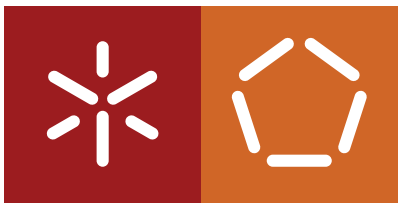
Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo José Mendes Marques

Live Projection Mapping

October 2015



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo José Mendes Marques

Live Projection Mapping

Master dissertation
Master Degree in Informatics Engineering

Dissertation supervised by
António Ramires Fernandes
Nelson Alves

October 2015

ACKNOWLEDGEMENTS

I would never have been able to finish my dissertation without the guidance of my supervisors, help from friends, and support from my family.

I would like to express my gratitude to my supervisor António Ramires, co-supervisor Nelson Alves and Professor Pedro Moreira for their feedback and guidance through all obstacles during this thesis.

I would like to thanks to CCG for the availability to use their space and material during this year to develop the application and test the results. I am also grateful to all members of CCG for all the tips for the project, for the help to mount the setup used and for providing a good environment while working.

I would also like to thank all my family, specially my parents and brother, for supporting and encouraging me to always search for improvements and innovations either in this dissertation or in everything on my life.

I would like to thanks all my colleagues and friends for the help during this journey, for motivating me, but also for not allowing me to work and distracting me when I needed some rest.

Finally, I also place on record my gratitude to one and all, who directly or indirectly, have lent their hand in this venture.

ABSTRACT

Traditional Projection Mapping systems use a static scene that is reconstructed in an off-line step. This kind of setups not only can't handle runtime interaction with the real scene but also can't be reused in other scenarios, as they need a recreation of the respective scene to be used in the virtual world, usually, hand measured and modeled.

The main goal of this work is to surpass this problem exploring a projection mapping system that reconstructs the surface in run-time, adapting the projection as the scene changes. To achieve this goal the system needs to handle the interaction between two distinct areas that have seen substantial progress within the past few years: Surface Reconstruction and Spatial Augmented Reality. With the recent advances in real-time Surface Reconstruction, this combination allows the development of a real time projection mapping system, in which the real world's scene can be interactively modified and reconstructed in the virtual world. To recreate the scene's surface model, a depth sensor is used, providing depth information alongside an RGB image. Subsequent to the scene's reconstruction, the projection can have two purposes, one is to simply manipulate the surface appearance and the other is to add virtual objects to the scene. The last scenario is where Spatial Augmented Reality and its View Dependent Rendering concept are introduced.

Keywords: Projection Mapping, Surface Reconstruction, Spatial Augmented Reality, Mixed Reality

RESUMO

Os sistemas tradicionais de mapeamento de projeções utilizam uma cena estática que é reconstruída offline. Este tipo de setups não só não podem lidar com interação com a cena real em tempo de execução, como também não podem ser reutilizados em outros cenários, visto precisarem de uma recriação da respectiva cena a ser utilizada no mundo virtual, geralmente, medida e modelada à mão. O objetivo principal deste trabalho é superar esse problema explorando um sistema de mapeamento de projeção que reconstrói a superfície em tempo de execução, adaptando a projecção assim que o cenário é modificado. Para atingir esse objetivo, o sistema precisa de lidar com a interação entre duas áreas distintas que têm visto progressos substanciais nos últimos anos, que são a Reconstrução de Superfícies e a da Realidade Aumentada Espacial. Com os recentes avanços em Recontrução de Superfícies em tempo real, esta combinação permite o desenvolvimento de um sistema de mapeamento de projeções em tempo real, em que a cena de mundo real pode ser modificada interactivamente e reconstruída no mundo virtual. Para recriar o modelo da superfície da cena, um sensor de profundidade é utilizado, fornecendo informações sobre a profundidade da imagem além da RGB. Após a reconstrução da cena, a projecção pode ter dois propositos, um é o de simplesmente manipular a aparência da superfície, o outro, é o de adicionar objectos virtuais na cena. No último caso é onde a Realidade Aumentada Espacial e o seu conceito de View Dependent Rendering são introduzidos.

Keywords: Projection Mapping, Surface Reconstruction, Spatial Augmented Reality, Mixed Reality

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 CONTEXT

The recent expansion of affordable real-time RGB-D sensors (e.g. the Microsoft Kinect and Asus Xtion Pro Live) has opened the way for the development of new computer vision applications that take advantage of the depth information provided by these sensors, such as gesture recognition and robotics control applications (Canessa et al., 2014; Cruz et al., 2012). In particular, 3D Surface Reconstruction (SR) has evolved significantly, allowing for real-time reconstruction which promises an interesting step forward in Augmented Reality (AR) (Izadi et al., 2011). These devices have already been used in the field of AR or Mixed Reality, to localize and track specific objects, markers and user movements, controlling the interaction between the user and the virtual objects. This area has been growing in the last years.

In contrast to traditional Virtual Reality, in AR the real environment is not completely suppressed, instead artificial information is mixed with the real scene. To accomplish this goal, the majority of the AR approaches involve rendering graphics over a live video feed on hand held or head worn devices, which have many limitations such as a restricted field of view, image quality, ergonomy, and discomfort due to simulator sickness (especially during fast head movements) (Bimber and Raskar, 2005).

Novel approaches have taken AR beyond traditional displays methods, allowing its use in museums, edutainment, research, industry and other environments where a per user device was impractical. These methods take advantage of large optical elements and video-projectors, using the real environment as a display. With the users head position information, Spatial Augmented Reality (SAR), or Projection Mapping (PM), can change scene surfaces appearance, projecting light over the physical environment so that virtual 3D objects appears superimposed over the real world.

Although multiple users can observe the projections, there are situations where only a single user will have the correct perspective, as it uses view dependent rendering (Benko et al., 2014). In Jones et al. (2014), they try to solve the multi-user problem by averaging user's head positions, although, this only works in scenes where the virtual content is near the physical surfaces, and theirs heads are relatively close to each other. Their applications illustrate the capabilities of a SAR system in a fixed scenario, but there are already applications dealing with dynamic scenes, such as in Piumsomboon et al. (2011), which presents a more similar AR system to the objective of this project, where users can create the scene using real objects, and the virtual objects interact with the reconstructed scene simulating physical collisions.

There are multiple examples of this concept using sandboxes to be able to easily mold the content of the scene, such as creating rivers and volcanoes interactively for example, where the virtual fluids will interact according to the laws of physics with the reconstructed scene (Kreylos). Another example of this concept was presented in Benko et al. (2014), in which the combination of SAR and SR enables the interaction between two users with a shared virtual scene.

1.2 MOTIVATION

The recent hardware evolution allowed the spread of Augmented Reality Systems and over the last years they are being increasingly used in multiple fields. Projection Mapping, or Video Mapping as is commonly named, is a specific type of AR, and is also increasing its popularity. More and more, public events start to use projection mapping to entertain the spectators, from projecting in buildings and cars, to backgrounds and objects in theatrical like events to change the perception of the real world.

The effects achieved with Spatial Augmented Reality or Projection Mapping do not suit only large scale entertainment events, but can also be applied in smaller scale applications such as terrain analysis for military purposes and interactive games at home. Spatial Augmented Reality allows to create completely immersive experiences that delude our brain mixing virtual objects in the real scene. The capability of this kind of systems to give our brain a 3D perception of a virtual object mixed on the real world, makes us remember the holograms that constantly appear in science fiction movies, being something that anyone would like to see.

Although Projection Mapping is not a novelty and has been used for more than forty years, as we can see in the figure 1, only in the last years it gained notoriety amongst the community. With the hardware available nowadays, it is also possible to manage these projections over a dynamic scene, allowing a new concept of interactive applications.

There are already some applications that achieve interesting results on the SAR field, also starting to appear interactive ones, however, each of them has a specific purpose, not handling all the concepts in a single application. Due to the wide range of techniques and methods required, the development of projection mapping systems is not a trivial task. The motivation for this work arouse from here, as an opportunity to study the building blocks, and how they can be connected together to build such an application that overcomes the previously mentioned issues.

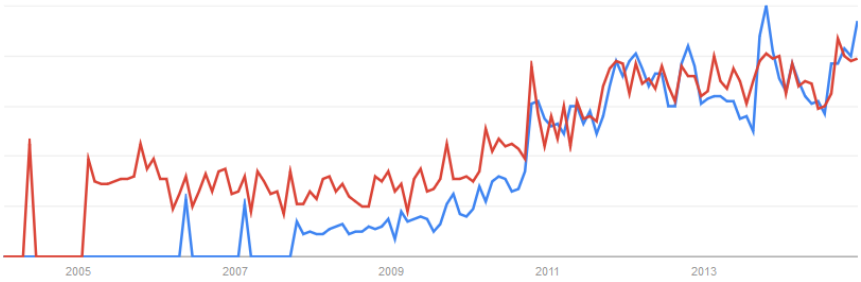


Figure 1: Interest Over Time - Google Trend Web Search. Y Axis represent search interest relative to the highest point on the chart. Red line represents Video Mapping and Blue line the Projection Mapping values.

1.3 OBJECTIVES

In regard to specific objectives for this project, we expect to develop a framework capable of combining the potential of reconstructing a real scenario that can be interactively modified in real-time, with the capabilities of Spatial Augmented Reality.

The framework should be able to reconstruct the real surface intended for projection using the input of a fixed camera, where the scene that the camera is viewing is mutable. The obtained scenario will be merged with a virtual environment, in order to interact with the virtual objects. These objects should be rendered in such a way that when displayed using a video-projector, they fit the real scene accordingly to the scene's shape and viewer's position, giving the impression that the virtual content is really there.

In addition, it is intended as well, to build a demo application using the created framework as a proof of concept.

1.4 DOCUMENT STRUCTURE

The first chapter (chapter 1), the current one, is concerned with the understanding of the theme of this dissertation, contextualizing and introducing the concepts related to the subject.

The second chapter (chapter 2) presents the state of the art, divided in two independent sections relevant to the project: Projection Mapping and Surface Reconstruction. In each of these sections, an overall explanation of the respective topic is given, how it is associated to our project and some of the related work being done in each of those areas that can be used to improve our work.

In the third chapter (chapter 3), the framework is detailed, explaining the architecture, its components, the implementation, and how these different components are combined to achieve the final output.

The fourth chapter (chapter 4) demonstrates the capability of the framework, by making a demonstration application.

In the last chapter (chapter 5), conclusions and future work are discussed.

2

STATE OF THE ART

> "If I have seen a little farther than others, it is because I have stood on the shoulders of giants"
>
> Newton

In this chapter, we will do an analysis to the work that has been done in the different areas that our project relies, beginning by analyzing similar systems and collecting what areas we need to explore in our ideal system. After that, we are going to define the concept of basic Projection Mapping (PM) and describe the fundamental concepts to any SAR system, followed by a more complete description of some of the similar projects exposed in the definition of our system. Subsequently, this process is repeated to the Surface Reconstruction field, expounding its importance, explaining some different categories of methods and grouping some of the known ones. Afterwards, some methods that perform in real-time are discussed. The last section is dedicated to introduce the base theory of common RGB-D sensors and to expose some tools that help integrating these devices in applications or ease the processing of their output data.

There are multiple proposals to achieve a system that projects over a real world surface. In this part of the document some of them will be mentioned with the intuit to understand how this kind of system works and what techniques we will need when implementing such system. The actual explanation of both the systems and the concepts is left for later.

The keyword "Live" can induce many understandings and a system can be considered "Live" under many circumstances. By now, we already defined that the "Live" in our system means that the surface that we project on can be changed in real-time, being the projection adapted to the new surface. When using the word "Live" in a projection mapping system, we can think in other situations. For example, that the projections are being changed in real time accordingly to some interaction, or the projection adapts to the user position as it moves. There is not a previously

fixed description of what a Live Projection Mapping system is, and in this part of the document, we will expose examples that in some way can be considered for the definition of our system.

RoomAlive (Jones et al., 2014) is a good example to start. Their goal was to transform any room into an immersive gaming experience changing its appearance to be the game's environment. In this system they use a virtual model of the room as the scenario of the game. The game elements are then projected over the surfaces of the room giving the user the perception that they are really present in the real world. The projection is adapted to the position of the user, while he can move and interact with the game objects. This system adapts the game to the world in real time using the user position and interaction. Despite the fact that the model of the room is previously reconstructed, not allowing changes in runtime, it fits well in the keyword "Live".

There are recent projection projects using sandboxes that became relatively known, such as the commercial project iSandBOX (iSandBox, 2013). This type of systems adapts the projection to the sand in the box. Using the height map of the surface they simulate volcanoes or the water-flow of the topographical shape formed by the sand. Although this system adapts the projection to a changing scene, it directly maps the surface using only the elevation information, not using a view dependent rendering as RoomAlive. This limits the system to virtual objects close to the surface. If they are distant from the surface or have a notorious height, the rendering needs to adapt to the user position as in RoomAlive to give a correct perception when projected.

Kinect Projection Mapping (Motta et al., 2014) is a system that also take into consideration changes of the surface. It also tries a dynamic projection mapping, although, through the tracking of a human body without its actual surface reconstruction. When the tracked body moves, the projection is adapted to its new pose.

The iSarProjection (Tan et al., 2013) is an interesting system that uses an handheld projector attached to a RGB-D sensor. It is based on Kinect Fusion (Izadi et al., 2011) to reconstruct the display surface. The RGB-D and projector positions are estimated in real-time, adapting the projection to the actual surface changing its appearance.

MirageTable (Benko et al., 2012) merge in some way all the concepts above. It is an interactive system designed to merge real and virtual world on top of a table, allowing physic interaction of the virtual objects with real objects in the scene. The only thing missing in this system is the use of the reconstructed surface also to change the appearance of the real one as in RoomAlive and iSarProjection.

We can summarize now the features that our ideal Live Projection Mapping system should have. The first one that was already defined is that the system should allow to project over surfaces that can change arbitrarily. To achieve this, we need to reconstruct the surfaces that

we will project over, but contrarily to RoomAlive, this reconstruction should be made in real-time. This system should not only map projections to change the surface appearance but also should allow the addition of virtual objects without spatial restriction. To be able to handle this, it should implement a View Dependent Rendering. To increase the interactivity it should continuously track the user position to adapt the virtual objects rendering. The last requirement is that the system should allow physically interaction of the virtual objects with the real surface, being possible to create interactive games using the reconstructed scene. The novelty of this system is that is not only a projection mapping system that changes the appearance of the real world handling surface modifications, but also mixes the SAR notion of virtual objects with 3D perception at the same time.

After having exposed some Projection Systems and described what our ideal Live Projection Mapping system should handle, we will now describe the concepts that we need to explore.

2.1 PROJECTION MAPPING

Projection Mapping (PM) became recently known outside the research community through its increased use in artistic events and advertising campaigns, but it has been used and growing for more than four decades (Catanese, 2013).

Essentially, it is a projection technique that can turn any surface in a display by adapting the image to the respective surface. This technique is also know as Video Mapping and SAR. Although each term is normally used in different contexts, all these terms represent the same general concept: map images to fit some surface which we want to use as a display. Actually, the more common definition for PM is more specific about which surfaces we are referring than in SAR, because SAR includes other technological methods such as screen-based video see-through displays and spatial optical see-through displays (Bimber and Raskar, 2005), whereas PM refers more specifically to the use of projectors directly in real surfaces like buildings, cars, objects or even people. Thus, we can consider PM a specific case of SAR. In figure 2 we can have an overall notion of the possible AR approaches and where SAR and PM are located.

2.1.1 *Calibration and Perspective Correction*

On regular displays that we use at home, the visual information is formed inside its boundaries without any kind of deformation. Each pixel of the display has its value and will remain untouched no mater if the spectator moves or even the display is moved. When we use a projec-

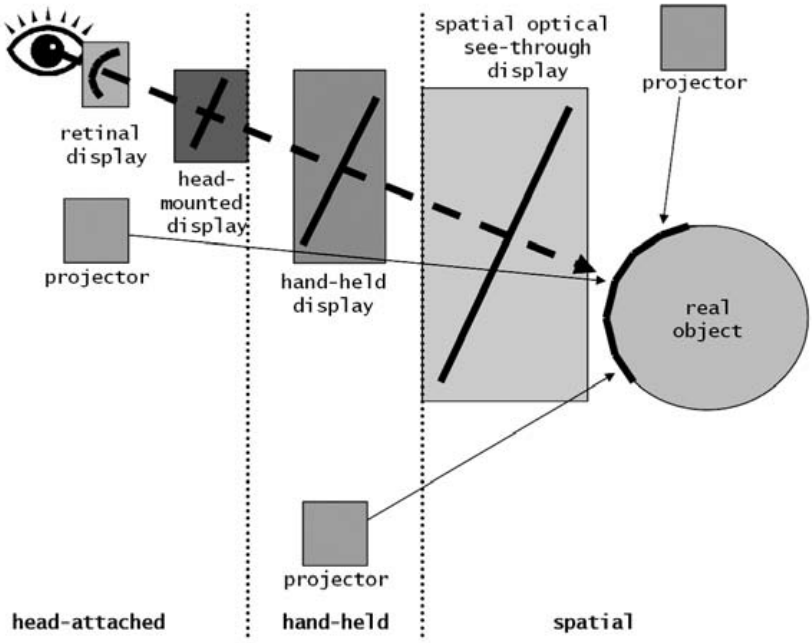


Figure 2: Image generation for augmented reality displays. (Bimber and Raskar, 2005)

tor to turn other objects in a display, a lot of variables have to be considered. The simplest case where the projection goes wrong is when the projector is not perpendicular to a simple planar wall, causing a distorted image. When the target is a non planar surface, the problem worsens, resulting not just in a skewed image but depending of the spectator's position, the image can be unrecognizable. In figure 3 this problem is demonstrated comparing an image taken from the perspective of the projector to an image taken from another viewpoint. In the left image, the projection seems skewed and some of the content is not visible from that viewpoint, however, the projection seems correct from the perspective of the projector. To handle this problem and give the users a correct perception, a transformation is needed accordingly to the positions of the projector and spectator relatively to the display surface, taking also in consideration that surface's shape.

The user's position is important not only for non planar displays, but when AR is added to the subject, the projection does not have a simple frame of a movie that just has to seem undistorted for the viewers, but it may contain virtual content to be added to the real scene, which should give the perception that it is on a specific position on the scene at a certain depth. A new level of perspective notion is added to the variables. If the virtual objects are far from the physical

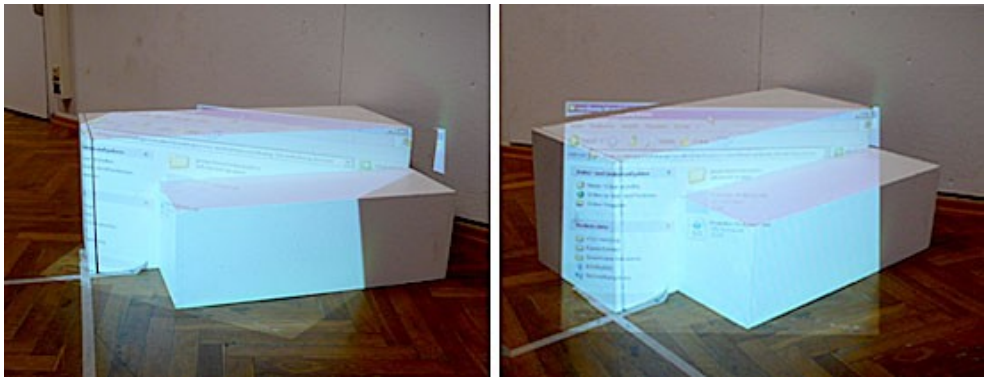


Figure 3: A projection onto an arbitrary surface looks undistorted (ie. exactly as being projected) when seen from the point of view of the projector. (vvvv, 2013)

surface and the viewer's position is inaccurate, a significant displacement error will occur as show in figure 4.

In the figure, we can see that for the user's correct position, the ray intersects the image plane at a certain pixel, however, if the position of the user is badly estimated, the ray traced will intersect at a different pixel giving the perception that the virtual object is at a different position. This type of rendering that uses the position of the spectator is called View Dependent Rendering (VDR), and can also be used with normal displays.

As we can have planar and irregular surfaces, and methods for different types of surfaces are different, we can separate this section in two distinct parts, one that exposes the methods to handle projections on planar surfaces and another where the problems of using irregular surfaces and existing approaches are discussed. Considering that some calibration methods can be used in both situations, it makes sense to have a first section exposing the existing calibration methods, and then, separate how their results are used in both cases.

