorem are satisfied. In the case of the function $g(\tilde{r})$, defined in (3.7), all the hypotheses are respected. In particular, $g(r) \in \mathcal{C}^\infty$ which implies $g(r) \in \mathcal{C}^2$. Moreover, we defined the domain $\mathcal{D} = [0, R]$, such that $\forall\ p \in \mathcal{D}, \dot{g}(p) \neq 0$. Finally, the fact that the function $g(\tilde{r})$ is monotonically increasing and that $g(0) \leq 0$, implies that $\exists\ p \in [0, R] : g(p) = 0$.

Therefore, we can use the Newton-Raphson method to find the distorted radius $\tilde{r}$ starting from the original one r. Algorithm 2 summarizes the key steps on which the distortion algorithm is based.

3.3.2 Image Size Pre-Computation

For some combinations of the distortion coefficients (k_1, k_2), the distorted image may appear surrounded by black borders: note the effect in Figure 3.4b. This artifact occurs in the case of barrel-type distortions in which the image is curved inwards. In this case, we do not have enough information to complete the distorted image if the starting original image has the same size. To compensate this effect, we first compute the acquiring

Algorithm 2 Iterative distortion method

Input: undistorted pixel (p_x, p_y)

Implementation:

1. Convert (p_x, p_y) to normalized coordinates (x, y) using the inverse of the intrinsic parameters matrix K. Formally,

$$x = \frac{p_x - u_c}{f_u} \ , \quad y = \frac{p_y - v_c}{f_v} \, .$$

2. Find the numerical solution $\tilde{r}$ of the inverse distortion model by using the Newton-Raphson method. Initialize $r_0 = r = \sqrt{x^2 + y^2}$ and, until convergence, repeat:

$$\tilde{r}_i = \tilde{r}_{i-1} - \frac{(\tilde{r}_{i-1} + k_1 \tilde{r}_{i-1}^3 + k_2 \tilde{r}_{i-1}^5 - r)}{(1 + 3k_1 \tilde{r}_{i-1}^2 + 5k_2 \tilde{r}_{i-1}^4)} \, .$$

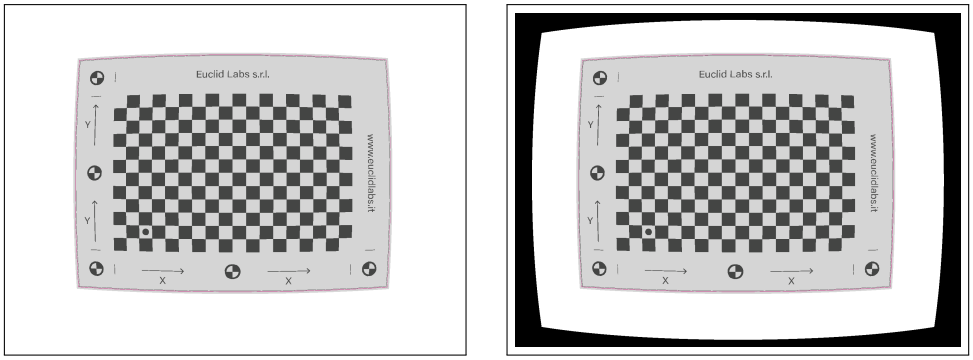
3. Given $\theta = \text{atan2}(\frac{y}{x})$

$$\tilde{x} = \tilde{r} \cos(\theta) \ , \quad \tilde{y} = \tilde{r} \sin(\theta) \, .$$

4. Convert $(\tilde{x}, \tilde{y})$ back to pixel coordinates using the intrinsic parameters matrix K, thus obtaining:

$$\tilde{p}_x = \tilde{x} f_u + u_c \ , \quad \tilde{p}_y = \tilde{y} f_v + v_c \, .$$

resolution, that will be larger than the original one in case of barrel-type distortions, and then we capture the image to be distorted using this information. Algorithm 3 shows how to compute the acquiring resolution. This procedure has the caveat that the final image, especially at the corners, will have a narrower field of view than a real camera lens with barrel-type distortion. One workaround to compensate for this effect is to increase the field of view of the camera sensor.