Projector Calibration

In this document, the meaning of projector calibration is to find the geometric relationship between the projector and the projection surface, as well as its intrinsic parameters. These components are needed to obtain an undistorted and properly sized image on the target surface. Similar to Projector Calibration, Camera Calibration is a well known problem in Computer Vision, whose purpose is also to find it's extrinsic and intrinsic parameters. Before we expose the way the calibration of a projector is made, the common concepts are introduced in a simpler case, starting

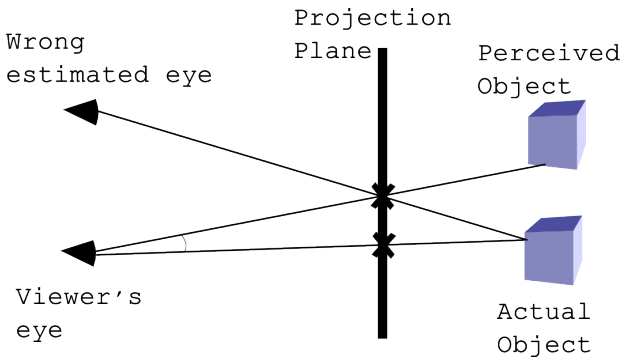


Figure 4: Effect of Displacement Error in a View Dependent Rendering. If the user's position is badly estimated, the final rendering will give the perception that the virtual objects are in a certain place, when in fact, they are displaced from the correct position.

with a single camera calibration to ease the explanation and relate it later with the methods to calibrate the projector.

As mentioned before, we need to calibrate the system to obtain the properties of an optical device such as a camera or a projector. The most common model used to describe the properties of a camera is the pinhole camera model (fig. 5 and fig. 8), and the calibration process has the purpose to find its parameters. To understand the result of the existing calibration methods, a quick description of the intrinsic and extrinsic parameters (fig. 7) can be made by saying that the intrinsic parameters are related with the internal properties of the camera, like focal length, center of image, and distortion coefficients, while extrinsic parameters define the 3D position and rotation of the camera relative to a known world reference frame. A more complete description of what each value means can help to understand better the process.

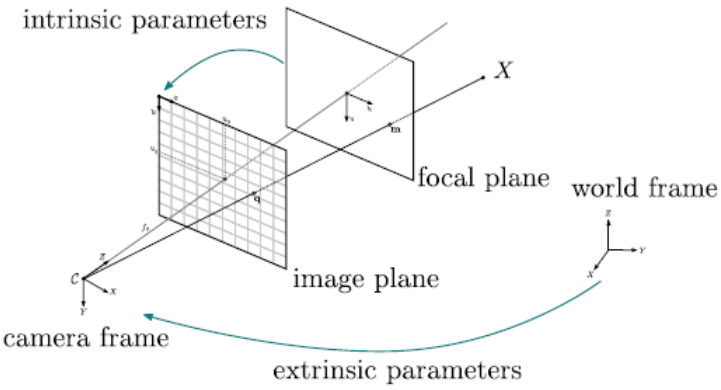


Figure 5: Pinhole camera model diagram. (openMVG, 2015)

The focal length parameters (f_x and f_y), are responsible for defining the angle of view and magnification in each axis, this is, how much of the scene will be captured. A higher focal length, leads to a smaller angle of view, which means that only a small part of the scene will be captured. This leads to a higher magnification. A smaller focal length has the opposite effect. Figure 6 illustrates this relation between the focal length and the angle of view. c_x and c_y parameters, represent the principal point, which is the position of the intersection of the principal axis and the image plane. Ideally, this position would be in the center of the image.

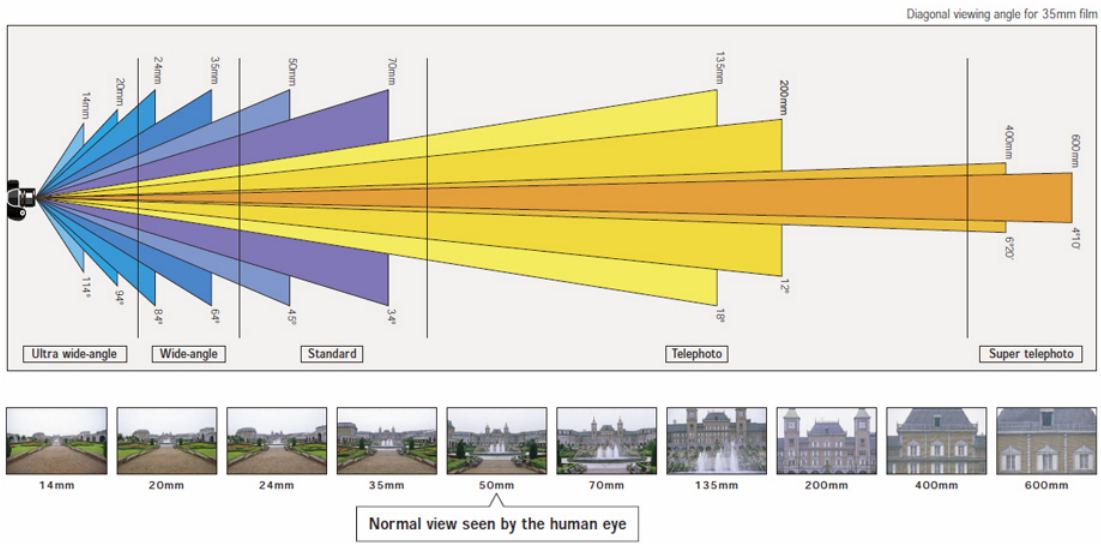


Figure 6: Focal Length simulation - Smaller focal length implies a wider angle of view and vice-versa. (Panasonic)

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 7: The intrinsic parameters compose the matrix represented in the left. The matrix in the right side represent the extrinsic parameters.

The extrinsic matrix is composed by two different parts, a rotation matrix, represented in the first 3 columns of the matrix, and the translation vector, the last column of the matrix. The sub matrix containing the rotation matrix provides the orientation of the camera. It is composed by three basis vectors, the Right Vector $[r_{11}, r_{21}, r_{31}]$, the Up Vector $[r_{12}, r_{22}, r_{32}]$ and the Forward Vector $[r_{13}, r_{23}, r_{33}]$, also called as Look-At Vector. The last column contains the translation of the camera.

Actually, the transformation described by the resultant extrinsic matrix of the calibration process does not represent the pose of the camera, represent instead, the position and orientation of the origin of the world coordinate system expressed in coordinates of the camera-centered coordinate system, where the origin of the world coordinate system is positioned in one of the known 3D points used in the calibration process explained in the next paragraphs. To obtain the extrinsic matrix as explained and position the camera from world coordinates, the inverse of this transformation should be used.

$$sm' = A[R|t]M'$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Figure 8: The pinhole camera model maps real space to image space. The intrinsic values represent the focal parameters and compose the camera matrix A, the extrinsic values represents the displacement between the camera/projector and a known world reference frame, composing the matrix $[R|t]$. This equation of the pinhole camera model demonstrates how real world coordinates M' relate to image coordinates m', by the camera matrix A and the rotation and translation matrix $[R|t]$.

To calculate the intrinsic and extrinsic parameters of a optical device, all methods need to estimate a series of points p1, p2, ..., pn found in its image plane, that matches to a series of known points P1, P2, ..., Pn in the three dimensional space. The intrinsic and extrinsic parameters are then calculated in a way that the virtual points p'1, p'2, ..., p'n, which represent the known points, when reprojected using the obtained parameters, overlap the observed points, minimizing the distance between the reprojected points and the observed ones.

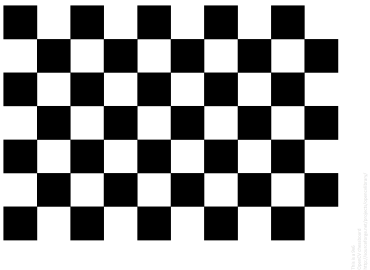


Figure 9: The most used calibration pattern to represent the known points in the world space is a chessboard pattern with know dimensions.

The most known procedure is the Zhang's method (Zhang, 2000), implemented in multiple frameworks such as OpenCV and Camera Calibration Toolbox for Matlab, being also one of the most used methods. Its complete procedure follows the next high-level steps:

1. Print a pattern and attach it to a planar surface
2. Take a few images of the model plane under different orientations by moving either the plane or the camera
3. Detect the feature points in the images
4. Estimate the intrinsic and extrinsic parameters
5. Estimate the coefficients of the radial distortion
6. Refine all parameters through a minimization process.

Although projector calibration can be made in the same way as we can calibrate a camera, projectors cannot take an image of the surface that they illuminate. Therefore, to do the correspondence between the 2D projected points and the 3D illuminated points, existing techniques use a camera based feedback, calibrating the projector seeing it as the inverse of a camera, wherein instead of light coming from real objects and incide on a projection plane, the light rays are seen as being emitted by the projection plane and passing through an optical center before inciding on the real object (Miranda et al., 2009). This assumption allow us to calibrate a projector like we calibrate a camera, considering that two devices viewing the same scene, see objects from a different perspective. This relation between different perspective images is called homography and in planar surfaces there is always a 2D homography between image planes of any two projective imaging devices (Park and Park, 2010). Due to this facility with planar surfaces, the majority of calibration methods are based in 2D patterns like a chessboard with known dimensions illustrated in figure 9. The calibration can also be made with a 3D apparatus with known points instead of a planar one, being possible to achieve a higher accuracy than with a 2D apparatus. In (Zhang, 2004), the researchers feedback is that the use of a 2D apparatus seems to be the best choice in most situations considering its ease of use and good accuracy.

In projector calibration, despite not having an image of its viewpoint, by projecting a pattern, we know the information in its projection plane without any image processing, and detecting that pattern with the camera, its possible to calculate the related pixels between the camera and the projector. After this process of calibration, it's possible to calculate the intrinsic and extrinsic

parameters, which will be used to obtain a correct projection using the pinhole camera model (figure 8).

Known methods to identify this relation between camera and projector pixels, can be divided in two groups, the ones that project a simple pattern like the chessboard one (fig. 9), and the ones who use a Coded Structured Lightning (CSL) (fig. 10). The methods in the first group, project a known pattern like the one used in a simple camera calibration. With that pattern projected, they capture an image with a pre-calibrated camera, detecting the marks of interest. Using the information obtained from the camera calibration process, they estimate the 3D position of the projected marks. With the 3D points calculated the rest of the process is the same as a normal camera calibration. Falcao et al. (2008) is one of the most known of the methods belonging to this group.

Methods on the second group use coded patterns trying to find a match for every pixel in the projector with a pixel in the camera. These techniques are also known for being used in surface reconstruction. The methods using CSL can be subdivided in the ones based on spacial codification, time-multiplexing or direct encoding. The most used methods, allowing higher accuracy, are based in the time-multiplexing concept (Salvi et al., 2004). These methods don't use a single pattern, instead, a sequence of patterns are successively projected over time. The basic idea of these techniques is to reduce the process of finding the matching pixels to a kind of binary search through the sequence of patterns, reducing the possible matches as more patterns are decoded. A more complete explanation of the time-multiplexing calibration technique will be made in section 3.1.1.

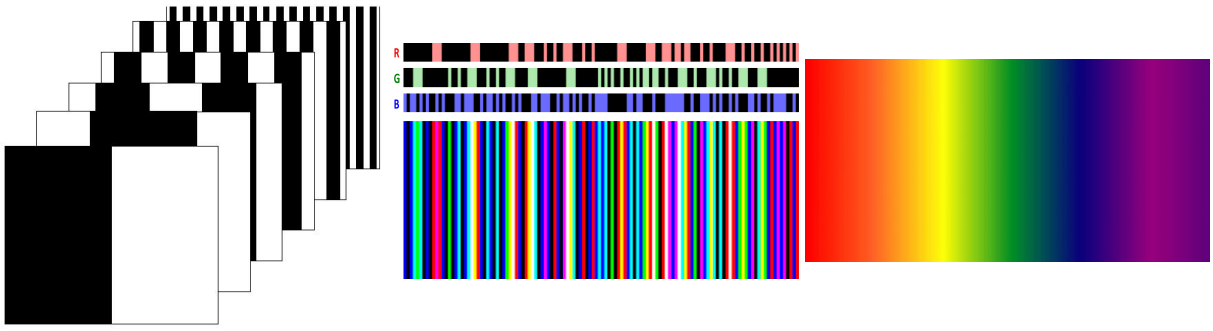


Figure 10: The left image (Chen, 2012) represent a time-multiplexed method, where a sequence of patterns are projected. The image in the middle is an example of spatial encoding pattern, using a color stripe indexing based on De Bruijn sequence (Geng, 2011). The last pattern uses the full RGB spectrum trying a direct decoding.

In Lee et al. (2004), the same concept of these structured light methods is used, although, instead of a camera feedback, they use light sensors in specific positions of the target surface. The use of the structured light patterns allows each sensor to discover its location in the projector's plane, analogous to the detection of the known markers in projector's plane in the camera based feedback methods.

Planar Surfaces

When the augmentation is done on Planar Surfaces, projectors are typically mounted so that their optical axis is perpendicular to the planar display surface, trying to avoid oblique projections simply by design. However, it's difficult to maintain a position where the axis is exactly perpendicular, and after a period of time it can become oblique due to mechanical or thermal variations, resulting in a keystoned and distorted image (Bimber and Raskar, 2005). To correct the projected image, a calibration of the projector is needed, calculating the parameters that we need to correct the image.

Those parameters can be obtained through any of the calibration methods exposed in the previous section (2.1.1), and as exposed there, because we are dealing with planar surfaces, there is always a 2D homography between image planes of any two projective imaging devices. With the obtained parameters an homography matrix can be computed, allowing the warp of the source image to fit onto the projection surface.

As the objective in SAR is not only to project a simple image on the surface but a virtual world rendered image, an additional problem needs to be solved. We can now warp the rendered image to correct the oblique projection due the position of the projector, but the rendered image also needs a transformation to match the perspective of the user. In planar surfaces, this is also solved using the homography technique as shown in figure 11. Although the perspective can be corrected using the homography between the projector and the user, objects occlusions can be different between both perspectives, generating a wrong image. To resolve this problem the depth buffer needs a transformation too. Applying the homography to the depth buffer brings some problems, because a clip of the values is needed, resulting that some parts of virtual objects may not be rendered. In the rendering framework by Bimber and Raskar (2005), they solved this problem using an approximation of the depth buffer in it's transformation step, to avoid the clipping with the near and far planes.

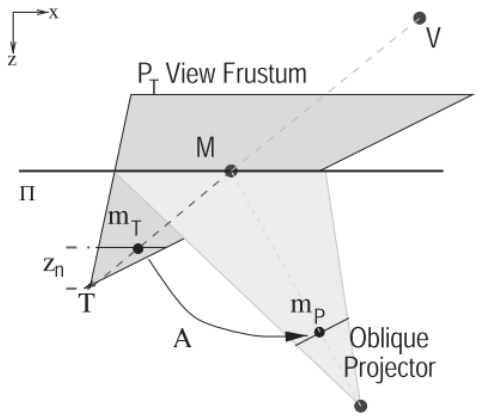


Figure 11: Rendering for a planar surface, T being the user position and V a Virtual Point. The point M is the one responsible for shading the pixel mT, being the closer point in the surface that intersects the ray between the user and the virtual point. Both pixels mT and mP are back projections of the point M, being related by a homography. (Bimber and Raskar, 2005)

Non Planar Surfaces

Projection based AR on non planar surfaces, i.e. real objects, works very well when the virtual objects are relatively close to the real ones. Usually, in artistic and advertising events much of the objects are just manipulated to simulate alternative appearances, alternating shading and lightning, but the virtual object represents only the real one with the same shape and position. In (Raskar et al., 2001) they replaced physical objects with a replica with neutral material, and reproduced the original and alternative appearances directly on the object using projections, since then, the use of projectors to change objects material is often called as Shader Lamps. That being said, we can assume as a requirement to have a 3D model that represents the real surface. A manual replica model can be used, but for complex scenarios where objects can change regularly we may need a surface reconstruction algorithm to obtain the object's model. In section 2.2 a review of this field is presented.

To understand the following explanations, we need to have in mind that can exist two types of virtual objects. The ones that are representing the real scene, allowing only the change of its appearance as in Shader Lamps, and the virtual objects that are intended to augment the real scene. In the next references to virtual objects without any other indication, we are referring to the ones that augment the real world.

As the display is not planar, we can't relay on the homography technique to warp the resultant image as we do on planar surfaces, because the assumption that there is always a 2D homography

between image planes of any two projective imaging devices, is only valid when the surface that both projective imaging devices are poiting is planar. With irregular surfaces, depending on the positions of devices, the user, the virtual objects and the surface relief, there can be gaps between the images of the two perspectives.

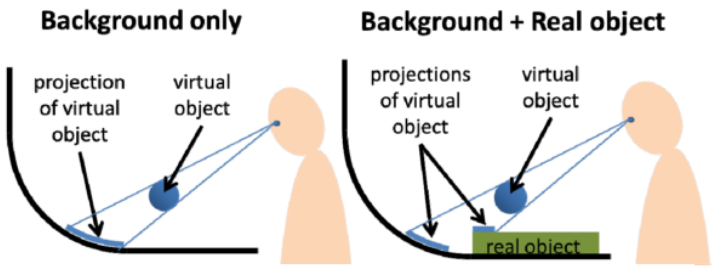


Figure 12: Projective texturing requires that the system takes into account the geometry of real objects in order to correctly present perspective 3D virtual information to the user's eye. (Benko et al., 2012)

To deal with this problem, all of the referenced frameworks that works on irregular surfaces of real objects, use the same technique, a multi-pass renderer based on the Projective Texture concept (Everitt, 2001) (fig. 12).

In the first step of the rendering process, the virtual objects in the scene are rendered to a texture map from the perspective of the user. Then, this texture is projected from that same perspective onto the objects that represents the real-world scenario. After this step, the objects representing the real-world display have the referred texture mapped on them. In the next step, they are rendered from the perspective of the projector. When the rendered image in this second step is projected onto the real surface, the user will see a correct perspective view of the virtual object over the real surfaces. This process is illustrated in figure 13 and figure 35 in section 3.3, where this technique is explained. This multi-pass technique can be simplified in specific cases, as in the case of planar surfaces.

By now, we already noticed that a simple homography applied to the image obtained from the user position won't generate a correct projection from the projector's viewpoint as in the planar cases. As explained in the previous paragraph, the virtual world needs to be an exact replica of the real world, having not only the shape of the surface but also a virtual projector and a virtual viewer representing the real ones, in the correct relative positions. As in the planar cases, a calibration of the projector is needed to obtain the intrinsic and extrinsic parameters of the projector, but instead of constructing an homography matrix, these parameters are applied to the

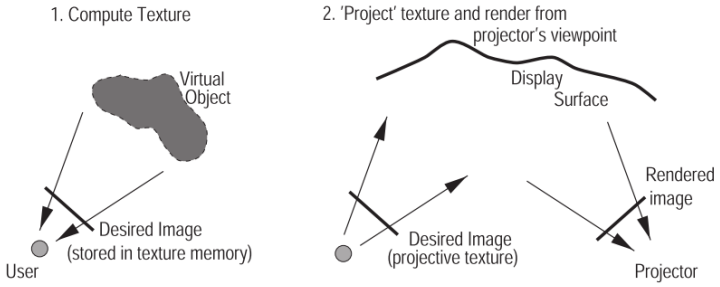


Figure 13: Two-pass rendering for a non-planar display surface involves forward and backward mapping. (Bimber and Raskar, 2005)

virtual projector, simulating the viewpoint of the real one. The pinhole camera model remains valid, positioning the virtual projector through the calculated extrinsic parameters. The intrinsic parameters are also applied to the virtual projector defining its internal properties.

2.1.2 *Generic Problems*

Projector Calibration and Perspective Correction are the most important problems that must be overcome on SAR applications, but they are not the only ones. There are other problems as obtaining an accurate 3D model of the surface that will be used as a display, the user tracking to correct the perspective dependently on its position and many others that will be discussed in this section.

User Tracking

As we have seen in the previous section (2.1.1), knowing the user's head position is critical to render a correct perspective, but also to obtain other view-dependent effects, such as reflections. An erroneous position may lead to an incorrect perspective from the user, perceiving that a virtual object is in an incorrect place as exemplified in image 4. This error can be somehow ignored in cases where the distance between the user and the display is significant, and the virtual objects are close to the display. This is easily confirmed by analyzing the displacement error equation 1, as demonstrated in Cruz-Neira et al. (1993). If the virtual object is on the same position of the

display object, the error due to the displacement is none, because the distance Z and PD are the same.

$$\boxed{DISP = \Delta P(\frac{Z - PD}{Z})} \tag{1}$$

Methods to track the user's position can use mechanical, electromagnetic and optical tracking, the last one being the most used in AR systems due to it's low-cost relatively to the others. Each of these methods can be used in two ways, a first one where the system is compound by fixed sensors within the environment, tracking the moving target, and another where the hardware is incorporated onto the user. Usually, when tracking users, the goal is to use the most non-invasive method, being the first one the preferred method because the user doesn't need to use equipment. Recently, many applications use algorithms like Fanelli et al. (2011), that rely on depth data to estimate the head position. Some examples of this use are in RoomAlive Jones et al. (2014) and Dyadic Benko et al. (2014), where depth sensors fixed in the scene are used to track user's position. Benko et al. (2012) uses the depth image not only to find the head's position but to find the shutter glasses, which will give a more precise position of the eyes and its orientation.