(a) **(b)**

Figure 3.4: A comparison of the same image obtained simulating a barrel-type distortion with (a) and without (b) image size pre-computation.

Algorithm 3 Image size pre-computation

Input: final distorted image size $(\tilde{w}, \tilde{h})$

Implementation:

1: $\tilde{x} \leftarrow \frac{\max(\tilde{w}-u_c, u_c)}{f_u}$
2: $\tilde{y} \leftarrow \frac{\max(\tilde{h}-v_c, v_c)}{f_v}$
3: $\tilde{r} \leftarrow \sqrt{\tilde{x}^2 + \tilde{y}^2}$
4: $\theta \leftarrow \text{atan2}(\frac{\tilde{y}}{\tilde{x}})$
5: $r \leftarrow$ `Apply Brown's model on` $\tilde{r}$
6: $w \leftarrow 2 f_u r \cos(\theta)$
7: $h \leftarrow 2 f_v r \sin(\theta)$

3.4 Numerical Results

Within this section we provide two numerical tests to evaluate the quality and robustness of the proposed algorithm. Simulation techniques are very difficult to evaluate and compare: they cannot be measured from the point of view of the computational complexity which depends proportionally on the number of pixels and can be easily accelerated with memory caching techniques. On the other hand, it is hard to compare the results with a real workbench, given the number of details characteristic of the real world that cannot be modeled in simulation, such as, for example, the effects of reflection and refraction of light on different materials. An interesting way to evaluate these methodologies is to test whether they are inherently robust. To prove this we developed two different tests. With the former, we analyzed the results of a state-of-the-art camera calibration algorithm from synthetic images generated using the tool we developed. If the calibration

provides estimated distortion parameters close to those used to generate the synthetic images, it means that our methodology is reliable. By means of the OpenCV calibration function, this test evaluates the ability of our algorithm to accurately reproduce radial distortions. The second experiment tests the robustness of our distortion procedure to be compliant with the un-distortion model by comparing an ideal image to a distorted and then rectified one. By feeding a distorted image into a state-of-the-art rectification algorithm, configured with the same distortion coefficients, we should get an image that is nearly the same as the original before applying the distortion. Both tests shall be carried on using different sets of distortion parameters to verify a wide range of conditions. To minimize the variability between individual tests, we defined a virtual environment within Vostok. We reproduced a particular sensor, the IDS GV-5250CP-C-HQ. Its resolution is 1600×1200 pixels whereas the pixel size is $4.5\,\mu m$ and the focal length is $8\,mm$. The camera is positioned $1500\,mm$ above a checkerboard of size $986 \times 732 \times 20.1$ mm. The latter is composed by 216 square blocks arranged in twelve rows and eighteen columns: each block is $40\,mm$ wide. Figure 3.5 shows a frame of this experimental virtual workbench.