Occlusions and Overlaps

Another issue that we face when using projections is the occlusion of some regions in the scene since all visible surfaces should be lit. The base concept of the solution is simple, we can illuminate the shadowed parts by using additional projectors. Although this solves the occlusion problem, it adds another layer of complexity, the merge of images from multiple projectors. Using more projectors creates overlaps that will increase the incident intensity in overlap zones, that we have to correct. A simpler solution could be the attribution of a single projector to each surface patch, but that will require an exact geometric calibration between the projectors or it can still lead to gaps and overlaps. As we know, there is no linear solution to make the alignment error equal to zero (Park and Park, 2010) and mechanical variations can increase the error over time, so the use of this technique will lead to artifacts. Another problem is the calibration of colors between projectors, as the manufacturing process and temperature influences the produced color (Bimber and Raskar, 2006), even with an exact calibration, there will be color differences between neighboring projectors. The solution to achieve smooth transitions in the overlap is to assign an intensity weight for every pixel in the projector, so that the sum of the intensity weights of the corresponding projector pixels is 1 and varies smoothly in order to avoid the color differences between neighboring projectors creating visible artifacts. This is know as cross-fading.

The following image (14) illustrates the concept of this methods, and includes a representation of the technique implemented in (Raskar et al., 2001).

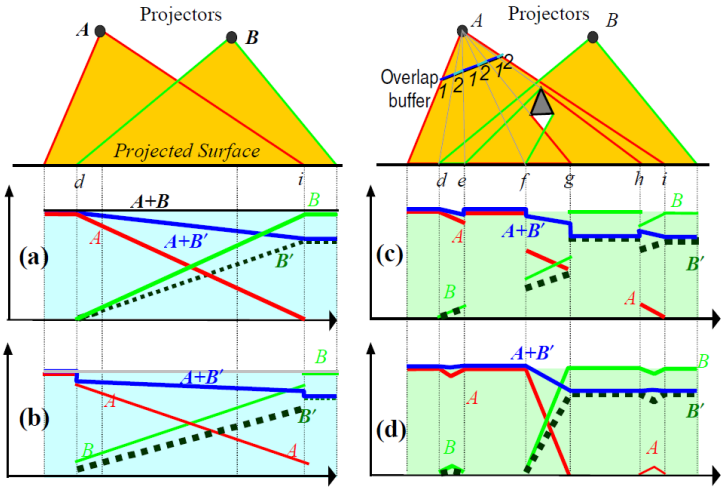


Figure 14: Intensity weights using feathering methods and the plots show the contribution of projectors A, B and B' and the resultant accumulation A+B and A+B' along the lit planar surface. (Raskar et al., 2001)

Intensity Correction

As we saw in the previous section, knowing the shape of the display surface is crucial to adapt projections to non-planar surfaces, but common surfaces have others properties that can change the expected image such as varying reflectance and color. In Bimber and Raskar (2005), the intensity of the output image is modified to take into account the reflectance of the neutral surface, the local orientation, and the distance with respect to the projector. To avoid the low sampling rate of the projected pixels and to minimize the misregistration artifacts they do not illuminate highly oblique surfaces.

Recent radiometric compensation techniques allow the projection of images onto colored and textured surfaces. With projector-camera systems the surface is scanned on a per-pixel basis, that will be used to calculate a compensation image that neutralizes color blending caused by underlying surfaces. Grundhofer and Bimber (2008) describes how to achieve this compensation in real-time. There are already smart projectors Bimber et al. (2005) that also use cameras to sense their environment and manipulate the rendered image to compensate for color blending and do geometry corrections.

These compensation techniques relay on the reflection of the surface, as the camera needs to capture the light to calculate the luminance and color of the scene. If the display surface's material absorbs the light, the correction will fail as it does not have the information of the incident light. Fortunately, most diffuse materials produce acceptable results, and absorbent materials are rare in usual environments.

2.1.3 *Related Applications*

In this subsection some related existing applications are described, exposing in figure 15 some images of the described applications. As said early, the hardware evolution allowed the use of the above techniques in dynamic scenarios, updating every frame the projection relatively to the perspective of the users and it's input. There have been multiple recent projects on this field with many differences and goals that already use that capacity. Some projects just track the user and update the rendering perspective, others allow the user interaction with virtual objects but have fixed scenarios and others also allow the change of the scene in real-time. There are plenty others beside the described below, such as *Shader Lamps* (Raskar et al., 2001), referenced previously, or Wilson and Benko (2010), but all of them follow the same concepts of the described ones.

RoomAlive

RoomAlive (Jones et al., 2014) is a good example of what this genre of frameworks can do. Theirs goal was to transforms any room into an immersive gaming experience changing its appearance to be the game's environment. The system is compound by multiple units that handle a region of the room. Each unit contains a projector, a depth camera and a computer. These unit projectors are automatically calibrated, having an overlap zone between them that is used to establish dense correspondences between each projector pixel and all depth sensors that can observe that point. These correspondences are used to calculate the intrinsic and extrinsic parameters of each unit. The 3D model of the room is generated off-line by combining the depth maps from each unit and planar surfaces are labeled. This process needs to be repeated when the scene is changed or a recalibration is made. After the 3D model acquisition, the virtual content is mapped on the scene automatically.

The user can interact with the virtual content being processed locally on the respective unit, and only if the changes in game state are synchronized across units. As the room has multiple surfaces, the two-pass view dependent rendering described on section 2.1.1 is used. This technique does not handle the presence of multiple users, but for scenes where the virtual content

is near the physical surface, the perspective does not change significantly. Theirs solution to this problem for these closer content is to render with the average head position, but does not resolving the perspective error of virtual objects that are far from the physical display surface.

MirageTable

MirageTable (Benko et al., 2012) is an interactive system designed to merge real and virtual world on top of a curved screen. It uses a depth camera, a stereoscopic projector, shutter glasses and a curved screen where the projector will be aimed. The camera and the projector are calibrated and the geometry of the curved screen is captured to correct projections perspective. The captured depth map can also be used to ease the segmentation of new objects and body parts in the scene. The user's eyes are occluded by the glasses, using that as an advantage to track them, because the depth map of the head will have holes at the glasses position, as their reflectivity disturbs the depth values obtained by the sensor. Using the estimated position of the eyes, the projective texture approach previously described is used to render a correct perspective for the user. The system uses the depth camera to scan the scene, using the obtained geometry and texture to render them on the curved display like it was a mirror. As there's only one depth camera, the object is filled and its back surface is mirrored from the scanned one, resulting in better results for symmetric objects. The user faces the display and sees the objects from the opposite direction compared to the depth sensor and as the projector shows them. Using the obtained geometry of the real objects, it simulates physical interactions between any kind of object.

Dyadic

Dyadic projected spatial augmented reality (Benko et al., 2014) is a little different from the previous mentioned projects. They also call theirs system as *Mano-a-Mano* because it enables two users to interact with virtual objects in a face to face interaction. To render a correct image for each user two opposite facing projectors are positioned so that each one displays the view corresponding to the user standing under it. With this configuration, a virtual object between the users is rendered twice. This multiple projection of the same object may be disturbing when users can see the rendered object intended for other users, but as this system is a specific case of multiple users (face-to-face), the object will be projected on the wall behind them or on their own bodies, being unlikely to see the projection destined to the other. As any system of this type needs the geometry of the scene, the room and static objects are handled off-line as in Mirage Table. The user's bodies are also part of the scene, so they have to be scanned in real-time.

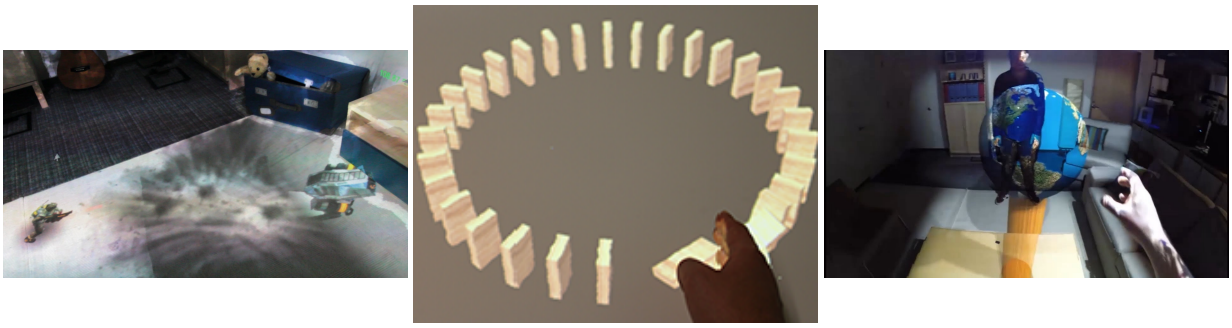


Figure 15: Related Applications. Left image is from RoomAlive, the middle from MirageTable and the last one from Dyadic.

2.2 SURFACE RECONSTRUCTION

Obtaining a three dimensional representation of objects in the real-world is a feat pursued by Computer Graphics researchers over the last decades, resulting in the existence of a lot of different techniques. SR methods are widespread because they have a broad scope of activity, being useful in applications from biology/medicine to reconstruction of buildings and towns, attracting people from many fields. Recently, it got even further attention mainly due to the ability to acquire 3D point clouds with affordable devices, depth sensors, increasing even more it's popularity. These devices revolutionized the Computer Graphics field, allowing its use in casual entertainment applications such as recreating scenarios in the Virtual World. The hardware available nowadays, allows to reconstruct surfaces in real-time which makes possible its use for dynamic interaction in augmented reality applications, leading to a more authentic mixed reality experiences. Camera based reconstruction techniques are not described here, neither the point-cloud obtainment methods, assuming that the reconstruction is based on point clouds already acquired.

2.2.1 *General Algorithms*

There are many algorithms of SR, each of them has its purposes, leading to advantages and disadvantages consonant the surface to reconstruct and the data that represents it. In this section an explanation about different types of algorithms is given, categorizing various methods and explaining the basic processes of techniques belonging to each group. There are also many ways to categorize surface reconstruction methods. First an explanation of some categorizations by their implementation approach and algorithms is made, followed by a more general categorization, which can help choosing a method knowing only the expected type of object.

Approach Categorization

Reconstruction methods can be categorized as Explicit or Implicit (Spickova, 2013).

EXPLICIT Explicit methods come from Delaunay-based algorithms and they have two main issues, the high sensitivity to noisy data and being very time-consuming. The advantage of these methods is their theoretical guarantee. The theorem states that if the point set satisfies a dense enough sampling, the restricted Delaunay triangulation of the input data set with respect to the surface is homeomorphic to the surface. However, such sampling can not be met at sharp

features (Wan et al.; Lafarge and Alliez, 2013). The requirement of such point set leads to a lot of reconstruction difficulties when handling noise and outliers. Methods such as CRUST in Amenta et al. (1998) and the Alpha shape in Edelsbrunner and Mücke (1994) are methods belonging to this category.

IMPLICIT Implicit methods reduce noise and have a lower computational cost, however, do not guarantee an exact approach to the real surface as explicit methods. While explicit methods use the actual point set to reconstruct the surface, implicit methods indirectly describe the surfaces using level-sets (Hornung and Kobbelt, 2006). They are based in the computation of a function from the raw point set, allowing to approximate and smooth the surface. This function normally discretizes the space through a signed distance function, also dividing the space in inside and outside of the surface through the signal of its value. The step that actual reconstructs the surface uses that function instead of the input point set. This category of methods can be subdivided in another two classes. Local methods are very accurate, as they can handle with large data sets. Algorithms such as Moving Least Squares (Cheng et al., 2008) and Partition of Unity based (Ohtake et al., 2005a) methods are included in this group. Global methods can obtain good results using low quality data input. Fourier, Poisson (Kazhdan et al., 2006) and RBF (Carr et al., 2001) are some algorithms that belongs to this group.

Algorithm Categorization

In Spickova (2013) and Berger et al. (2013), they also divide the methods by the algorithm they employ. The division nomenclature is also used to describe the basics of the different algorithms.

INDICATOR FUNCTION Methods of this class reconstruct the surface of a three dimensional object O by finding the scalar function χ, which determines if the points are inside or outside the model, and extracting an appropriate isosurface. This scalar function is known as the indicator function.

Poisson, Fourier or Wavelet surface reconstruction, are some examples of methods using this principle. For instance, Poisson surface reconstruction (Kazhdan et al., 2006) takes advantage of the points oriented normals and correlates them with the indicator function of the model. The gradient of the indicator function is a vector approximated through the normal-field, which will have an equal value to the inward surface normal at points near the surface and zero everywhere else. The indicator function is derived from this gradient field allowing the extraction of the object's surface. The extraction of the iso-surface from the calculated indicator function can be

done using a method like the Marching Cubes (Lorensen and Cline, 1987). Figure 16 illustrates the described process.

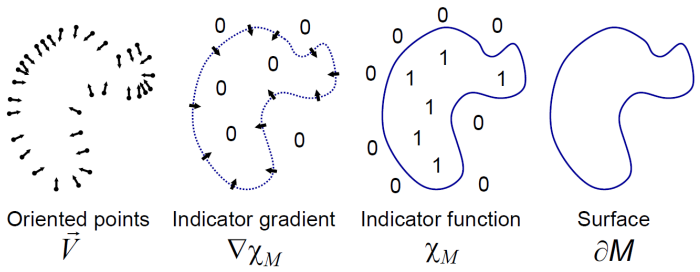


Figure 16: Intuitive illustration of Poisson reconstruction in 2D. (Kazhdan et al., 2006)

POINT SET SURFACE Point Set Surface methods use a projection operator to define a surface by its representation points. They are based on the Moving Least Squares algorithm, using it to approximate the surface with polynomials. Figure 17 illustrates the process. In the first image the input points and its defined curve are represented. The smooth surface is re-sampled through some points belonging to that curve, called as representation points, usually composing a smaller point set than the original.

Although the original definition were to handle unoriented points, when taking advantage of oriented normals like Poisson, its definition is simplified, allowing the use of its implicit surface definition instead of its projection operator.

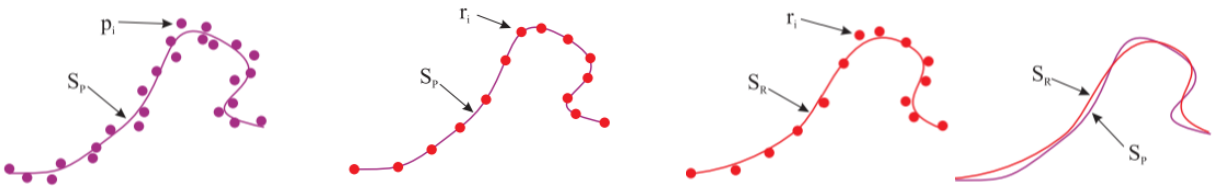


Figure 17: An illustration of the paradigm: The possibly noisy or redundant point set (purple points) defines a manifold (purple curve). This manifold is sampled with (red) representation points. The representation points define a different manifold (red curve). The spacing of representation points depends on the desired accuracy of the approximation. (Alexa et al., 2003)

MULTI-LEVEL PARTITION OF UNITY Techniques based on Multi-level Partition of Unity approach the reconstruction problem as a hierarchical fitting problem. Given a set of points sampled from a surface, the closer they are from the surface, the more accurate is the approximation

of the Signed Distance Function (SDF) that is provided by the Moving Least Squares implicit function. To create an implicit representation, a box that bounds the point set is constructed and sub-divided using an octree-based system. A quadratic function that fits the points in the respective cell is created for each one of the octree divisions. If that shape function approximation is not accurate enough the cell is subdivided and the procedure is repeated until a certain accuracy is achieved. Once all shape fits have been performed, an implicit function over the entire volume is formed by smoothly blending nearby fits.

SCATTERED POINT MESHING These algorithms are based on the Garland-Heckbert local quadric error minimization principle. Weighted spheres are covering surface samples, which grow around the respective point to determine the connectivity in the output triangle mesh. They have three main steps (Spickova, 2013) that are illustrated in Figure 18. Figure 18a and 18b represent the input data without and with unit normals respectively. The first step is responsible for reducing noisy data, resulting in the points of figure 18c. The adaptive sparse spherical cover is created in the second step, also represented in fig 18c, analyzing here the connectivity between points. The last process is the cleaning of the redundant connections 18e.

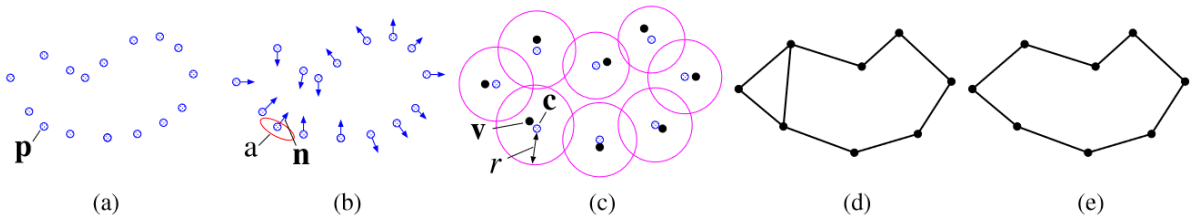


Figure 18: Illustration of the main stages of mesh reconstruction from scattered data in 2D. (Ohtake et al., 2005b)

Priors Categorization

There are other ways to categorize reconstruction methods such as in Tishchenko (2010), dividing in Region Growing, Computational Geometry and Algebraic methods, or other example, if they relay in Oriented or Unoriented Normals, but instead of explaining all the possible divisions, we can focus now in the division of the reconstruction techniques in a more perceptual way, based for example in the object's expected shape. This separation will help us decide which method to use according the expected object and data. In *State of the Art in Surface Reconstruc-*

tion from Point Clouds (Berger et al., 2014), they propose a categorization based on the existence of properties and imperfections in point clouds, and in the nature of the object's shape. We can call this division as by assumptions or by priors. Hereafter, a simple explanation of what each group consists and examples of algorithms that belongs to the respective category are given.

SURFACE SMOOTHNESS Methods of this category, were designed to handle data with varying sampling densities and noise artifacts, supporting data that may contain regions with no measurements at all (missing data), assuming that the object's surface is watertight. These methods can be divided into local smoothness and global smoothness, but both have the surface smoothness to constrain the output surface. Local smoothness based methods ensure the smoothness of the reconstructed surface in regions with depth information, failing to reconstruct the correct global geometry and topology. This happens because only the local neighborhood of that point is used to compute a point of the surface. Moving Least Squares and Partition of Unity are both local smoothness methods. Methods based on algorithms like Radial Basis Functions overcame this problem prescribing that the entirety of the output surface is smooth. These methods use Global Smoothness Priors.

VISIBILITY Because of the general assumption of Surface Smoothness methods, they can support many types of inputs, but this assumption sets limits on the magnitude of the artifacts which they can handle. To manage complex artifacts, it is useful to consider specific priors. This category of techniques consider visibility as a prior, and usually, makes little assumptions. There are three classes of methods inside the Visibility group. The Scanner Visibility, which is used to obtain the line of sight associated with each sample, Exterior Visibility, that uses line of sight that is not provided from the scanner, but rather approximated from the exterior space, and the last one, which uses the number of times a ray intersects a surface in order to approximate the interior and exterior, called Parity. The first class is probably the most popular and is usually used in techniques that merges individual range scans. Curless and Levoy (1996) is a known method of this group, where the SDF is updated for each scan merged. Recent projects that run in real-time are based in this method, such as *KinectFusion* (Izadi et al., 2011) and its derivatives.

VOLUMETRIC SMOOTHNESS Other way to deal with complex cases of missing data is to measure the local shape volume and make it vary smoothly. However, the local thickness measurement is another complex problem, existing various ways to resolve this issue. An example implementation of this approach is described in Sharf et al. (2006), which slowly grows multiple

surface fronts guided by a signed distance field, from within the interior volume out to the point cloud.

GEOMETRIC PRIMITIVES In situations where the target objects have simple surfaces as in the cases of Computer-aided Design for some architectural models, the reconstruction can be made by composing simpler geometric primitives. With this in mind, the detection of simpler geometric primitives has shown extremely helpful in handling noise and missing data. In Schnabel et al. (2009), they use this concept and perform reconstruction aligning and merging the boundaries of adjacent detected primitive's planes.

GLOBAL REGULARITY There are other properties that can help handling artifacts in the point cloud, such as the existence of symmetry, repetition and canonical relationships. The assumption that an object surface has these properties eases the process of reconstructing regions with missing or noisy data. Man-made objects consists of primitive faces conforming to various global relations, thus these properties are present in almost all of them. A building for example, is composed of multiple facade elements. An example implementation is described in GlobFit (Li et al., 2011), which presents an algorithm that simultaneously recovers a set of locally fitted primitives along with their global mutual relations, assuming that the data corresponds to a man-made engineering object consisting of basic primitives, possibly repeated and globally aligned under common relations.

DATA-DRIVEN In practice, some shapes are extremely challenging and simply do not fit the previous discussed priors. A possible way to handle this issue is the use of a data-set with known shapes. This allows the reconstruction of multiple types of objects with the same method, only requiring the presence of a similar object in the collection of shapes. When the reconstruction is impossible, the most similar object in the data-set is fetched and, if necessary, deformed in order to fit the input data. Shen et al. (2012) presents a technique that combines existing labeled parts with respect to the acquired data, allowing the reconstruction of high-level structures from data with a relative amount of artifacts. Baak et al. (2013) is an example of a data-driven method that reconstructs a full human body pose in real time.

INTERACTIVE Although data-driven methods can handle complex data, sometimes the interaction of the user can ease the procedure and help obtaining a more perfect result. Interactive methods integrate the user interaction with the reconstruction algorithm and can also be called

user-driven. An example of this approach is Arikan et al. (2013), which creates an initial model by extracting planar primitives and allows the user to sketch modifications. These user generated polygons will be automatically aligned to the model.

Table (1) presents a more complete list of algorithms accordingly to the previous described groups, indicating which point cloud artifacts they can handle, theirs input requirements, the shape class which is intended for and the form of the reconstruction output.

2.2.2 *Real-time Reconstruction Algorithms*

As said earlier, the ability to reconstruct 3D surfaces is of special importance in Augmented Reality, and there are already gaming platforms such as Vuforia (Qualcomm) that require the user to scan an interior scene from multiple angles. These entertainment applications can be even more immersive if the reconstruction can be performed in real-time, making possible dynamic modifications in the scene while playing.

There are already multiple proposed methods to achieve a real-time reconstruction, almost all of them using the same principle and an identical work-flow. Baak et al. (2013) is one of the few exceptions, being a hybrid data-driven method that reconstructs a full human body pose in real-time. Data-driven methods have some disadvantages and all of them have limitations when dealing with complex scenarios. In the referred method, the downside is that the reconstruction is specific for human bodies. There are other examples that also can achieve a good performance using databases that cover all the poses to be tracked, but this process is unmanageable when multiple types of objects are expected in the scene, leading to failures when poses are missing. Passing now to the most known methods, they are based in Volumetric Fusion, being able to handle well general scenes with a relatively good accuracy, performing in real-time independently of how many objects are in the scene. In the following paragraphs, a basic description of the most known project (Izadi et al., 2011) and a high level comparison with its derivatives is made.

KinectFusion Izadi et al. (2011) was one of first systems enabling a detailed 3D reconstruction in real-time using a depth map, triggering the publication of multiple methods based on the same work-flow. The user holds a Kinect camera and move it along the scene, while the reconstruction is being made at that precise moment. As new views of the physical scene are obtained, these are merged into the model, completing and refining the model over time. To achieve this, their main pipeline consists in four steps: Depth Map Conversion, Camera Tracking, Volumetric Integration and Raycasting, which are implemented on the GPU to obtain interactive rates. The first step is

2.2. Surface Reconstruction

Methods	Point Cloud Artifacts					Input Requirements				Shape Class	Reconstruction Output
	nonuniform sampling	noise	outliers	misalignment	missing data	unoriented normals	oriented normals	scanner information	RGB image		
Surface Smoothness											
Tangent Planes	⊙	⊙								general	implicit field
RBF	⊙				⊙		✓			general	implicit field
MLS	⊙	⊙				✓				general	point set
MPU	⊙	⊙			⊙		✓			general	implicit field
Poisson	⊙	⊕	⊙	⊙	⊙		✓			general	implicit field
Graph Cut	⊙	⊙	⊙	⊙	⊙					general	volumetric segmentation
Unoriented Indicator	⊙	⊕	⊙	⊙	⊙	✓				general	implicit field
LOP	⊕	⊕	⊙	⊙						general	point set
Visibility											
VRIP	⊙	⊕			⊙			✓		general	implicit field
TVL1-VRIP	⊙	⊕	⊙	⊙	⊙			✓		general	implicit field
Signing the unsigned	⊙	⊕	⊕		⊙	✓				general	implicit field
Cone Carving	⊙	⊙			⊕		✓	✓̶		general	implicit field
Multi-Scale Scan merge	⊕	⊕			⊙			✓		general	implicit field
Volumetric Smoothness											
ROSA	⊙	⊙			⊕		✓			organic	skeleton curve
Arterial Snakes	⊙	⊙			⊕		✓			man-made	skeleton curve
Vase	⊙	⊙			⊕			✓		general	implicit field
l_1 Skeleton	⊙	⊙			⊕					organic	skeleton curve
Geometric Primitives											
Primitive Completion	⊙	⊙	⊙		⊕		✓			CAD	volumetric segmentation
Volume Primitives	⊙	⊙	⊙		⊕			✓		indoor environment	interior volume
Point Restructuring	⊙	⊙	⊙	⊙	⊙	✓		✓		general	volumetric segmentation
CCDT	⊙	⊙	⊙		⊙	✓		✓		indoor environment	volumetric segmentation
Global Regularity											
Symmetry	⊙	⊙			⊕	✓				architectural	point set
Nonlocal Consolidation	⊕	⊙	⊙		⊕	✓				architectural	point set
2D-3D Facades	⊙	⊙			⊕	✓			✓	architectural	point set
Globfit	⊕	⊙	⊙	⊕		✓				man-made	primitive relations
Data Driven											
Completion by Example	⊙	⊙			⊕		✓			general	point set
Semantic Modeling	⊙	⊙			⊕	✓			✓	indoor scene objects	deformed model
Shape Variability	⊙	⊙			⊕	✓				indoor scene objects	deformed model
Part Composition	⊙	⊙			⊕	✓			✓	man-made	deformed model parts
Interactive											
Topological Scribble	⊙	⊙			⊕		✓			general	implicit field
Smartboxes	⊕	⊙	⊙		⊕	✓				architectural	primitive shapes
O-Snap	⊙	⊙	⊙		⊕	✓				architectural	primitive shapes

Table 1: A categorization of surface reconstruction in terms of the type of priors used, the ability to handle point cloud artifacts, input requirements, shape class, and the form of the reconstruction output. Here ⊙ indicates that the method is moderately robust to a particular artifact and ⊕ indicates that the method is very robust. ✓indicates an input requirement and ✓̶ indicates optional input. Berger et al. (2014)

the conversion of the depth map coordinates into a vertex map and normal map in the coordinate space of the camera. Considering a pixel $u = (x, y)$ and the depth map value of a pixel $D_i(u)$,

a specific depth measurement can be converted to a 3D vertex in the camera's coordinate space using the intrinsic calibration matrix K.

The next step is the Camera Tracking, where a rigid 6 degrees of freedom transform is calculated to align the new obtained points with the previous frame using the Iterative Closest Point (ICP) algorithm. The first step of this process is to find correspondences between the current oriented points and the previous ones. They use projective data association for this step, consisting on the conversion of a unique point into camera space and perspective projecting it into image coordinates. To find the possible matching points, this 2D point is used to lookup into the current vertex and normal maps, finding corresponding points along the ray. These points are converted into global coordinates to be submitted to a compatibility test. If the Euclidean distance and angle between them are greater than a threshold they are considered outliers and rejected. The output of ICP is a single relative transformation matrix that minimizes the point-to-plane error metric between the actual frame and the corresponding oriented points from the previous frame. With the ICP output transform matrix $T_i = [R_i|t_i]$ any vertex and its normal can be converted into a single consistent global coordinate space:

$$v_i^g(u) = T_i v_i(u) \tag{2}$$

$$n_i^g(u) = R_i n_i(u) \tag{3}$$

These values are stored in a volumetric representation with fixed resolution, subdivided into a 3D grid of voxels where vertices are combined using a SDF, indicating the distance to the surface. Positive distances are considered in front of the surface and negative values behind. Every iteration that a point is mapped into a voxel that already has a distance value, it is updated by measuring the difference between the previous value and the calculated one, using a weighted average.

The last step is the Raycasting, which is responsible for generating the view of the surface. For each pixel a single ray is traced, voxels along the ray are tested and when a change in the sign of distance values happens (zero-crossing) between the actual and the previous voxel, the respective position is considered a point in the surface. An interpolated vertex and normal are calculated, being used to render the surface.

The majority of online reconstruction methods are based in similar work-flows. Other articles, also based on the volumetric method in Curless and Levoy (1996), improved this workflow in

order to achieve a more accurate and robust result, such as Bylow et al. (2013), or to handle larger scenarios efficiently, such as Chen et al. (2013). Zhou et al. (2013) improved significantly the reconstruction accuracy using an elastic registration method, reconstructing locally smooth scene fragments and letting these fragments deform in order to align to each other.

2.3 DEPTH SENSORS

Currently, there are multiple depth sensors available for general consumers. The most common are the RGB-D cameras such as Microsoft Kinect Cruz et al. (2012) and Asus Xtion Pro Live, which are capable of capturing a colored image and depth of each pixel in the scene. To estimate the depth value they use an Infrared (IR) projector working together with an IR camera. The IR projector creates a pattern of structured IR light which will be projected on the surface. This pattern is not visible to the human eye, but is visible to the IR camera. The camera captures the projected dots and compares to a known pattern. As the IR emitter and camera are not in the same position, the dots will end up at different image positions depending on the depth. The difference between a dot position in the IR projector and the camera is known as disparity, and the depth value can be obtained using a triangulation based in the emitter, camera and pixel positions (eq 4). In figure 19 an illustration of these concepts is presented.

$$disparity = x - x' = \frac{df}{Z} \Leftrightarrow Z = \frac{df}{disparity} \tag{4}$$

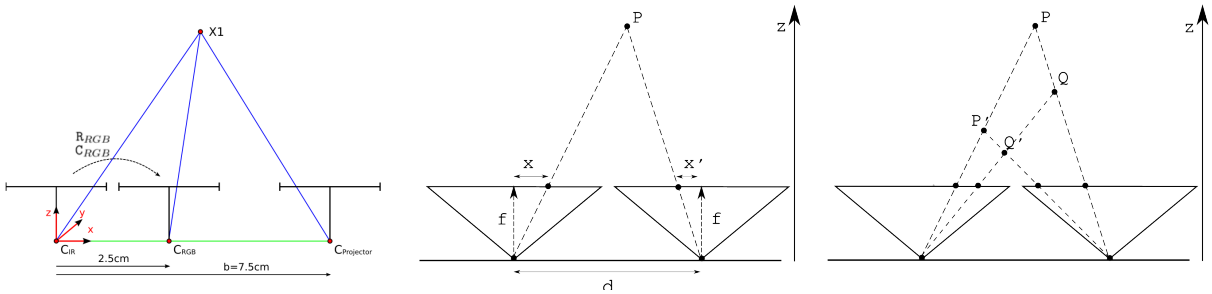


Figure 19: An illustration of the kinect geometric model (left), disparity (middle) and the depth estimation with different disparity values. Note in the right image, that for a determined pixel in the left cone (emitter), the 3D point that it intersects can be situated along the traced ray. The position is determined by triangulating using the position of the pixel that correspond to the same 3D point in the right sensor's image plane. P and P' are two possible positions along the ray traced for two positions of the correspondent pixel in the right sensor, as well as Q and Q'.

The recent Kinect for Windows 2 camera (Microsoft) also uses IR light to estimate depth distances, but instead of the structured pattern, it uses the time-of-flight principle (Lange and Seitz, 2001). This technique measures the time that a light signal takes between the camera and the surface, which based on the known speed of light, allows the calculation of the covered distance.

The following table (2) presents some differences between known devices for general consumer.

Feature	Kinect	Kinect v2	Asus Xtion Pro Live
Color Camera	640x480 @ 30fps	1920x1080 @ 30fps	1280x1024 @ 30fps
Depth Camera	320x240 @ 30fps	512x424 @ 30fps	640x480 @ 30fps 320x480 @ 60fps
Min Depth Distance(meters)	0.5	0.5	0.8
Max Depth Distance(meters)	4.5	4.5	3.5
Field of View(degrees)	H-57 V-43	H-70 V-60	H-58 V-75 D-70

Table 2: Some Depth Sensors Specifications.
(H-Horizontal, V-Vertical, D-Diagonal)

2.3.1 *Tools*

There are some tools that ease the integration of these devices in applications for various platforms, providing some base functionalities. OpenNI, OpenKinect and Microsoft Kinect for Windows are the most known, and also used as backend implementation in other libraries such as OpenCV and Point Cloud Library (PCL). Both Microsoft Kinect for Windows and OpenNI offer several capabilities such as skeleton recognition and skeleton tracking, with some different specific functionalities, but Microsoft Kinect for Windows is limited for Kinect and Windows OS, while OpenNI and OpenKinect are cross-platform frameworks, but only OpenNI supports multiple devices. This leads OpenNI to be the most used in actual multi-platform applications.

The previous frameworks are used to obtain data from sensors and have specific functionalities for user interaction applications, on the other side there is PCL, which is a more specific to the subject of 3D perception. PCL contains algorithms for filtering, segmentation, feature estimation and many others that are important to 2D/3D images and point cloud processing. One specific feature is surface reconstruction, which as seen in section 2.2 is an important theme to many applications areas, including to the subject of this dissertation.

2.4 SUMMARY

In this chapter the different areas that our system relies are exposed and explored, describing the main concepts, the problems and how to resolve some of them.

Some of the existing SAR applications can achieve similar effects to the ones that our system should achieve. The ones mentioned in the introduction of this chapter are the closest at the moment, however, they all have different purposes, not handling all the concepts in a single application.

The use of depth sensors to acquire a virtual representation of a real world scene is becoming more common, being used in the referred applications but also in others AR applications.

There are multiple types of reconstruction methods, each one with its advantages and disadvantages. The ones based in Volumetric Fusion are the most used when real-time performance is a requirement. Kinect Fusion is the most known and is used in some AR applications that also use the real scene surface. This type of methods allows the possibility to use multiple cameras to increase the information received of the scene, although, this was not done yet. RoomAlive uses multiple inter-calibrated cameras to reconstruct the entire room, but in an off-line step.

3

LIVE PROJECTION MAPPING

> "Good ideas are common – what's uncommon are people who'll work hard enough to bring them about."
>
> Ashleigh Brilliant

In this chapter the developed system is presented, starting with a description of the planned architecture, followed by the theory and implementation of the different modules of the work-flow: the projector-camera calibration, the surface reconstruction module and the projection component.

To ease the process of accomplishing the target application, the development followed a divide and conquer procedure, being divided into simpler separated applications, that when merged together should produce the wanted result. This modular development allows the use of the Separation of Concerns design principle, separating the global application into distinct sections, such that each section addresses a separate concern. From the state of the art, we can derive three main components that should be present on the application:

1. System Calibration
2. Surface Reconstruction
3. Projection Manipulation

The System Calibration is responsible for the calibration of the optical devices that will be used in the runtime execution, more precisely, the depth sensor and the projector. While the target scene will be allowed to change dynamically, the setup composed by these optical devices will be fixed. The application should start with a configuration and end with the exact same

configuration. These requisites lead to the definition of the first application of the system. It will be the first one to be executed, giving as output the calibration result that will be used in the main application. Being a process that is independent of the use of its output, makes sense to implement it as an off-line module that can be interchangeable or reused with another purposes.

The process of reconstruct the surfaces needs to perform at interactive rates allowing changes of the surface in real-time. This means that it should be running during the main application execution, updating the virtual scene as the real one changes. Ideally, this component should act as a background process communicating with the main application to retrieve the reconstructed surface, being independent and interchangeable as the calibration process. To achieve this purpose, the communication could be made with sockets, implementing a client/server procedure, allowing the computation to be made on a remote server, useful if we need a powerful computer and we do not have it on the field, and of course, with its disadvantages, such as latency.

Another option would be to implement the module as a plugin. This option allows the application to run as a normal application, not needing the procedure of configuring multiple executables running at the same time. This ease of use for anyone to start the system and reduced latency were considered priorities, bearing in mind the scale and purpose of this system.

The last step is responsible for the correct rendering of the image that should be projected. It needs the output of the calibration procedure and can't be independent of the main application as the previous two. Although the concept is the same independently of the rendering framework used, its implementation needs to be integrated with the engine used, as it modifies the normal rendering procedure through the use multiple passes. If we were dealing only with planar surfaces, an external independent process could be implemented where it should apply an homography transformation as explained before, but as it should also handle non planar surfaces, these transformations cannot be applied.

Having exposed the main components the system should have, a more detailed description of the processes and theirs interaction as a whole should be made. Figure 20 illustrates the planned architecture and its workflow, that will be described in the next paragraphs.

When using the system, the first thing that is needed for the system work is the calibration between the camera and the projector. Normally, in non dynamic systems with man made reconstructed scenes, the camera is used only to help calibrating the projector, and can be discarded after the process is done. On this system, that's not the case. The surface that the projector will be lightning can be modified in runtime, needing to update the display shape on the system immediately. This can be achieved using an RGB-D sensor (sec. 2.3). The plan here is to calibrate the projector using that sensor's feedback, and use it also to the real time reconstruction. That way,

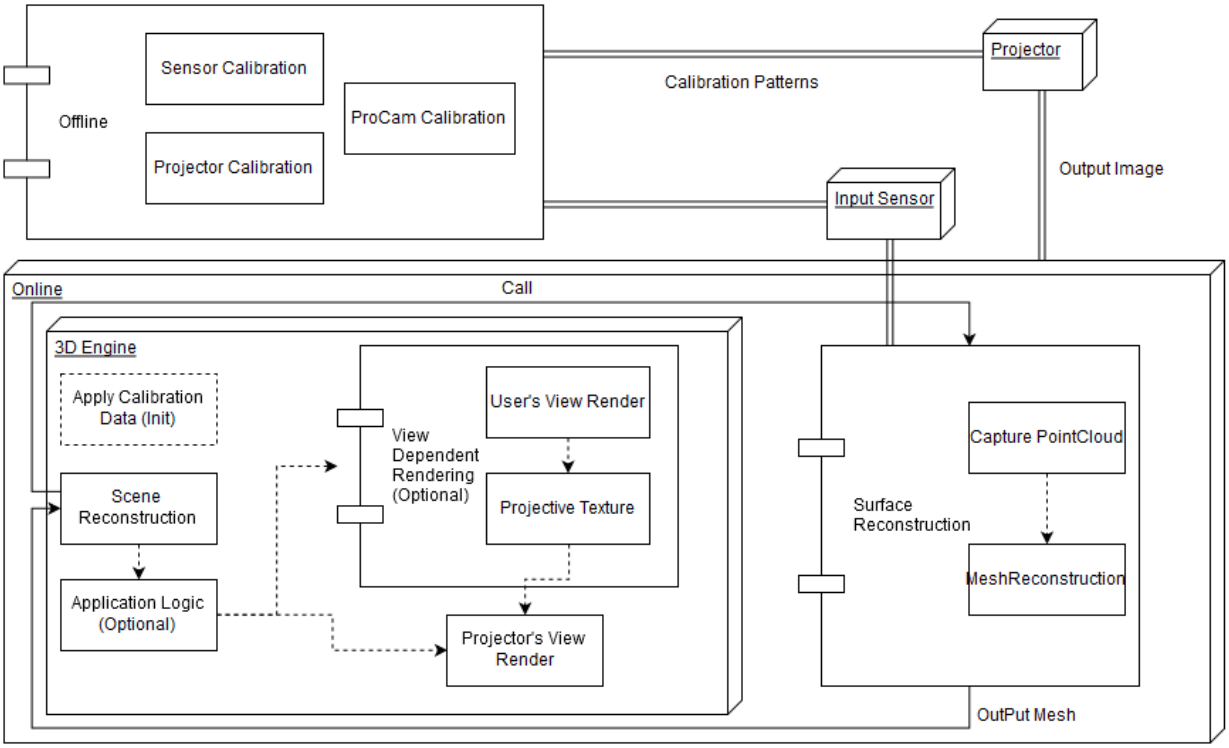


Figure 20: Architecture

we can reconstruct the surface and positioning it in the virtual world relatively to the calibrated sensor.