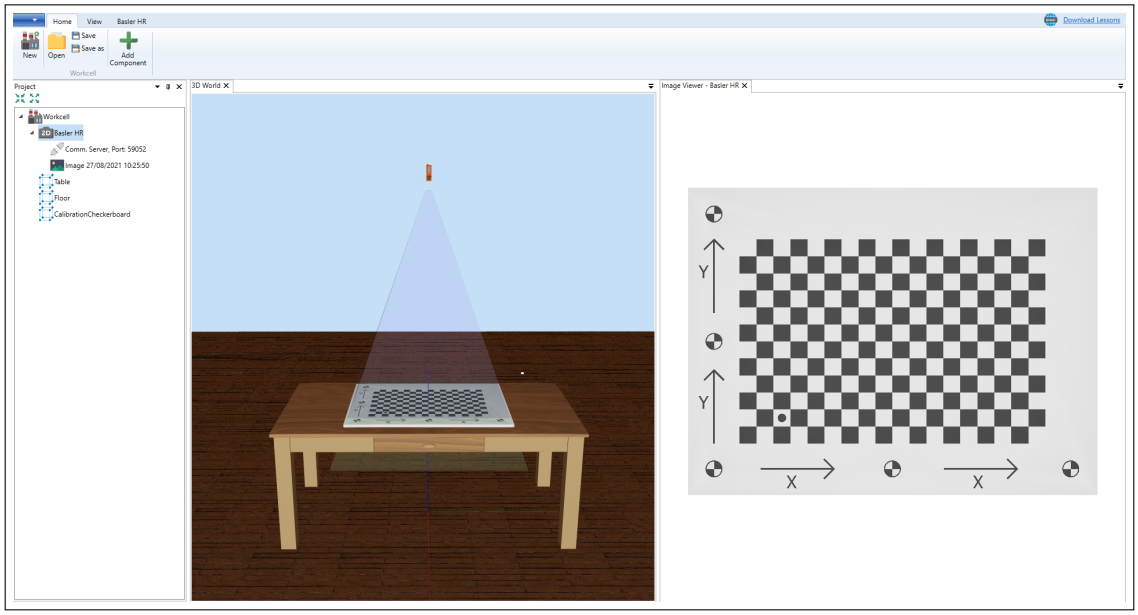


Figure 3.5: A frame of our test environment.

3.4.1 Calibration Test

This test aims to verify that the distortion reproduced by our algorithm is correct. The most effective way is to provide a calibration algorithm with a set of distorted synthetic

images generated by our method and, then, compare the estimated distortion parameters with those used to generate the input images. The widely adopted OpenCV library[55] provides one of the most optimized calibration algorithm, whose implementation is based on the well-known technique proposed by Zhang [53, 56]. The calibration function, `cv.calibrateCamera`, requires a series of photos taken with the camera to be calibrated. These images must frame a checkerboard from different poses: the best way is to maintain the distance between the camera and the checkerboard and keep the focus locked. The more poses are achieved, the better the result: we decided to use 20 poses, a value that allows us to obtain a RMS lower than 1 pixel. The latter describes the goodness of the estimate made by the algorithm: a value lower than one is an indication of a good estimate.

Table 3.2: Experimental results using several (k_1, k_2) pairs.