RGB-D sensors have two separated sensors, one to capture the visible light and another to capture IR light. The one used to measure depth distances is the IR sensor but, that sensor cannot capture the visible light that the projector emits, being impossible to directly calibrate them. Having this in mind, the calibration component will need two intermediate calibrations, one between the IR sensor and the RGB sensor, and another between the RGB sensor and the Projector. This procedure is explained in depth in section 3.1, being a process that needs to be done with the camera and projector fixed in exactly the same positions they will remain during the live execution of the system.

When the online system starts, the virtual cameras that will represent the real depth sensor and projector are positioned accordingly the calibration data obtained previously. Due to the RGB sensor being the intermediate between the depth sensor and the projector, it will be considered the center of the world's coordinate system. This way, the projector and the depth sensor are positioned relatively to it using directly the respective calibration data.

After the virtual setup is configured, the reconstruction module starts. For each frame, the depth data is obtained from the sensor and it is passed to the reconstruction module. As it receives new data from the sensor, this module will continuously update the mesh and send it to the main application executing in the 3D engine. That mesh is then positioned in the virtual world relatively to the depth sensor, representing the real surface. In the 3D engine, the virtual world logic and interactions should be handled with the reconstructed model being part of that Virtual World. The application logic is optional, representing the logic of the virtual objects, in cases that the virtual objects are elements of a game for example.

The last module will be responsible for rendering the Virtual World elements manipulated on the 3D engine, so that these objects augment the real surfaces. This process has two options of augmentation, the first is to simply change the surface appearance, and the other is to add virtual objects to the real scene. To produce the adapted image in the last case, it will be based on a multi-pass rendering based on the user and projector positions.

The planned modular procedure allows to improve the system by modifying or replace the developed modules, but also to easily add extra modules in the future, such as a user tracking module to update its position in realtime.

After having delineated the planned architecture, the next sections will describe the methods used to handle each module and why they were chosen instead of the others available.

3.1 PROJECTOR-CAMERA CALIBRATION

This section will explain the implementation of all the calibrations needed in this system, including the description of the calibration of the IR sensor explaining how to calibrate it as a normal camera, the basic description of a general stereo system relating later the two stereo calibrations needed. All of this calibrations will be made in an off-line step, thus, its result will be stored in xml files that will be loaded by the main application.

Although there are multiple variations of calibration methods either for cameras either for projectors, all of them use the same base theory described in the state of the art (section 2.1.1). They need to find the correspondence between a set of points in the image plane of the device with the respective known points in the three dimensional world. As in section 2.1.1, we will start again to describe the simpler cases, starting with the calibration of the IR sensor, which follows the process described for a normal camera, before explain the two stereo calibrations that together relate the depth sensor and the projector.

Before all, the calibration methods used in every calibration of the system are based on 2D objects with known points. As seen in 2.1.1, a 3D apparatus could be used, allowing a higher accuracy in some cases, but for the simplicity of the calibration and because the accuracy obtained with planar patterns is sufficient in most situations, a 2D pattern was chosen. The most used calibration pattern is a chessboard with know dimensions, which was the one used in all calibrations.

OpenCV provides all the tools to do a normal camera calibration using one of the most used calibration processes, the Zhang's method (Zhang, 2000). It requires that a few images of the physical pattern under different poses are taken. After this process is complete the feature points are detected in each image. By relating the detected feature points with the known physical pattern ones, the intrinsic and extrinsic parameters are estimates, as well as the coefficients of the radial distortion. This procedure is the one followed not only in the single camera calibration but also in the stereo systems as will be described later.

To calibrate the IR sensor, that process can be followed without difficulties, having just one difference from a normal camera. As explained before, the IR sensor cannot capture in the visible light frequency, so, a special attention is needed when calibrating it. Depth sensors have an IR emitter, but emit using a structured light pattern, thus, the chessboard plane will not be clean as we need. To solve this problem, that structured light emitter is blocked and another external IR emitter is required to illuminate the chessboard uniformly, allowing the IR sensor the capture it (Foundation, 2015). Another option is to do this calibration process in an outside environment

where the IR rays emitted by the sun can reach the chessboard plane. The resultant image can be used as a normal gray scale image, as demonstrated in figure 22.

3.1.1 *Stereo Calibration*

Having explained how we can calibrate the IR sensor as a regular camera, we can pass now to the explanation of the process in stereo calibrations, where two optical devices should be calibrated together. While in a single camera calibration the extrinsic parameters represent the transformation needed to pass from the camera coordinate system to the world coordinate system, in a stereo calibration the purpose is to determine the extrinsic parameters between both cameras. In this case, the rotation and translation matrices represent the transformation needed to pass from one camera's coordinate system to the coordinate system of the other camera. In figure 21 a common stereo system is illustrated.

Both cameras have their own calibration parameters that express theirs poses relatively to the same World Coordinate System (WCS). These parameters were estimated by relating the points in their Camera Coordinate System (CCS) with the points in the WCS as referred in 2.1.1. In the same way that these calibrations were made, the calibration between both cameras can be made by relating the points of the CCS of one camera to the CCS of the other. To obtain the extrinsic parameters from one camera to another, a match between the points in one CCS and the other should be made. Considering that the extrinsic parameters relatively to the world coordinates of the first and second cameras are denoted by $[R_1, T_1]$ and $[R_2, T_2]$ respectively, that match can be done by transforming the CCS of the first camera to the WCS by applying the inverted extrinsic matrix ($[R_1^t, -R_1^t T_1]$), and then transforming to the CCS of the second camera by simply applying the transformation matrix from WCS to the second camera's CCS $[R_2, T_2]$.

As explained before, a calibration between the camera and the projector is needed. Seeing the projector as an inverse of a camera, we can resolve the problem by stereo calibrating them as two regular cameras. As the sensor used to reconstruct the scene is the IR sensor, we need to calibrate it relatively to the projector but, the projector can't project in the IR light frequencies, making impossible a direct calibration because the sensor cannot capture the visible frequencies emitted by the projector. To solve this problem an intermediate sensor needs to be used. Calibrating the IR sensor with the Red-Green-Blue (RGB) one, and the RGB with the projector, we can find the relative position between the IR sensor and the projector. These two distinct calibrations will be explained in the two following subsections.

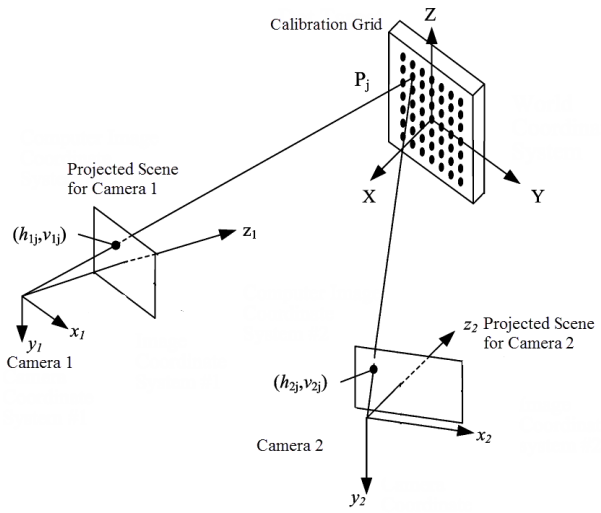


Figure 21: Grid from two cameras (Walker, 2014).

Depth - RGB Calibration

The depth information obtained from the RGB-D sensor is calculated using an IR sensor that is displaced from the RGB sensor. The objective in this calibration is to calculate the geometric transformation from the IR sensor Coordinate System to the RGB sensor Coordinate System.

To be able to find a transformation that relates both cameras, they should maintain the relative positions fixed along all the process. As in our case the camera used has both the RGB and IR sensors fixed in the same container, we can move either the camera or the pattern that lead to the same results. The capture of the chessboard images should be made on pairs, this is, for every pose of the chessboard, an image should be taken from both cameras. Although we are using different cameras, after detecting the marker points in both images, the process of calibrating between them is similar to the one explained before for a common stereo system.

As can be seen in fig.22, the resultant IR image can be handled has a normal gray scale image, allowing the detection of the corners of the chessboard in the IR image. Knowing the position of the corners, the relation between the IR image detected corners with the RGB image ones can be computed to find the relative position amongst them.

RGB - Projector Calibration

As stated before, an image from the projector's viewpoint cannot be obtained but, we can calibrate it with the help of another camera, seeing the projector as the inverse of a normal sensor.

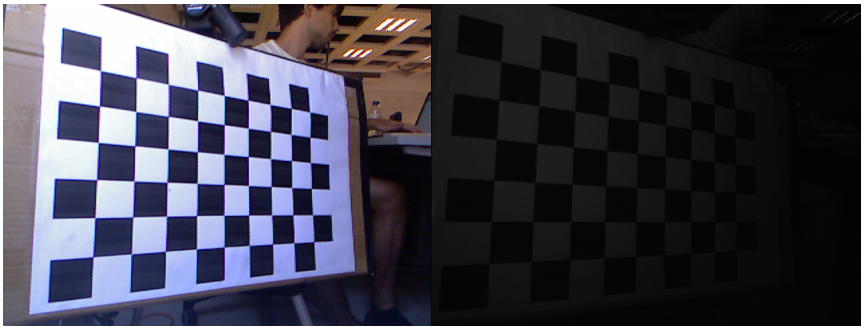


Figure 22: RGB - IR images pair

Although not having an image of its viewpoint, we know the information that it is projecting. Knowing this, the normal camera will capture the projected pattern, help finding the 3D coordinates that relates to the known projected markers in the 2D projection plane. Note that a projector follows the same mathematical model that a normal camera, contrasting only because one emits light and the other receives it. Therefore, if we have a correspondence between the projector pixels and 3D world points, any calibration technique available for a normal camera can be applied to the projector. Both methods that are described in this section have the purpose of finding correspondences between projector pixels and 3D world points. The process followed after that correspondence can be the same in both cases.

It is important to note that a new calibration is needed for each configuration of the projector-camera pair. A calibration is only correct for a certain relative position between the elements of the stereo pair. Changes on viewpoint, zoom or focus either in the camera or projector to adjust to a specific scenario also invalidates the previous calibration, even if their positions remain unchanged.

The correct calibration of the projector is essential to achieve an accurate projection over a non planar surface, hence, the more precise the method, the better results we can achieve. The first method tried was based on an already calibrated camera, using the Ray-Plane intersection (figure 23) to estimate the 3D position of the projected markers Falcao et al. (2008). Its high level process is as followed:

1. Calibrate the camera using Zhang's method
2. Recover calibration plane in camera coordinate system
3. Project a checkerboard on calibration board and detect corners
4. Apply ray-plane intersection to recover 3D position for each projected corner

5. Calibrate the projector using the correspondences between the 2D points of the image that is projected and the 3D projected points

The method was dependent of the camera calibration accuracy to achieve a good projector accuracy. Due to low resolution of the camera used, the process of recovering the 3D positions to the projected markers was error prone, leading to low global accuracies of the camera-projector pair calibration.To achieve better accuracy a time multiplexed Structured Light method Moreno and Taubin (2012) was used in the final system. In the following references to Structured Light, the time multiplexed class is the one to take in consideration.

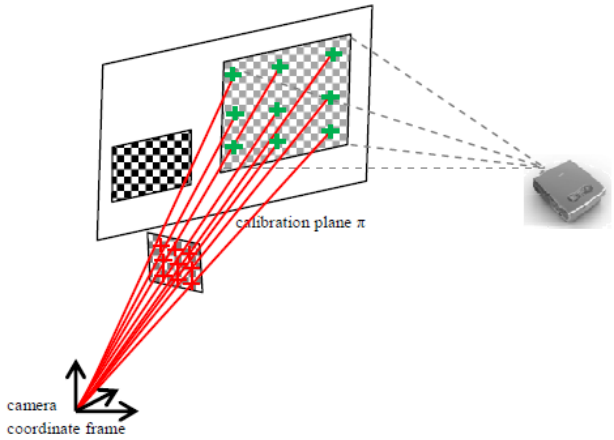


Figure 23: On the left, the printed chessboard is attached to the plane. On the right, is the projected chessboard on the same plane. Ray-plane intersection gives 3D projected points. (Falcao et al., 2008)

The structured light method was chosen over the previous one not only because it achieves better accuracy. Furthermore, it is more practical in terms of use. The first method needed a big planar surface to hold a printed chessboard and have space to project another chessboard in that same plane. The translation between the printed and the projected influences the estimation of the 3D position of the projected points. For every pose of the plane, a new estimation of the extrinsic parameters for the camera is needed, before actually calibrate the projector. Over a number of calibrations it becomes a tedious process. With a structured light method, we can use a normal chessboard plane used in a single camera calibration, being that the structured light is projected on that same pattern. Between each pose we only have to run the structured light pattern again, not depending on a previous camera calibration. Although its a long process as the first one, it is simpler as the structured light pattern projection and capture can be automated, having only to change the pose after the process.

In a structured light system, the content to project is not a single pattern as the example of the chessboard projection, but a sequence of patterns. There are multiple sequences that can be used, but the most common, and the one used in our system, is the gray pattern sequence (fig. 25), as they are robust to decoding errors. As in a normal camera calibration, a known dimension pattern is needed to relate pixel positions with known 3D positions. The pattern used was again the printed chessboard. While in other calibration methods a single image is taken for each pose of the physical pattern, in a structured light system the entire sequence must be captured with the target pattern fixed, hence, the process of projecting and capturing each element of the sequence was designed to be automated. The sequence of images is generated dependently of the resolution of the projector, being then projected and captured one by one with fixed time intervals. After the entire sequence is captured, the physical pattern can be moved to another pose and start the projection of the sequence again.

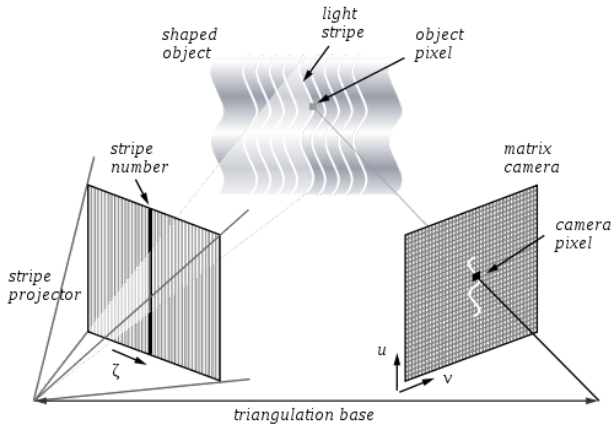


Figure 24: Structured Light Diagram - The projector displays a sequence of striped pattern to identify the relation between projector and camera pixels. (thefullwiki, 2009)

For each pose, the captured sequence of images will be used to find the correspondences between the camera and the projector pixels (figure 24). The process to find the chessboard corners coordinates in the projector image is divided in three steps. In the first one, the structured-light sequence is decoded, associating every pixel of the camera to a projector row and column, or set to "uncertain" if not found a match. In the second step, a local homography is estimated for each checkerboard corner in the camera image. This step is a contribution of Moreno and Taubin (2012), allowing better accuracy than with the global homography. The last step uses the local homography found to convert each of the corners from camera coordinates to projector coordinates.

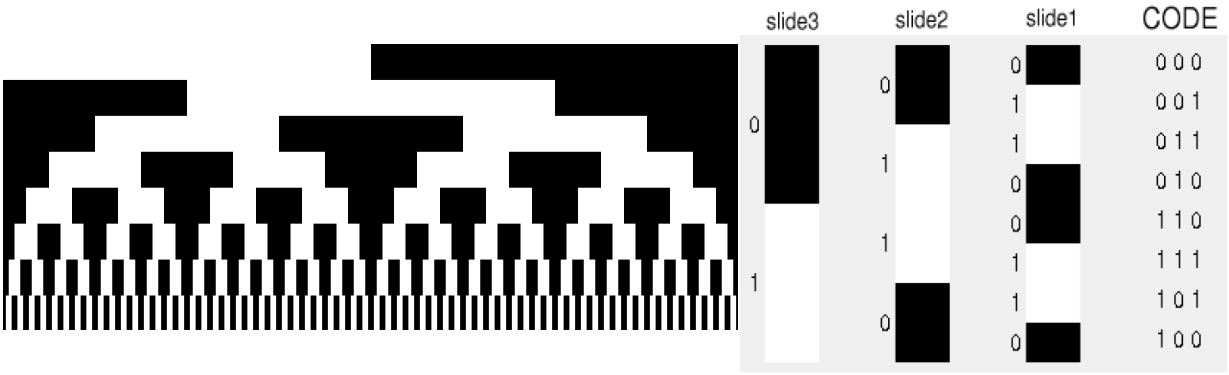


Figure 25: In the left image a 7-bit horizontal Gray-code sequence is illustrated and in the right image, the codewords for a sample sequence with three bits identifying the possible cases. By projecting a sequence with enough binary coded stripe patterns for the resolution of the projector, the subsequent light levels for each scene pixel are forming a unique binary word revealing the individual stripe number.

The decoding step depends on the projected pattern. As the patterns used are the complementary gray codes both for horizontal and vertical matches, this step needs to accurately classify pixels located within the black-and-white stripe. In scenes containing complex surface-light interactions such as strong indirect lightning, achieving a robust classification is difficult. The use of complimentary patterns and the estimation of the direct and indirect components of a pixel intensity allows a more precise classification for this cases.

For a pixel directly illuminated by the projector, the direct component d of its intensity p represents the direct light from the projector, being invariant under different illumination patterns. Contrastingly, the indirect or global component i resultant of other light sources including reflections from other projector pixels, depends on multiple factor such as the bidirectional reflection distribution function (BRDF) at the scene point and the set of the surface patches that are lit.

Using a subset of the gray code images that are high frequency patterns (the ones with thinner stripes), the intensity measured in each pixel can be splitted in the direct and global components.

After each component is estimated, two intervals of pixel intensities are determined from a set of rules, one for pixels that are directly illuminated and the other for pixels that are not directly illuminated. If a pixel intensity p is within one interval but not in the other, the pixel belongs to that category, otherwise, it is labeled as uncertain. The use of complimentary patterns improve robustness as a pixel is only classified as illuminated or not only when a pixel has consistent results for both inverted patterns, as complimentary patterns are the inverse of the other. For more details of the pixel classification method refer to Xu and Aliaga (2007).

After the classification of every pixel, the association of each pixel with a projector's row and column is direct. For each pattern, a pixel directly illuminated is marked as On (1), the ones not illuminated are marked as off (0). Concatenating the bits from all the binary classification images we obtain an identifier of the row and column as each one of them is represented by a specific sequence of bits. The pixels with uncertain bits are not associated with a projector pixel. Figure 26 illustrates this association through different color attribution to each row and column.

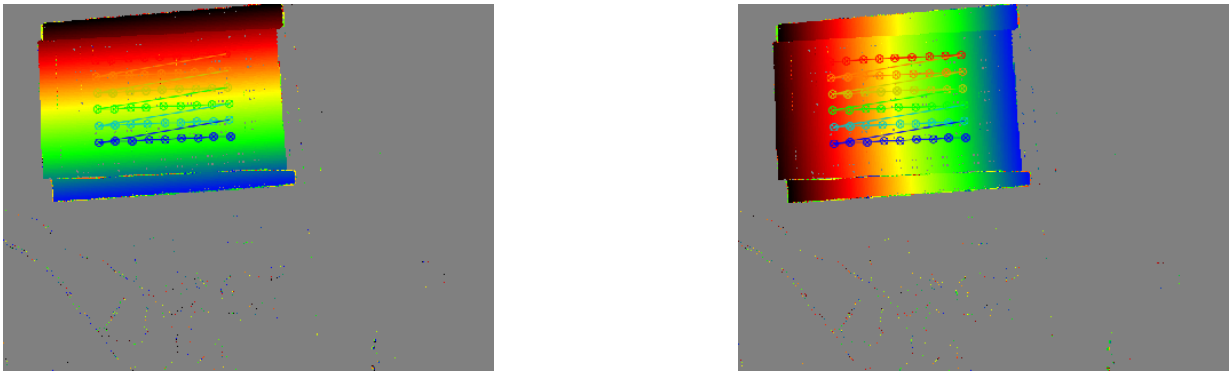


Figure 26: In the left image the pixels with the same color correspond to the same row of the projector, while in the right image correspond to the same column.

The map created between projector and camera pixels is not bijective as many camera pixels can correspond to the same projector pixel. To resolve this problem and the fact that chessboard corners are not located at integer pixel locations, Moreno and Taubin (2012) proposed to compute a local homography for each corner. Due to radial distortion being significant in a complete image, a single global homography is not enough, but that radial distortion is negligible in a small neighborhood. For each corner, a local homography is found taking into consideration all the correctly decoded points in a patch of the camera image centered at the corner location.

Each corner in the camera image is then translated to the projector coordinates by applying the respective local homography.