k_1	k_2	Test 1 - Calibration					Test 2 - Pearson Correlation		
		k_1^e	k_2^e	RMS	$k_1 - k_1^e$	$k_2 - k_2^e$	$\rho_{I_1^u, I_0}$	$\rho_{I_2^u, I_0}$	Diff.%
−1.25	1.25	−1.284	1.345	0.533 84	0.034	−0.095	97.61%	97.37%	0.23%
−1.00	1.00	−1.022	1.037	0.536 09	0.022	−0.037	97.79%	97.41%	0.38%
−0.75	0.75	−0.771	0.766	0.523 98	0.021	−0.016	97.81%	97.38%	0.43%
−0.75	1.00	−0.778	1.105	0.537 57	0.028	−0.105	99.19%	99.00%	0.19%
−0.50	0.25	−0.529	0.353	0.534 72	0.029	−0.103	97.83%	97.39%	0.44%
−0.50	0.50	−0.534	0.683	0.533 70	0.034	−0.183	99.38%	98.95%	0.43%
−0.50	0.75	−0.530	0.874	0.566 21	0.030	−0.124	97.84%	97.52%	0.32%
−0.40	0.00	−0.420	0.068	0.528 40	0.020	−0.068	97.88%	97.68%	0.20%
−0.30	−0.20	−0.325	−0.086	0.527 67	0.025	−0.114	99.46%	99.26%	0.20%
−0.25	−0.25	−0.280	−0.081	0.528 49	0.030	−0.169	98.66%	98.37%	0.29%
−0.25	0.25	−0.279	0.408	0.512 77	0.029	−0.158	97.83%	97.59%	0.24%
0.00	0.00	−0.006	−0.008	0.353 05	0.006	0.008	100.00%	99.91%	0.09%
0.25	−0.25	0.276	−0.499	0.527 05	−0.026	0.249	97.71%	97.66%	0.05%
0.25	0.25	0.273	0.041	0.529 68	−0.023	0.209	98.78%	98.80%	−0.02%
0.50	−0.75	0.523	−0.938	0.536 08	−0.023	0.188	97.62%	97.59%	0.04%
0.50	−0.50	0.526	−0.736	0.533 65	−0.026	0.236	98.76%	98.78%	−0.01%
0.50	−0.25	0.525	−0.468	0.538 47	−0.025	0.218	98.76%	98.78%	−0.03%
0.50	0.50	0.519	0.383	0.540 15	−0.019	0.117	96.35%	96.32%	0.03%
0.50	1.00	0.530	0.789	0.536 33	−0.030	0.211	95.99%	96.06%	−0.07%
0.75	−1.00	0.769	−1.144	0.554 48	−0.019	0.144	97.52%	97.53%	−0.01%
0.75	−0.75	0.759	−0.807	0.552 40	−0.009	0.057	96.38%	96.34%	0.04%
0.75	−0.25	0.767	−0.380	0.556 52	−0.017	0.130	95.96%	96.03%	−0.08%
0.75	0.50	0.755	0.516	0.555 28	−0.005	−0.016	96.32%	96.30%	0.02%
0.75	0.75	0.767	0.659	0.560 48	−0.017	0.091	97.42%	97.54%	−0.12%
1.00	−1.00	1.015	−1.126	0.564 23	−0.015	0.126	95.77%	95.90%	−0.13%
1.00	0.75	1.013	0.723	0.568 00	−0.013	0.027	96.17%	96.27%	−0.10%

To perform the experiment, we generated a set of 26 (k_1, k_2) pairs and, for each pair, we took 20 photos of the checkerboard each time with a different pose using Vostok, our simulation tool. Those images are then processed using the calibration algorithm provided

by OpenCV, obtaining the estimation of the distortion parameters (k_1^e, k_2^e). Table 3.2 summarizes the results of this test (*Test 1*). The table should provide a sufficient range of paramteres to cover the majority of industrial applications. Looking at the results it is clear that the method proposed works very well for (k_1, k_2) pairs with high amplitude, i.e., those generating marked distortions. As far as minor distortions are concerned, the deviation between the values used and their estimate becomes important only for k_2, always in excess of magnitude. Our hypothesis is that our parameters and those estimated by OpenCV represent a very similar distortion, closer than the numbers tell as can be seen in the last column of Table 3.2. Anyway, this issue will be analyzed in the next section, where the second experiment will be explained.

3.4.2 Pearson Correlation Test

The previous test showed that the distortion produced by our method is usually correctly recognized by the OpenCV calibration algorithm. However, in the case of low-magnitude coefficient pairs, the difference in values cannot be ignored or justified by an error in the calibration algorithm. Our hypothesis is that some pairs and their estimates reproduce very similar, if not equal, effects, even if they are quite different in value. For this reason, we decided to develop two different tests: in the first we verify that, by taking a distorted image generated by our tool and, then, applying a correction filter with the same parameters, the obtained image coincides with the original one. In the second test, instead, we applied the estimated parameters on the correction filter to the distorted image and compared the output with the initial image. To perform these tests, once again, we exploited an OpenCV function, `cv.undistort`, that allows to rectify a distorted image knowing the camera parameters. To be more precise, the tests can be reproduced by following these steps:

1. Define a camera pose and grab a photo I_0;
2. Define a pair of (k_1, k_2) and grab a photo I_d using our distortion method;
3. Calibrate the camera using the OpenCV calibration algorithm and retrieve (k_1^e, k_2^e);
4. Apply the OpenCV rectification filter on I_d using (k_1, k_2) to obtain I_1^u;
5. Apply the OpenCV rectification filter on I_d using (k_1^e, k_2^e) to obtain I_2^u;
6. Compute Pearson correlation coefficient of I_1^u and I_2^u with regard to I_0, i.e., $\rho_{I_1^u, I_0}$ and $\rho_{I_2^u, I_0}$.