At this point, the location of all corners in the projector coordinate system is known, allowing its calibration following the same procedure described earlier for a normal camera.

3.2 RECONSTRUCTION

In this section, the surface reconstruction module is explained. This module should generate a mesh identical to the surface that the camera is pointing at, with the purpose to insert it in a virtual world.

As exposed in the state of the art (2.2), there are multiple types of surface reconstruction methods, ones faster, others with higher accuracy, ones that deal better with specific types of objects, others that try to deal with them all, ones completely autonomous and others that need user interaction, and so on.

The proposed system needs to perform in real time, being capable of allowing interactivity with it. Having said that, the chosen method needs to be fast. The faster, the better. A method that needs a second to update the scene will break this interactivity. The less delay the method has, the faster and smoother will be the interaction with the system. Some methods that have been described in the first part of this document can achieve a relatively good performance, allowing this interactivity, but, the majority of them doesn't meet this requirement. It also should be noted that some methods require a very dense and noiseless point sets to work well, and our system will scan the surface with a RGB-D sensor, which cannot capture small details and may have significant noise.

Almost all known methods in real-time are based on KinectFusion (Izadi et al., 2011), which also use a RGB-D sensor (Microsoft Kinect) as our system, handling well its data. Baak et al. (2013) is a rare example that achieves real time reconstruction and is not based on volumetric fusion. This kind of methods can be a good choice too for some specific cases. In this particular one, the target to reconstruct is a human body, but a more general data-driven method that handles all kinds of objects could work very well, with its downsides. In a complex scene with multiple objects, where the type and poses of objects that can enter in the system are countless, a large data base is needed. In complex scenes, the speed of the method would also decrease, leading to restrict the possible scenarios. The goal of this system is to interact with all kind of scenes, from a simple wall, to deformable surfaces or multiple moving bodies. The Kinect Fusion procedure can handle general surfaces like that better than a data-driven method, being the one we based the implementation in this system.

There are many differences between our system and the original Kinect Fusion process, one is that the Kinect Fusion was designed to reconstruct a complete scene or object by moving the sensor, while merging the new captured information with the one already stored. To be able of a coherent fusion of data, the camera pose must be estimated each frame relatively to previous

frame. In our system the camera is fixed during all the execution, not needing this pose estimation. Another important difference between the systems is that Kinect Fusion assumes that the scene is fixed, trying to ignore rapid or momentary alterations. In our system, interactivity with the scenario is one of the goals, so, the scene can be changed in runtime, needing a fast update of the reconstructed surfaces. To achieve this faster update, the old data will be averaged with the new data just to minimize noise and artifacts, but with a higher weight for the new input to achieve a faster renovation of data. Even though the sensor will be fixed along all the execution, using a method with volumetric fusion opens the possibility to use multiple sensors, augmenting the range of the scene that is captured and its accuracy. As in the original Kinect Fusion the fusion of the data uses the new camera's pose, the procedure could be changed to merge the data of N sensors from N poses. This implies that we have estimated the poses of all sensors in the off-line calibration. The last difference is the rendering part. In Kinect Fusion the volume is raycast (Direct Volume Rendering (DVR)) to extract views of the implicit surface, not producing an object usable outside of the Graphics Processing Unit (GPU). In our system, this reconstruction process will be just an auxiliar component that should return a mesh, allowing it's import in the virtual world. For this, the last step should generate a mesh of the surface. Instead of a DVR method, the Marching Cubes (Lorensen and Cline, 1987) method is used as the last step. The workflow of this process is illustrated in figure 27.

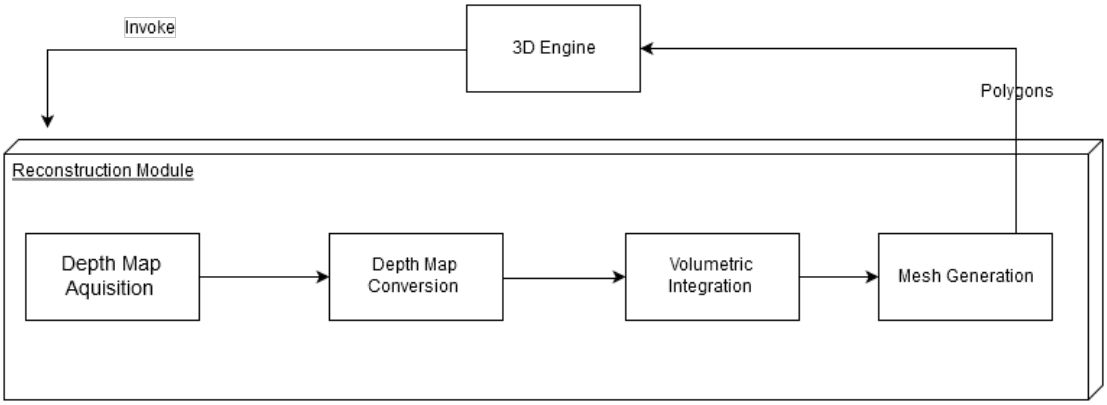


Figure 27: Reconstruction Module Schema.

3.2.1 *Implementation*

The implementation of this module was based on an freely available BSD licensed implementation of the Kinect Fusion (Izadi et al., 2011) by the Point Cloud Library (Rusu and Cousins, 2011), but with the respective changes that our system needs.

As said earlier, this module will be a secondary component that should communicate with the main engine to retrieve the reconstructed surface, being implemented as a plugin (dll).

Data Handling

The data received from the sensor is not a group of points but a 2D image where its pixels values represent a depth value. First, these pixels should be converted into 3D points. The process is responsible for the the transformation of the depth map into a point cloud. This conversion to 3D points is based on the pinhole camera model (fig. 5) as the calibration module, needing the intrinsic parameters of the IR sensor obtained in the calibration stage. Using the focal length (fx,fy) and principal point (cx,cy) of the sensor, the values of x and y axis of the vertex in the world can be calculated based in the pixel's coordinates in the 2D image (u,v) and its depth value (z).

$$\begin{aligned} v_x &= z * (u - cx) * fx^{-1} \\ v_y &= z * (v - cy) * fy^{-1} \\ v_z &= z \end{aligned} \tag{5}$$

In practice, this process is made inside the Volumetric Integration component, sending to it the depth map already scaled (meters), and converting the pixels to points only at the time of merging them with the actual Truncated Signed Distance Function (TSDF) volume, see below.

Volumetric Integration

This component of the work-flow is the key to handle the input data efficiently while allowing good reconstruction quality. Instead of fusion the new point cloud with the old ones, or generate the mesh directly from the actual point cloud, which would limit the system, a volumetric surface representation (Curless and Levoy, 1996) is used to handle the data (fig. 28). This structure is called a Truncated Signed Distance Function (TSDF), and allows the fusion of new data as it is received, acting as a unified global coordinate space. This process discretises this global space

into a regular 3D grid of voxels. Each voxel stores the distance to the nearest surface, being the surface itself at the distance zero. Voxels that lie between the sensor and the surface stores positive values and the ones behind the surface, stores negative values. The surface is easily found by detecting a change of sign (zero-crossing). These voxels will be updated every time that new data is received, based on a signed distance function specifying a relative distance to the actual surface.

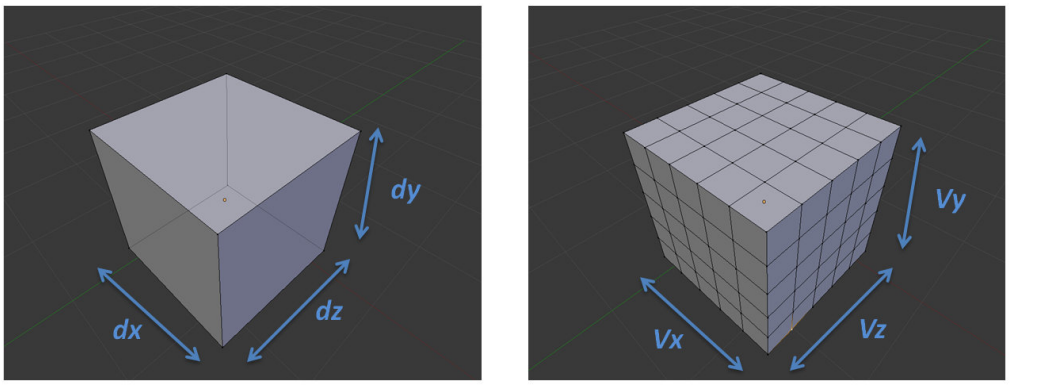


Figure 28: The cube is subdivided into a set of Voxels. These voxels are equal in size. Both the number of voxels and the size in meters give the amount of detail of our model. (PointCloudLibrary)

Knowing the camera pose relative to this unified global world, a ray is emitted from the camera through each captured vertex. For all the voxels in the grid that are intersected by the ray, the relative distance from the voxel center to the vertex is computed. This is made calculating both distances from the vertex to the sensor's plane and the distance from a determined voxel to the sensor's plane too. The relative value of the SDF of a particular voxel is then obtained by the difference between these two distances. This difference will result in positive values for voxels that are in front of the surface, a zero value if they are at the same distance, this is, the voxel is intersected by the surface, and a negative value if the voxel lies behind a surface. As the region of interest is the one closer to the surface, we do not need to compute and store the distance for every voxel. A truncation of the SDF can be made, ignoring voxels that are behind the surface. The resultant values of these calculations are averaged with the previous values, using a running weighted average. Both the new weight and averaged TSDF are stored at the voxel. Figure 29 illustrates the TSDF values in a specific example by coloring the pixels accordingly to the distance values to the surface.

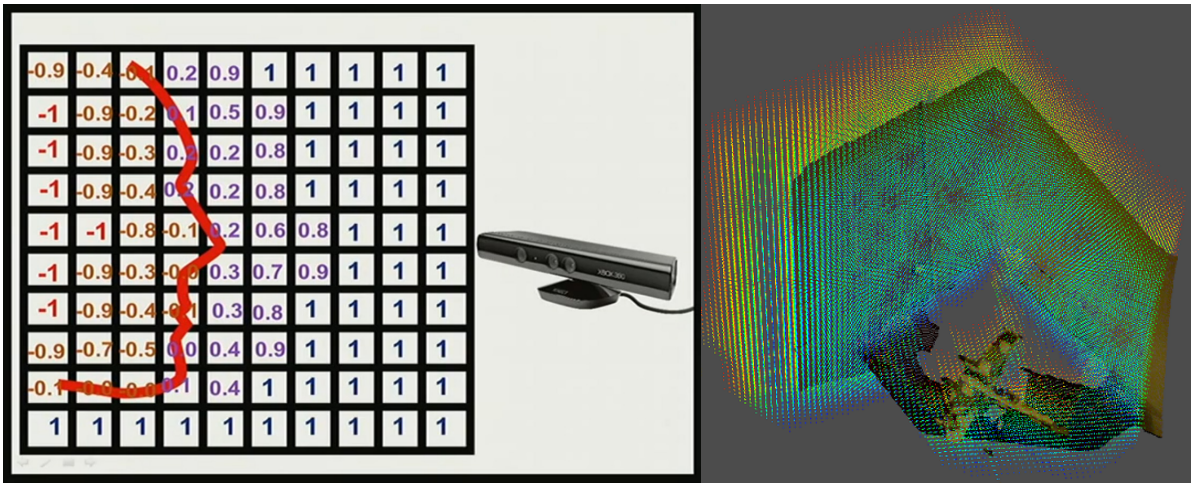


Figure 29: In the left image a 2D TSDF is exemplified, being the red line the surface. In the right image (Morgan, 2014) a real-world example of a 3D TSDF. Blue points represent positive values of distance, red points represent the voxels behind the surface, having a negative value. The actual surface, the zero-crossing is represented as green.

All the procedure described is highly parallelized on the GPU, running very efficiently. The full 3D voxel grid is allocated on the GPU as aligned linear memory, allowing simultaneous access from multiple parallel threads. The downside of using this structure is that it is not memory efficient, as it will always need N^3 voxels, being N the resolution of the grid. The compensation is the efficiency in computation time.

Mesh Generation

As explained before, the last step of the reconstruction module is responsible for generating the polygonal mesh that will be imported in a graphics engine. To obtain that mesh, the rayscast step of Kinect Fusion is replaced with a polygonal mesh generation method, the marching cubes. Using an Indirect Volume Rendering (IVR) method like Marching Cubes generally implies some resources consumption disadvantages against DVR methods, such as more memory consumption. However, performance wise it allows interactive frame rates, as the more recent DVR methods that use acceleration techniques (Bartz and Meißner, 2000).

Marching cubes is an algorithm that extracts an isosurface from a volumetric data set like the one constructed in the Volumetric Integration step. It process the cubes that makes the volume. Each voxel is considered as a corner vertex of the virtual cubes used to generate the polygons. For each cube, the vertices's values are evaluated. If a vertex has a value equal or above the isovalue it is marked, the others are left unmarked. The isovalue is the value that represent where

Algorithm 1 Volumetric Integration Pseudo Code

procedure TSDF
  for each voxel g in x,y volume slice **in parallel do**
    while sweeping from front slice to back **do**
      $v^g \leftarrow$ convert g from grid to global 3D position.
      v $\leftarrow T_i^{-1} v^g$.
      p $\leftarrow$ perspective project vertex v
      if v in camera view frustum **then**
        $\text{sdf}_i \leftarrow ||t_i - v^g|| - D_i(p)$
        if $\text{sdf}_i > 0$ **then**
          $\text{tsdf}_i \leftarrow min(1, \text{sdf}_i/\text{max truncation})$
        else
          $\text{tsdf}_i \leftarrow max(-1, \text{sdf}_i/\text{min truncation})$
        $\text{w}_i \leftarrow \min(\text{max weight}, \text{w}_{i-1} + 1)$
        $\text{tsdf}^{avg} \leftarrow \frac{\text{tsdf}_{i-1}\text{w}_{i-1} + \text{tsdf}_i \text{w}_i}{\text{w}_{i-1} + \text{w}_i}$
        store w_i and $\text{tsdf}^a vg$ at voxel g

the isosurface is intersected. In majority of the cases the isovalue is zero. As described in the previous sub section, in the used structure, a positive value represent a distance to the surface of a point that is between the surface and the camera, and the negative ones represent points behind the surface. The zero-crossing areas, is where the surface that we want to construct a polygonal mesh is located.

The marked vertices will define the edges intersected by the surface. Every edge that has a marked vertex (inside) and the other unmarked, will be intersected by the surface. This process gives us the active cubes. As each of the eight vertices can be marked or unmarked, there are 2^8 (256) possible cases. By enumerating these cases a look-up table can be created, corresponding each case to a cube-isosurface intersection pattern. In figure 30 the 15 base intersection cases are illustrated.

The exact position where each vertex is positioned in the respective edge is estimated by interpolation. The vertex should be placed where the scalar value is equal to the isovalue, zero in this implementation. Having two vertices sharing an intersected edge, one of them would be positive and the other negative. With linear interpolation the zero-crossing position can be easily estimated.

In figure 31, a single case is illustrated. The index to search in the lookup table is generated following the vertex order and their state, being formed by eight bits, one for each vertex. In this

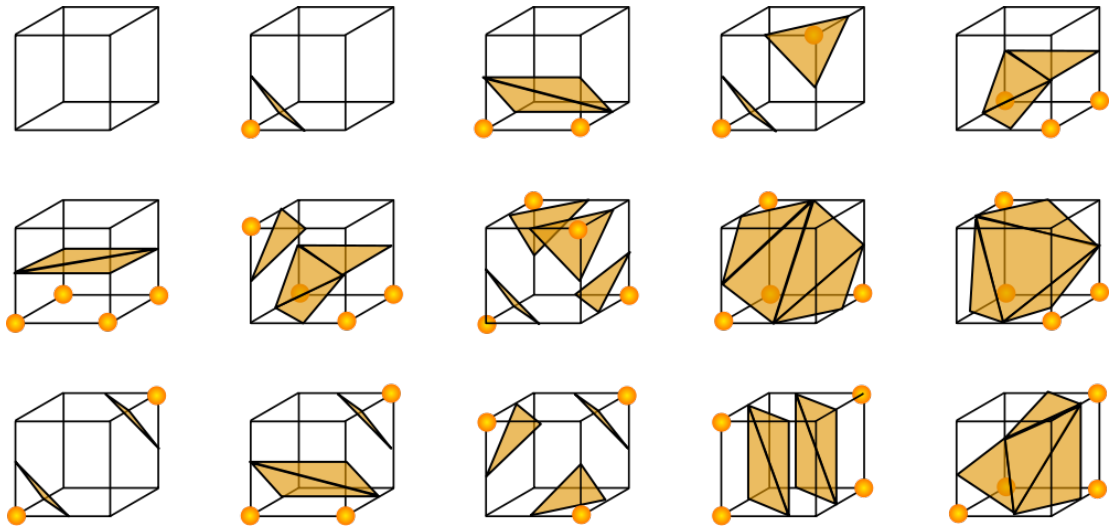


Figure 30: The 15 basic intersection topologies considered in the Standard Marching Cubes exploiting reflection and rotation symmetry. (wikipedia, 2010)

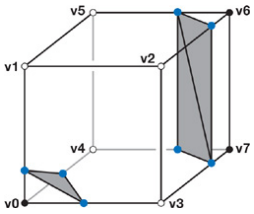


Figure 31: Implicit Surface to Polygon Conversion. (nVidia, a)

case the index should be:

index = $\boxed{\text{v7|v6|v5|v4|v3|v2|v1|v0}}$ = $\boxed{\text{1|1|0|0|0|0|0|1}}$ = 11000001 = 193

The obtained index should be used to find the edges configuration in the look-up table. In the figure, these edges are marked with the blue dots. The gray areas illustrate how vertices will be connected to form the triangles. The blue dots' correct positions are then estimated by interpolation. The cases correspondent to 0 and 255 indexes, will be the cases where all vertices are outside or inside the surface, not causing intersections on edges. This cases would not generate any triangle as the cube is completely outside or inside the surface. In all the other cases, the cube will be on the surface, generating between one to four triangles depending of how many edges the surface intersects.

The list of vertices resultant of marching all the cubes is the final output of the module. The array of vertices that will be returned generates the mesh by form a triangle each tree sequential vertices.

In this module the normals of the surface are not calculated, being estimated only on the main module if they are required in a specific example.

The procedure of this method to generate the triangles can be resumed as following:

1. Read Data Into Memory and generate the virtual cubes
2. Classify the eight vertices of each cube
3. Generate the index number
4. Use Index to Look Up List of Edges
5. Estimate correct position for the edge intersection using Linear Interpolation
6. Output the Triangle Vertices

3.3 PROJECTION

The generation of correct images to project on a surface is the main goal of this system. This section is responsible for the explanation of the process that allows the achievement of that goal.

As exposed in 2.1, unlike the previous modules, where multiple methods could be chosen, the options available weren't much. Or we limit the target surfaces to planar ones, to use a simple homography method, or if we need to project on non-planar surfaces, we have to create a replica of the real world in the graphics engine. The two previous sections described the implementation of two crucial modules, where the output of them are used to create this replica of the real world.

Before explain the actual implementation of the projection related procedure, we start by describing how that replica is created, using the modules previously described.

To create a replica of the world, the position of the depth sensor and the projector must be known, to position its virtual representations with the same poses as the ones in the real world. This information is obtained in the calibration module. Although this calibration process is made in an offline module, needing to be made before starting the main application, it should be done with the sensor-projector pair in the exact relative positions that will be used in the runtime execution. If the calibration is executed with them in a certain configuration but that configuration is changed afterwards, the virtual world won't represent the real one anymore. Each time the configuration changes, a new calibration should be made.

When the application starts, the virtual camera and projector are then positioned accordingly to the parameters obtained from the calibration process, starting then the execution of the second module. Both camera and projector are positioned in the virtual world as the real ones. To obtain a complete replica, the real surfaces should also be represented in the virtual world. The reconstruction module is invoked from the main application and starts running in parallel, using the sensor to obtain data from the scene. With that data, it generates a polygonal mesh and sends that mesh to the main application. In the main application the mesh is positioned in the virtual world relatively to the virtual camera that represents the depth sensor, as the generated mesh was reconstructed using the data obtained from the real depth sensor.

The world replicated in the graphics engine after this procedure is ready to apply appearance modifications to surfaces, as the famous Shader Lamps (Raskar et al., 2001), although, to be able to have virtual objects at different depth from the surface and give a correct perception to the user, another step is needed. This module is responsible for handling these cases.

We start by remembering that when a virtual object is distant from the surface a certain distance, the image that a user needs to see when it is at the left side of the system is different from

the image that he should see when seeing it from the right side. The projection correspondent of the virtual object should adapt consonantly the position of the viewer. This concept of rendering using the position of the user is known as View Dependent Rendering (VDR). In projection systems, it allows a similar effect to holograms, as it adapts the rendering to our eyes' position, giving a 3D perception when the user moves. When entering in this subject on SAR systems, a multi-pass renderer is used.