This test tries to verify if

$$\mathrm{U}_{\mathrm{k}_1,\mathrm{k}_2}(\mathrm{D}_{\mathrm{k}_1,\mathrm{k}_2}(I_0)) = I_0 \tag{3.9}$$

where $\mathrm{D}_{\mathrm{k}_1,\mathrm{k}_2}(\cdot)$ represents our distortion algorithm whereas $\mathrm{U}_{\mathrm{k}_1,\mathrm{k}_2}(\cdot)$ the OpenCV rectification filter. If (3.9) is valid, it means that $\mathrm{D}_{\mathrm{k}_1,\mathrm{k}_2}(\cdot) = \mathrm{U}^{-1}_{\mathrm{k}_1,\mathrm{k}_2}(\cdot)$, i.e., our algorithm provides a perfect distortion following the Brown model. We chose the Pearson Correlation Coefficient (PCC) to compare images since it provides a synthetic parameter of similarity of two bivariate distributions: a value equal to 1 indicates that the two variables are linearly correlated whereas a PCC equal to zero means that the twos are totally uncorrelated. The formula is

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\mathrm{var}\,(X)\,\mathrm{var}\,(Y)}$$

where X and Y represent the two images in gray scale.

The obtained results are reported in Table 3.2: the first column reports the Pearson coefficients comparing I_1^u with I_0 for each pair whereas, the second one, the ones comparing I_2^u with I_0. The PCC using (k_1, k_2) in the rectification filter are very close to one and this proves that our algorithm produces a reliable result, especially for a barrel distortion in which PCC higher than 99% are achieved. The PCC using (k_1^e, k_2^e), instead, suggest us a couple of considerations. First of all, it can be noticed that, for barrel distortion, the OpenCV calibration algorithm is not so precise in estimating the parameters since, by applying them to the filter, we get a worse result than using the reference (k_1, k_2) pair. However, this is not repeated in the case of pincushion distortion, where, although the values of k_2 and k_2^e differ appreciably, the PCC of the two is very close if not the same. This leads us to the second consideration, i.e., very different pair can represent very similar if not identical effects.

3.5 Conclusions

In this chapter, we introduced a robust simulation technique for radial optical distortion. The reliability of the proposed algorithm makes this tool very useful for testing robotics and machine vision applications. Through the use of GPU architectures, the generation of realistic images occurs in negligible time. This allows to use the simulator to generate synthetic image datasets to train deep learning models which are increasingly being used in industry. At the same time, it can be a great tool to verify the match between a particular sensor and an optic, and even to perform preliminary analysis on new designs.

After a brief summary of the pinhole model, in Section 2.3 we discussed the distortion model proposed by Brown. From that, we analyzed the distortion function in order to determine the analytical bounds of invertibility. Section 2.5, showed an iterative method for finding solutions to the inverse problem and solving the black borders issue. Finally, in Section 2.6, we presented two numerical tests to evaluate the quality and robustness of the proposed algorithm. Both tests demonstrated excellent ability to simulate distortion and numerical robustness of the iterative algorithm.

Obviously, the proposed simulation model can be extended. A future improvement might be to consider also the tangential distortion component even though it contributes minimally to the final image formation in today's cameras. Instead, to improve the simulation tool offered, we are going to add the simulation of the focal aberration effect that results to be the other greater factor of distortion in the final image.