Before explaining the next step, is better to understand the differences between a VDR and a normal rendering procedure.

Considering a First Person View application, there's a virtual camera representing the virtual user position. Remember that the virtual camera is representing the virtual user position and not the user position in the real world. The resultant rendering from that camera is the image that we see on the screen, that is unaffected by our position in the real world. The physical display is the element of the real world that is inside the perspective of the virtual user in the virtual world.

When we are dealing with VDR, the final rendering should take into consideration the user position in the real world. In our system, to add the user position in the virtual world, we should estimate the relative position of the user to the projector device or to the sensor, as they are already represented in the virtual world. This way, another virtual camera can be added to the virtual world, that will represent the user viewpoint. In this system, the user position is simulated, and the head tracking module was left for future work.

Besides the need to know the user's head position to have a virtual camera representing its viewpoint, it is also needed a correspondence between the pixels from the camera that is representing the viewpoint of the user, to the viewpoint of another camera that is representing the projector. Once again, to deal with non planar surfaces we cannot rely on homographies to obtain the transformed image. As exposed in section 2.1, the state of the art to resolve this problem is to implement a projective texture system, which will be explained in the next subsection. After the description of the projective texture, the explanation of the actual process to achieve a VDR is described.

3.3.1 *Projective Texture*

Projective Texture is a texture mapping method that allows a 2D textured image to be projected on a 3D surface as if by a slide projector (fig. 32). It is one of the critical concepts that are used to perform advanced rendering techniques for real time applications, being useful in multiple techniques such as soft shadows, projective light maps and reflections.

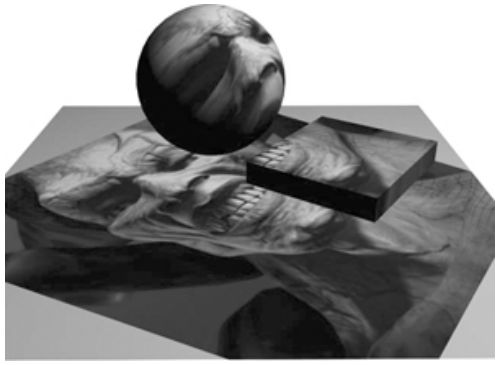


Figure 32: Projective Texturing. (nVidia, b)

The texture used in a projective texture method is a normal texture as the ones applied directly on the material of an object. The difference is in the way that it is applied to the geometry. It works in a similar way to the one used in a normal camera rendering a 3D scene onto the 2D image that we see on the screen. In a normal camera, for each triangle a series of transformations are applied from the object space to a sequence of coordinate systems, projecting the vertices from the initial object-space to the final window space that we see. In a virtual projector, the same basic properties can be applied. Like a normal camera, it has a viewing transform that transforms world space coordinates into projector space, and a projection transform that maps the projector space view volume to clip coordinates. By applying these transformations the object-space can be mapped to a 2D space. As we want to project a 2D texture, we use the coordinates obtained from the transformations to map each vertex to that texture. Assigning that texture coordinates for the respective vertex the correct piece of the texture can be applied to each triangle when rendered.

The actual sequence of transformations needed is illustrated in figures 33 and 34, being M the model matrix, responsible for the modeling transformation to the vertices, V_p the view matrix for the projector, applied to transform vertices to the projector frame of reference, and P_p the the projector's projection matrix, that defines its frustum, field of view and aspect ratio. The last matrix of the transformation is responsible for the range mapping, as the transformed vertices values range from -1 to 1 and texture coordinates are indexed in the [0,1] range.

There are some issues with this technique that should be considered. The first, is that it produces a back-projection, resulting that the texture projected is not only projected on the surfaces in front of the projector, but also in the ones behind it. This happens because the texture coordinate is computed independently that a vertex is in front or behind of the projector. There are some solutions to this problem, but the simplest and efficient is to use the fragment shader to verify

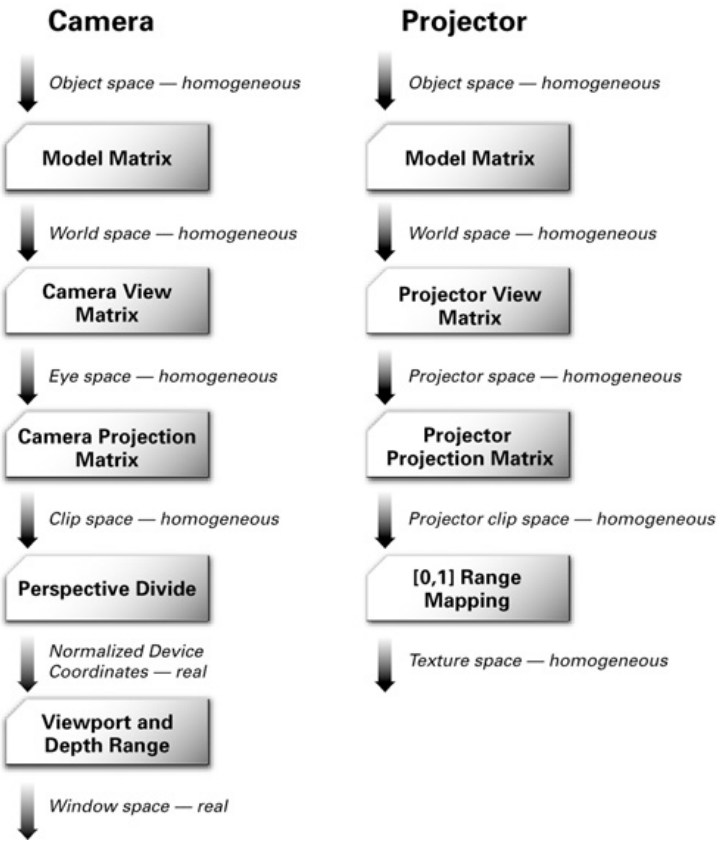


Figure 33: Transformations for a Conventional Camera vs. Those for a Projector. (nVidia, b)

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} P_p V_p M$$

Figure 34: Sequence of Transformations for Projective Texturing.

this situation, and ignore the projective texture computation in that cases. The second problem is due to the two frustum that can be in bad orientations relatively to each other, resulting in zones of a single texture that needs extreme magnification and others that needs extreme minifications, in a single pass. The last problem is that there are no occlusion checks, subsequent the projection will be applied to all the triangles that are in the view frustum. The GPU transforms each triangle independently of the others, not having the knowledge of occlusions or intersections. This breaks the expected result of triangles shadowing another. In our system, as the image rendered will be dependent of the user's position, the artifacts projected behind the visible surfaces to the

user won't be visible to him, as it only sees the first surface, but if another user is watching in another position, it can happen to observe that artifacts from its viewpoint. As expected, a VDR is dependent of a view pose, being difficult to simulate a correct effect to multiple users at the same time. In the majority of the cases, it is even impossible. This occlusion problem in the projective texture technique is another that hampers the handling of multiple users in a complex 3D scene.

3.3.2 *View Dependent Rendering*

Having mentioned all the components that the process needs, we can now explain the complete procedure of the View Dependent Rendering. The actual procedure implemented on the system is as illustrated in figure 13, needing three steps. In the first step of the rendering process, the three-dimensional virtual objects are rendered from the camera representing the user's viewpoint. This way, the virtual points are mapped to pixels in the view of the user. The image resultant of this process represent how the virtual objects should be seen from the user's perspective. This image is not sent to the projector as in a normal rendering, but stored in a texture map to be used in a next step. Using the Projective Texture method from the viewpoint of the user, that texture is projected onto the virtual surface that represents the real-world scenario. The projective texture projects over the surfaces in a similar way that real projectors work, adapting the texture to the different surfaces accordingly to its position. After this process, the virtual surfaces representing the real-world display, have now the user's desired image texture mapped on them. The last step is the rendering from the camera that is representing the projector viewpoint. When the rendered image in this step is projected onto the real surface, the user will have a correct perspective view of the virtual object. In figure 35, the steps of the entire process are represented, as well as the final projection and the image from the viewpoint of the user. As we can see, virtual objects that are behind the real surfaces can also be adapted to the surface occluding it.

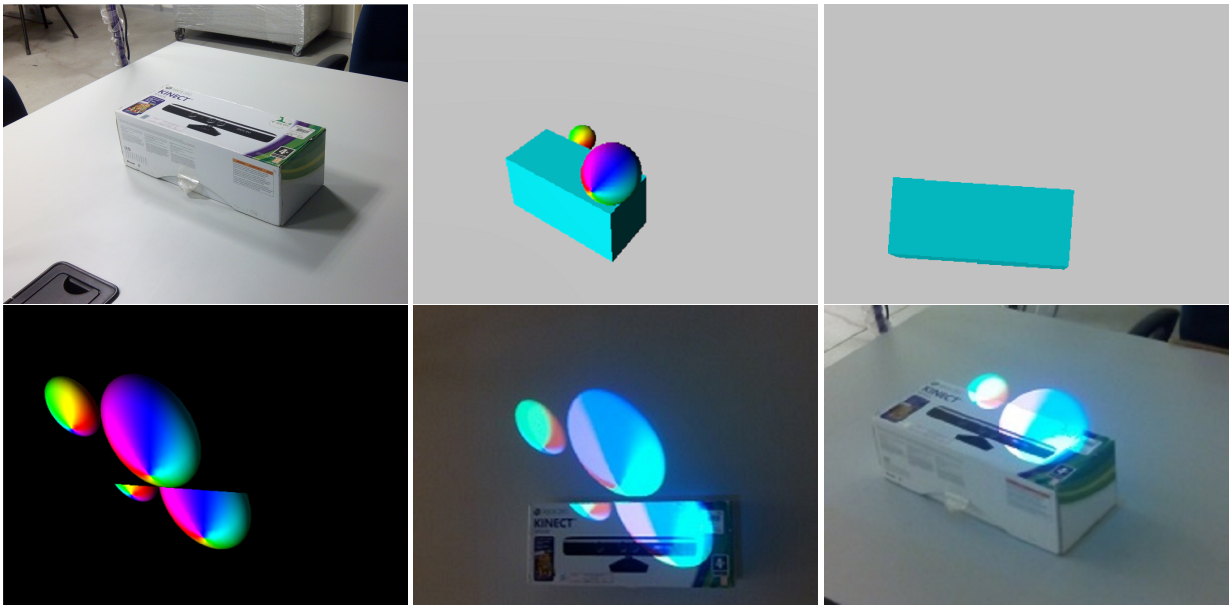


Figure 35: Illustration of the multi-pass rendering process, where the white floor and the blue cube are representing real world surfaces. The ray-bow spheres are virtual objects that the user should see projected on the real surfaces. The top-left image is taken from the viewpoint of the user in real world, the top-center image is also taken from his viewpoint but in the virtual world, having already the virtual objects. The top-right image represent the viewpoint of the projector before the projective texture step. The bottom-left image is the resultant image that the camera representing the projector sees after the projective texture have been applied. The projected texture includes only the virtual objects, not considering the ones representing the real world. The bottom-center image is illustrates that texture projected on the real world. The last image is again from the user's viewpoint representing the final result.

4

RESULTS

In this chapter, the results of the developed components are exposed. As the different components were implemented separately, their separate analysis can be made, helping to isolate possible problems either in performance or quality. For some components a quantitative analysis of the result in terms of quality was hard to calculate, analyzing just the performance or making a qualitative evaluation. To contextualize both the performance and quality results, we start by the description of the hardware and tools used, followed then by the analysis of the modules separately and finishing with the evaluation of the system as a whole.

4.1 SETUP AND TOOLS

The performance results exposed in this section were obtained in a system with an Intel Core i7-2630QM CPU, 4GB of RAM and a nVidia GT635M. The RGB-D sensor used was the Microsoft Kinect for Xbox 360 with a PC adapter, using the OpenNI2 as driver in modules that use its input. The projector used was a Projection Design F1$^+$ XGA, having used the resolution of 1024x765 in all demos described in this chapter.

The calibration module was implemented in C++ as an external application with the help of OpenCV, and QT. The reconstruction module was implemented as a plugin in C++ using OpenCV, PCL and its dependencies. The main application was implemented in the Unity3D game engine.

4.2 CALIBRATIONS

As described in section 4.1, the sensor used in our setup was a Microsoft Kinect and to calibrate its depth sensor with the projector, two intermediate calibrations are made through its RGB sensor, one between the depth sensor and the RGB, and another between the RGB sensor and the

projector. The RGB sensor has a significant lower resolution (640x480) than the projector used (1024x768), leading to some restrictions in the setup mounting. In the CSL decoding process, the projector image should be mapped to the camera image which, in our setup, has a lower resolution than the projector, leading to fail in classifying the pixels in the high frequency patterns of the CSL.

To achieve good accuracy on the decoding of the CSL with our hardware, our tests showed that the sensor should be located at a minimum distance of around 40 cm in front of the projector, minimizing the problem of using a low resolution sensor.

If only the projection target were placed closer to the sensor-projector pair with the purpose to fill the camera image, the results would be poor in the same order, as by approaching the projection target, the projection itself becomes smaller too. The use of a bigger physical pattern only helps to find accurately its known markers, not helping in the CSL decoding process. Relatively to the chessboard corners detection, the pattern was printed in a A2 paper. Because of the low resolution of the camera and that the pattern needs to be at a certain distance of the projector, using smaller ones resulted in bad detection of the corners. In figure 36 two examples of the setup configurations used are illustrated, demonstrating the relative positions used between the sensor and projector.

Figure 36: Two examples of positions between the sensor and the projector. In the left they are placed in a table to project against the wall. In the right they are placed in the ceiling for top-down projection.

The first validation of the calibration process is made by the re-projection error value. This value represents the distance in pixels between a pattern keypoint detected in a calibration image, and a corresponding world point projected into the same image by applying the parameters calculated. The better the calibration, the lower this value. Ideally it should be near 0, but acceptable below 1.

With the setup described, the re-projection error of the stereo system between the RGB sensor and the projector a value of 0.8 was reached . With the sensor on top of the projector, this value was around 2.

The re-projection error in the stereo calibration of the RGB-IR pair achieved a value of 0.4.

In the case of the RGB sensor - projector calibration, the validation can be performed visually in two ways. As explained in the previous chapters, the calibration needs the capture of multiple poses of the physical known pattern. To validate that each of these poses is good, an homography can be used. For each pose captured, the homography for that pose is calculated, allowing the re-projection of the chessboard pattern captured by the camera. If the calibration procedure succeed for that pose, the projection of the chessboard will superimpose the physical one. If the calibration result to a specific pose is bad, i.e., it has a high re-projection error, it won't match the physical one, indicating that it should be excluded.

The other verification method is by the use of the epipolar geometry concept (fig. 37). The use of the epipolar lines allows to verify not only a single pose calibration but the final calibration too, being more useful than the homography process used for single poses. The epipolar geometry concept is a geometric property of a multi-view setup. It is used to correlate the points of a camera to the possible points in the other, because when two cameras are capturing the same scene from different positions, there are geometric relations between the 3D points and the 2D points obtained when they are projected onto the 2D images. This relation makes possible to predict where a certain 3D point captured in the first camera, is positioned in the image of a second camera. When a camera takes a picture of a scene, it looses the depth information of the points captured, leading to an unknown position of that point. Although, we know that this point is situated somewhere in a fictional line that travels along all the depth possibilities. That line is known as Epipolar Line. These lines are the ones used in triangulation systems to estimate a point depth.

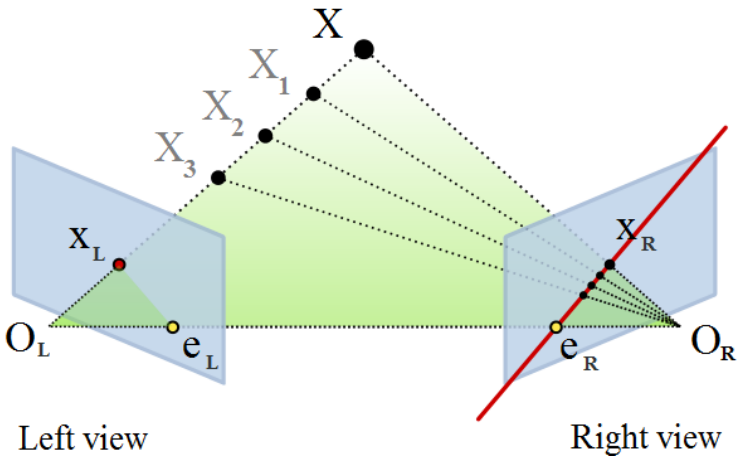


Figure 37: The Epipolar Geometry. (openMVG, 2013)

The relation between the two cameras in pixel coordinates is described by the fundamental matrix, that encapsulates both the intrinsic and the extrinsic parameters of both cameras. Knowing a certain point in a camera image and the fundamental matrix, the correspondent epipolar line can be computed (with the OpenCV we can use the method *computeCorrespondEpilines*).

As mentioned in the previous chapters, the projector can be seen as the inverse of a camera so, the epipolar constraints can also be applied. The process to verify the correctness of the calibration visually is through the computation of the epipolar lines for the detected chessboard corners in the RGB image, projecting them in the projector viewpoint. If the calibration is correct, every line projected will cross the respective point for what it was generated. Figure 38 shows an example of the resultant epipolar lines projected on the physical pattern. In that example only the corners in the last row were considered to ease the process of visually identifying which line should cross each corner, but the same process can be applied to all corners. As can be seen in figure 38, every line cross at a certain point with a chessboard corner in the top of the bottom row. In a bad calibration case, some or all of that lines won't pass exactly through the corners.

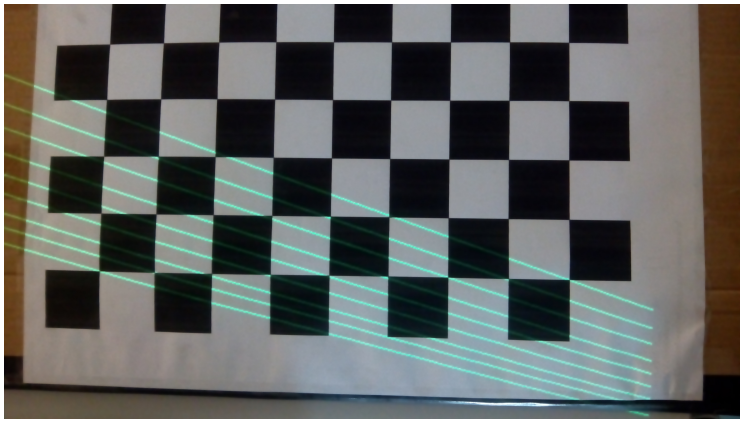


Figure 38: Projected Epipolar Lines

As the calibration module is an independent application, its results are then uploaded through a xml file. The setup is automatically replicated by reading the parameters resultant of the calibration, positioning the cameras and defining its internal properties in the 3D engine (figure 39).

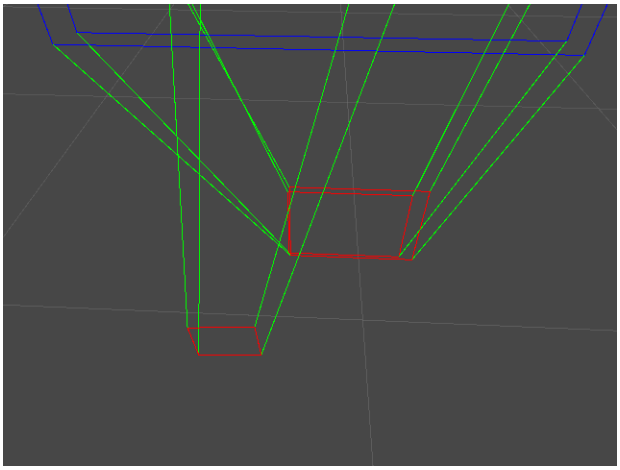


Figure 39: Image of the depth sensor and the projector positioned relatively to the RGB sensor in Unity3D. The depth and RGB sensors are very close to each other. The setup represented in this image is the top-down case exposed in figure 36.

4.3 RECONSTRUCTION

In the reconstruction module a performance analysis was made to verify the possibility to run in real-time, as well a qualitative evaluation of the reconstructed mesh.

To measure the average frame per seconds and time consumed by the main methods multiple captures, for 60 seconds each, of the module running in different scenes were made, after some warm up executions. These captures were made using various resolutions of the TSDF volume, allowing to perceive its influence in the performance of the reconstruction. The table 3 expose the frame rate variation according to the resolution used, as well as the number of triangles of the generated mesh.