4

Conclusions

In this dissertation, we have talked about the increasing role that digital twins play in the modern manufacturing industry and proposed two methods to push further the boundaries of such technology. This work has been the result of a collaborative effort with the company Euclidlabs, a company that works in the machine vision field and provides bin picking solutions. In collaboration with their team we developed and tested solutions to improve the effectiveness of their software.

Specifically, in chapter 2, we introduced a novel approach to solve the visible surface determination problem using Plücker coordinates. This study presents a robust technique for addressing the global visibility problem in object space, ensuring theoretical convergence to the optimal solution. Such novel method allows us to determine, within a finite number of steps, whether a given face of the mesh is globally visible or not, and with what degree of visibility. As a result, we were able to optimize 3D models and enhance the performance of various simulator tasks, including rendering, vision system simulation, collision detection, pose estimation algorithms tuning and setup.

Then, in Chapter 3, we improved the simulation of 2D cameras by developing an efficient and robust approach for radial distortion simulation. Such approach has been derived by first conducting an analytical examination of the optical distortion model, emphasizing its constraints regarding invertibility. Such approach has been then validated through numerical tests, confirming the algorithm's robustness.

In conclusion, it is our hope that through our efforts, our work has contributed to the advancement of the digital twin technology, leading to more realistic simulations, optimization of vision algorithms and ultimately in reducing production times and costs for goods and services.

Bibliography

[1] J. Pallant, S. Sands, and I. Karpen, "Product customization: A profile of consumer demand," *Journal of Retailing and Consumer Services*, vol. 54, p. 102030, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0969698919311609

[2] X. Ji and S. Abdoli, "Challenges and opportunities in product life cycle management in the context of industry 4.0," *Procedia CIRP*, vol. 119, pp. 29–34, 2023, the 33rd CIRP Design Conference. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827123004225

[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *ICRA 2009*, vol. 3, 01 2009.

[4] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[5] Alessandro Rossi and Marco Barbiero and Ruggero Carli, "Vostok: 3D scanner simulation for industrial robot environments," *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, vol. 19, no. 3, pp. 71–86, 2020. [Online]. Available: https://elcvia.cvc.uab.es/article/view/v19-n3-rossi-barbiero-carli

[6] Mars, the digital twin of your project. [Online]. Available: https://www.euclidlabs.it/mars-pick-inspect-place-software/

[7] A. Rossi, M. Barbiero, P. Scremin, and R. Carli, "Robust visibility surface determination in object space via plücker coordinates," *Journal of Imaging*, vol. 7, no. 6, 2021. [Online]. Available: https://www.mdpi.com/2313-433X/7/6/96

[8] D. C. Brown, "Close-range camera calibration," *PHOTOGRAMMETRIC ENGINEERING*, vol. 37, no. 8, pp. 855–866, 1971.

[9] S. Chen, Y. Li, and N. M. Kwok, "Active vision in robotic systems: A survey of recent developments," *The International Journal of Robotics Research*, vol. 30, no. 11, pp. 1343–1377, 2011.

[10] J. Posada, C. Toro, I. Barandiaran, D. Oyarzun, D. Stricker, R. de Amicis, E. B. Pinto, P. Eisert, J. Döllner, and I. Vallarino, "Visual computing as a key enabling technology for industrie 4.0 and industrial internet," *IEEE Computer Graphics and Applications*, vol. 35, no. 2, pp. 26–40, 2015.

[11] N. Okino, Y. Kakazu, and M. Morimoto, "Extended depth-buffer algorithms for hidden-surface visualization," *IEEE Computer Graphics and Applications*, vol. 4, no. 5, pp. 79–88, 1984.

[12] M. Uysal, B. Sen, and C. Celik, "Hidden surface removal using bsp tree with cuda," *Global Journal on Technology*, vol. 3, 2013.

[13] J. E. Warnock, "A hidden surface algorithm for computer generated halftone pictures," Ph.D. dissertation, Utah University Salt Lake City Department of Computer Science, 1969.

[14] I. E. Sutherland, R. F. Sproull, Robert, and A. Schumacker, "A characterization of ten hiddensurface algorithms," *ACM Computing Surveys (CSUR)*, 1974.

[15] G. Vaněkčkek Jr., "Back-face culling applied to collision detection of polyhedra," *The Journal of Visualization and Computer Animation*, vol. 5, no. 1, pp. 55–63, 1994. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/vis.4340050105

[16] F. Durand, G. Drettakis, and C. Puech, "The 3d visibility complex : a new approach to the problems of accurate visibility," in *Rendering Techniques '96*, X. Pueyo and P. Schröder, Eds. Vienna: Springer Vienna, 1996, pp. 245–256.

[17] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," in *Eurographics Italian Chapter Conference*, V. Scarano, R. D. Chiara, and U. Erra, Eds. The Eurographics Association, 2008.

[18] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *J. Graph. Tools*, vol. 2, no. 1, p. 21–28, Oct. 1997.

[19] S. Nirenstein, E. Blake, and J. Gain, "Exact From-Region Visibility Culling," in *Eurographics Workshop on Rendering*, P. Debevec and S. Gibson, Eds. The Eurographics Association, 2002.

[20] F. Durand, "3d visibility: analytical study and applications," *PhD dissertion, universitè Joseph Fourier, France*, 1999.

[21] J. Bittner and P. Wonka, "Visibility in computer graphics," *Environment and Planning B: Planning and Design*, vol. 30, no. 5, pp. 729–755, 2003.

[22] J. J. Jiménez, C. J. Ogáyar, J. M. Noguera, and F. Paulano, "Performance analysis for gpu-based ray-triangle algorithms," in *2014 International Conference on Computer Graphics Theory and Applications (GRAPP)*, 2014, pp. 1–8.

[23] J. Havel and A. Herout, "Yet faster ray-triangle intersection (using sse4)," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 3, pp. 434–438, 2010.

[24] N. Altin and E. Yazgan, "Rcs prediction using fast ray tracing in plücker coordinates," in *2013 7th European Conference on Antennas and Propagation (EuCAP)*, 2013, pp. 284–288.

[25] H.-P. S. Leif Kobbelt, Jens Vorsatz, "Multiresolution hierarchies on unstructured triangle meshes," *Computational Geometry Journal: Theory and Applications*, vol. 14, pp. 5–24, 1999.

[26] H. Plantinga and C. R. Dyer, "Visibility, occlusion, and the aspect graph," *International Journal of Computer Vision*, vol. 5, no. 2, pp. 137–160, 1990.

[27] S. Zhukov, A. Iones, and G. Kronin, "An ambient light illumination model," in *Rendering Techniques '98*, G. Drettakis and N. Max, Eds. Vienna: Springer Vienna, 1998, pp. 45–55.

[28] S. M. Méndez-Feliu Àlex, "From obscurances to ambient occlusion: A survey," *The Visual Computer*, vol. 25, pp. 181–196, 2009.

[29] M. Pellegrini, "Ray shooting and lines in space," *Chapter 41 of:Handbook of discrete and computational geometry*, 2004.

[30] J. Arvo, "Stratified sampling of 2-manifolds," *SIGGRAPH 2001 Course Notes*, vol. 29, no. 2, 2001.

[31] R. Bridson, "Fast poisson disk sampling in arbitrary dimensions." in *SIGGRAPH sketches*, 2007, p. 22.

[32] M. E. Muller, "A note on a method for generating points uniformly on n-dimensional spheres," *Commun. ACM*, vol. 2, no. 4, p. 19–20, Apr. 1959.

[33] Á. González, "Measurement of areas on a sphere using fibonacci and latitude–longitude lattices," *Mathematical Geosciences*, vol. 42, no. 1, p. 49, 2010.

[34] B. Keinert, M. Innmann, M. Sänger, and M. Stamminger, "Spherical Fibonacci Mapping," *ACM Transactions on Graphics*, vol. 34, no. 6, pp. 1–7, nov 2015.

[35] J. Březina and P. Exner, "Fast algorithms for intersection of non-matching grids using plücker coordinates," *Computers & Mathematics with Applications*, 02 2017.

[36] R. Jones, "Intersecting a Ray and a Triangle with Plücker Coordinates," *RTNews*, 2000.

[37] K. Shoemake, "Plücker Coordinate Tutorial," *RTNews*, 1998.

[38] L. Bavoil and M. Sainz, "Screen space ambient occlusion," *NVIDIA developer information: http://developers. nvidia. com*, vol. 6, 2008.

[39] MeshLab, "MeshLab GitHub Repository," https://github.com/cnr-isti-vclab/meshlab, 2021.

[40] J. Bittner, V. Havran, and P. Slav´ik, "Hierarchical visibility culling with occlusion trees," *Proceedings. Computer Graphics International (Cat. No.98EX149)*, pp. 207–219, 1998.

[41] Y. Amit, *2D object detection and recognition: Models, algorithms, and networks*. MIT Press, 2002.

[42] Vostok 3D Studio. [Online]. Available: https://www.vostok3dstudio.com/

[43] A. E. Conrady, "Decentred Lens-Systems," *Monthly Notices of the Royal Astronomical Society*, vol. 79, no. 5, pp. 384–390, 03 1919. [Online]. Available: https://doi.org/10.1093/mnras/79.5.384

[44] D. C. Brown, "Decentering distortion of lenses," *PHOTOGRAMMETRIC ENGINEERING*, vol. 32, pp. 444–462, 1966.

[45] C. Ricolfe-Viala and A.-J. Sánchez-Salmerón, "Lens distortion models evaluation," *Applied optics*, vol. 49, pp. 5914–28, 10 2010.

[46] L. Alvarez, L. Gomez, and J. R. Sendra, "Algebraic Lens Distortion Model Estimation," *Image Processing On Line*, vol. 1, pp. 1–10, 2010.

[47] F. Bukhari and M. N. Dailey, "Automatic radial distortion estimation from a single image," *Journal of mathematical imaging and vision*, vol. 45, no. 1, pp. 31–45, 2013.

[48] P. Drap and J. Lefèvre, "An exact formula for calculating inverse radial lens distortions," *Sensors*, vol. 16, no. 6, p. 807, 2016.

[49] C. Kolb, D. Mitchell, and P. Hanrahan, "A realistic camera model for computer graphics," in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '95. New York, NY, USA: Association for Computing Machinery, 1995, p. 317–324. [Online]. Available: https://doi.org/10.1145/218380.218463

[50] P. Y. Maeda, P. B. Catrysse, and B. A. Wandell, "Integrating lens design with digital camera simulation," in *Digital Photography*, N. Sampat, J. M. DiCarlo, and R. J. Motta, Eds., vol. 5678, International Society for Optics and Photonics. SPIE, 2005, pp. 48 – 58. [Online]. Available: https://doi.org/10.1117/12.588153

[51] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, ser. Always learning. Pearson, 2012. [Online]. Available: https://books.google.it/books?id=gM63QQAACAAJ

[52] R. Swaminathan and S. K. Nayar, "Non-Metric Calibration of Wide-Angle Lenses and Polycameras," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1172–1178, 2000.

[53] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[54] A. Wang, T. Qiu, and L. Shao, "A simple method of radial distortion correction with centre of distortion estimation," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 3, pp. 165–172, 2009.

[55] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[56] W. Burger, "Zhang's Camera Calibration Algorithm: In-Depth Tutorialand Implementation," University of Applied Sciences Upper Austria, School ofInformatics, Communications and Media, Dept. of DigitalMedia, Hagenberg, Austria, Tech. Rep. HGB16-05, May 2016. [Online]. Available: http://staff.fh-hagenberg.at/burger/