Volume Resolution	60	96	128	160	192	256	384	512
FPS	22.31	22.19	21.71	20.94	19.75	13.68	6.46	3.31
Triangles (K)	6	14	27	45	68	129	328	615

Table 3: Average frames per second while running with the same input and varying the Voxel Volume resolution, indicating the average amount of triangles of the mesh generated in each case for the scenes measured.

Analyzing the table we see that increasing the resolution up to 192 has small impact in the performance, but increasing pass that mark the frames per second dropped considerably. Figure 40 details the time consumed by the main steps for each resolution tested. Higher resolutions

of the volume have not been tested due to the hardware limitations. Methods having constant execution time independently of the used resolution were not illustrated.

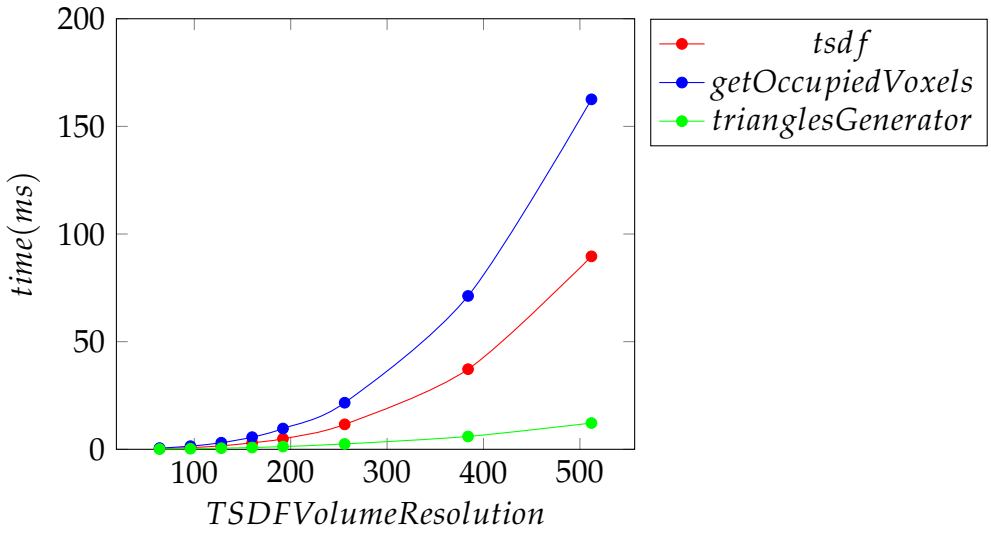


Figure 40: Execution time in milliseconds of the resolution dependent main methods, showing how the time consumed scales when increasing the Voxel Volume resolution.

As we can see in figure 40, both the methods of searching the occupied voxels and the one that continuously updates the TSDF volume increase theirs execution time exponentially when increasing the volume resolution, justifying the significant frame-rate drop when increasing resolution above a certain value.

The following images in figure 41 demonstrate how the resultant mesh varies using the different resolutions. As can be seen, with higher resolution the edges are aliased in a smaller scale, giving a perception of a granular surface but allowing to fit better the real object. We can also observe that for some cases we won't even need a volume with high resolution, allowing the reconstruction at interactive frame-rate.

Depending of the target scenario, a balance has to be found between quality and performance. Note also that the bigger the resolution of the voxel volume, the bigger the number of triangles in the resultant mesh, leading not only to spend more time generating the mesh, but also possibly leading to performance problems in the rendering step, if too many triangles are being generated.

The module has the option to return only the point cloud instead of the final mesh, allowing to change that option in runtime and visually compare the final mesh with the raw pointcloud. The figure 42 illustrates this comparison.

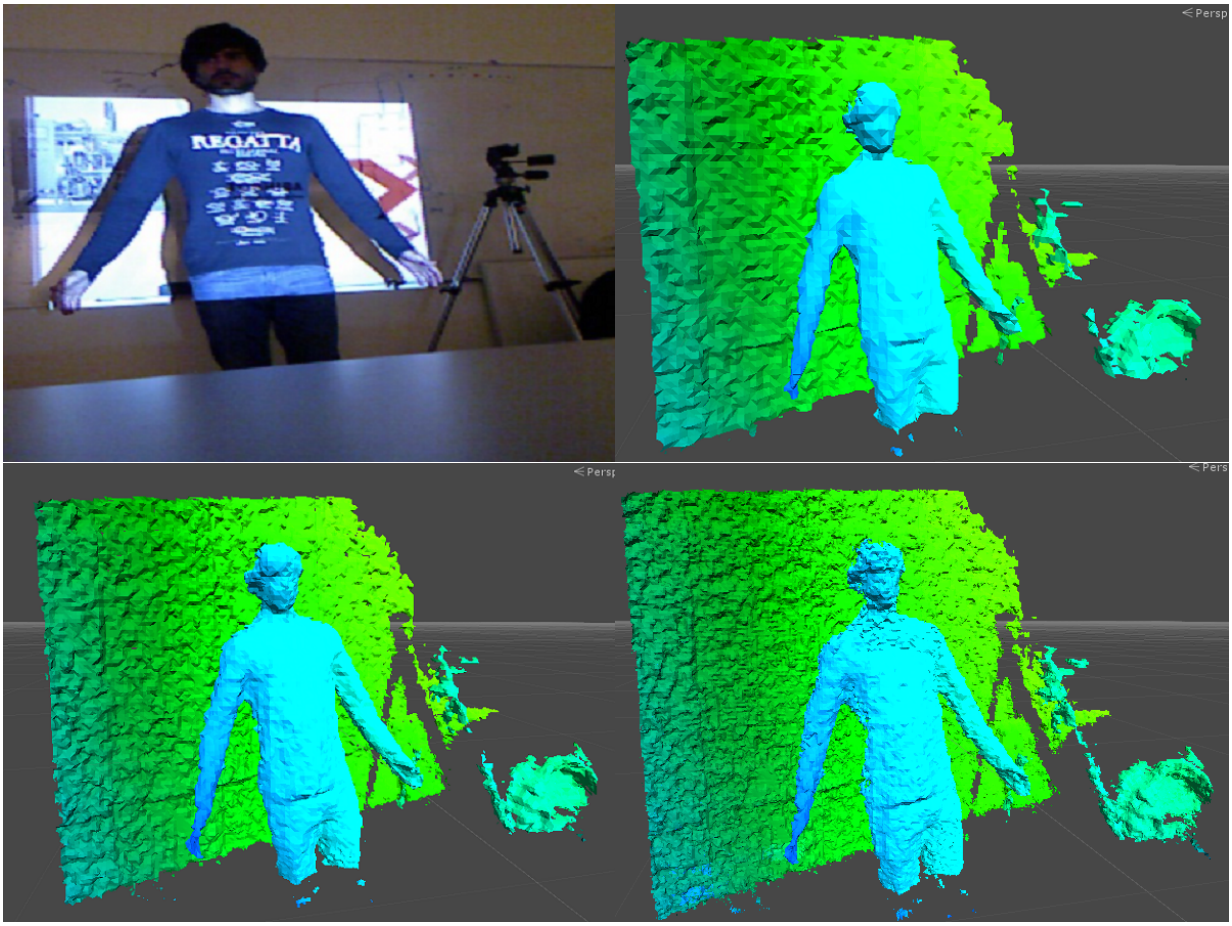


Figure 41: Mesh obtained using different resolutions of the voxel volume for the same scene. The top-left image represents the real scene. The top-right is the resultant mesh using a voxel volume of 96x96x96, the bottom-left uses a volume of 160x160x160 and the last one uses a 256x256x256 voxel grid. The reconstructed meshes were draw in the unity3D engine.

4.4 PROJECTION

The purpose of the projection module alone is a normal projection mapping system, where the scene is previously recreated in the virtual world, with the addition of virtual objects that augment the reality. To verify the results of the module, a static scene was used to obtain results that are independent of another module, leaving for later the analysis of the modules combined.

Figure 43 demonstrates a demo where we can control a character over the scene. The scene is composed by the table-top and a box, and was recreated in a virtual world manually. In this module we need to verify the View Dependent Rendering (VDR) process used to adapt the virtual

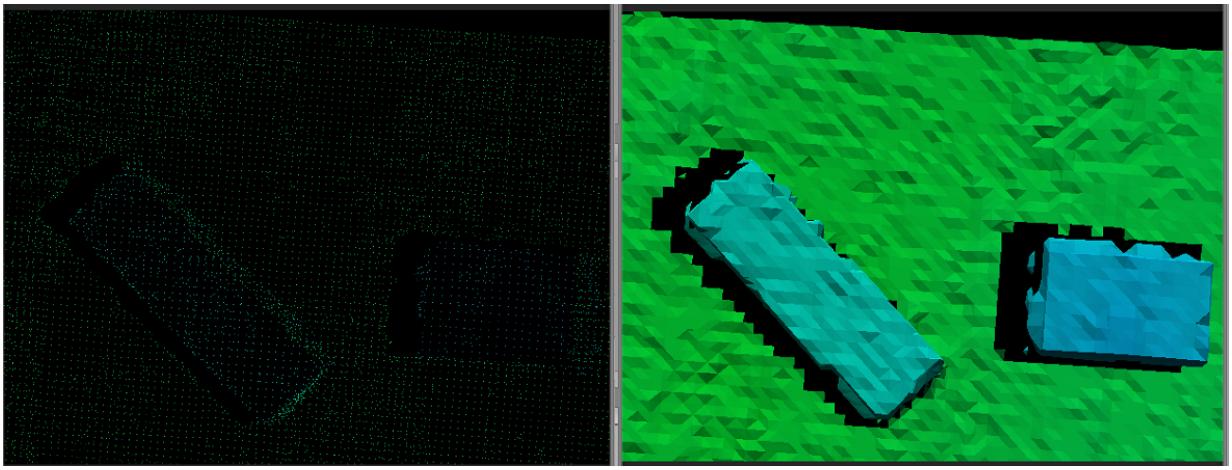


Figure 42: PointCloud generated from depth map in the left and the resultant mesh in the right, both draw in Unity3D side.

objects projection. The virtual character and all the other virtual objects should be projected over the real surfaces in a way that the user sees them correctly, giving a 3D perception.

Figure 43: Results of a demo using a static scene. Both images represent the same scene, although, using different user positions in the rendering process. The images were taken from the positions where the user should be to have a correct perspective.

Is hard to do a quantitative analysis of the resultant image, but through the demos tested such as the one in figure 43 we can say that the result is good. It can adapt the projection based on the position of the user and also to project on top of irregular surfaces as demonstrated by the demos in figures 35 and 48. Figure 44 demonstrates how the resultant projection seems when looked from other positions than the one that the system expects the user to be.

As in the reconstruction module, an analysis of the performance was important to have notion of its impact on the system. The times were measured in the scene represented in figure 43, with 16.7K triangles.

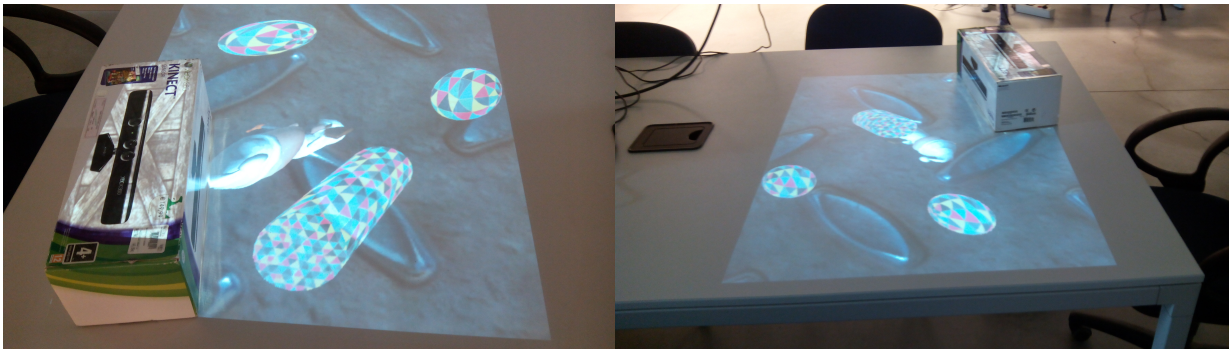


Figure 44: Results of a demo using a static scene when the user is not in the position that the system assume it is. The virtual objects are distorted in both cases and don't represent the real perspective of the user.

Texture Resolution	400	500	600	700	800	900	1000	1250	1500	2000
Avg. FPS	86	57	42	34	23	21	17	11	8	4

Table 4: Performance of the multi-pass rendering technique in a static scene. Texture resolution N represents that a NxN texture was used for the intermediate rendering.

Analogous to the reconstruction module, the higher the resolution of the texture used in the intermediate step of the multi pass rendering, the lower the frame-rate achieved (table 4). While with higher resolutions the result is good, using lower resolutions leads to the typical blurred and aliased images (figure 45). A balance between quality and performance has to be done to achieve interactive frame-rates.

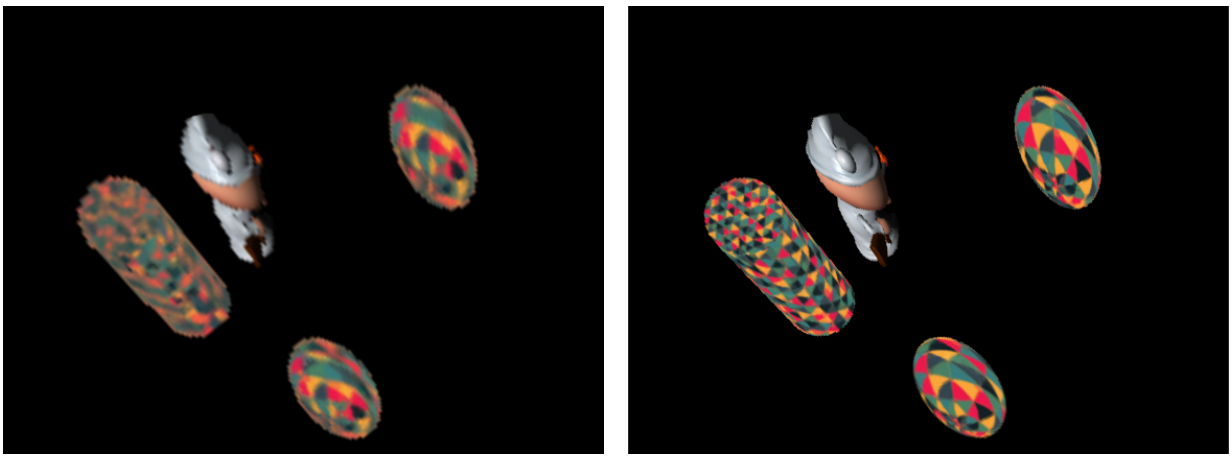


Figure 45: Both images represent the final image to be used when projecting over the real scene. The elements on the image are virtual objects that passed through the multi pass rendering technique. In the left image, the texture used to store the user's view has a resolution of 500x500, while in the right one, that texture has a resolution of 1500x1500.

4.5 GLOBAL SYSTEM

The previous results are module specific, not being the result of the application as a whole. In this section we will analyze the final product of applying the different modules together.

The first thing to verify is if the calibration made is actually correct when applying in the final application. If the verification process in the calibration module succeeded, it will succeed here either. Nevertheless, a demo that just changes the color accordingly to the depth of the surface was implemented. This demo allows to verify if the virtual world is well aligned with the real one. If we see that objects with different heights are colored different and with no displacement, the setup is ready to be used. In figure 46 we can see this verification demo.

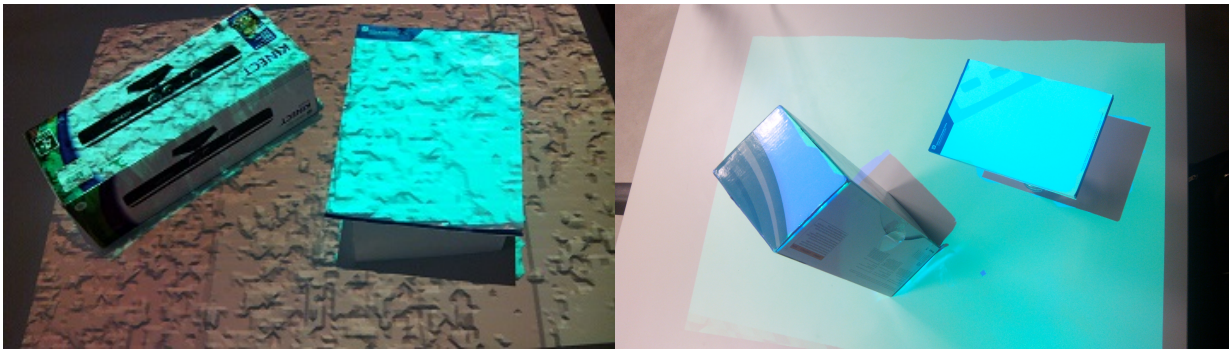


Figure 46: Color attribution from height values. In the left image the virtual surface is lightened to have the exact notion of the reconstructed surface. In the right image the color is directly applied only from the height value, resulting in flat colors.

Another version of this demo was made to project just over some of the foreground surfaces instead of all the scene. In that demo, a simple segmentation by depth ignores the surfaces farther or closer than a certain value, giving the idea the object is tracked when it moves, not illuminating the surfaces in the background. It allows to project just over a human body for example (figure 47), or just the boxes in the scene of figure 46, not illuminating the table itself.

These demos verify both the calibration and that the reconstruction module is working well, allowing to change the appearance of the target surfaces. The simple color attribution by depth was just a proof of concept and can be replaced by more complex materials, being possible to project whatever someone may want on the reconstructed surface. This type of demos follows the typical projection mapping applications but over the reconstructed scenes in real-time instead of static ones.

The system also allows the addition of virtual objects to the scene, adapting theirs projection accordingly to the position of the user. In the images of figure 48, a demo that also uses the VDR

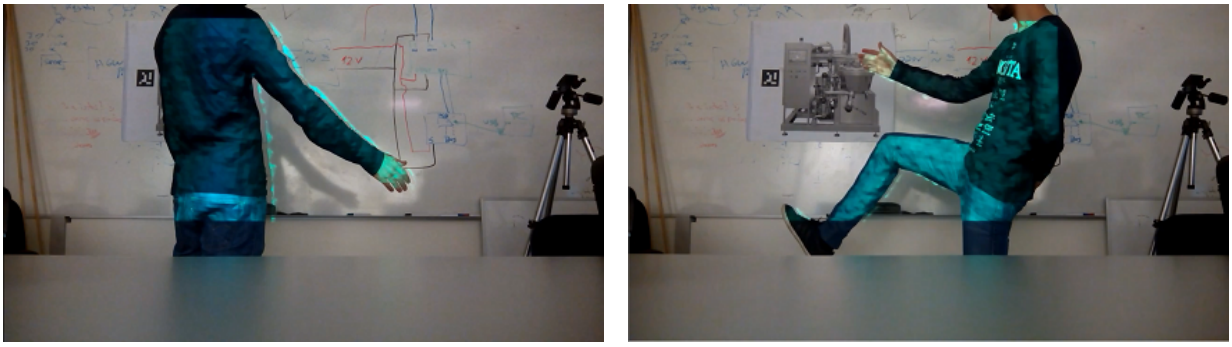


Figure 47: Demo where background surfaces are ignored, projecting just over the foreground surfaces. In this case the body reconstructed is reprojected over the real one.

is represented. It generates the image following the process illustrated in figure 35, but using the reconstructed surface instead of a static one, adapting the virtual objects' projection to the real surfaces.

The virtual objects can be simple static objects or objects with physics that interact with the reconstructed scene. In the example of the figure 48, both the cylinder and the character fall until collide with the reconstructed surface. The character can move around over the table and jump to the top of the box. The surface is updated in runtime, allowing to move the box or add another objects on top of the table changing the scene while playing. Although in this example the table and box are not illuminated, with the intuit to simply add the virtual objects to the real world as it is, a material can be applied to change theirs appearance as in the previous examples.

As the system as a whole depends on the modules analyzed before, the global performance depends on the performance of its components, but when referring to the global system, we need to separate the measured values in two types. The first one, is related with the frequency that the mesh representing the scene is updated, while the second one represents the frequency that a new frame is rendered.

The number of times the mesh is updated is considerable lower than in the reconstruction module alone, as the generated mesh needs to be sent to the 3D engine and applied in the virtual scene. In the lower resolution tested for the voxel volume (64x64x64), the values were close to the ones measured as an isolated module, maintaining 20 updates per second of the mesh in the 3D engine. When increasing the volume resolution, the overhead of applying it in the engine increases, reducing from an average of 21 updates per second at a voxel volume resolution of 160x160x160, to an average of 16 updates per second. Using higher resolutions for the voxel volume breaks the interaction with the scene, achieving an average of only 8 updates per second to a resolution of 256x256x256, and 2 frames per second to a resolution of 512x512x512.

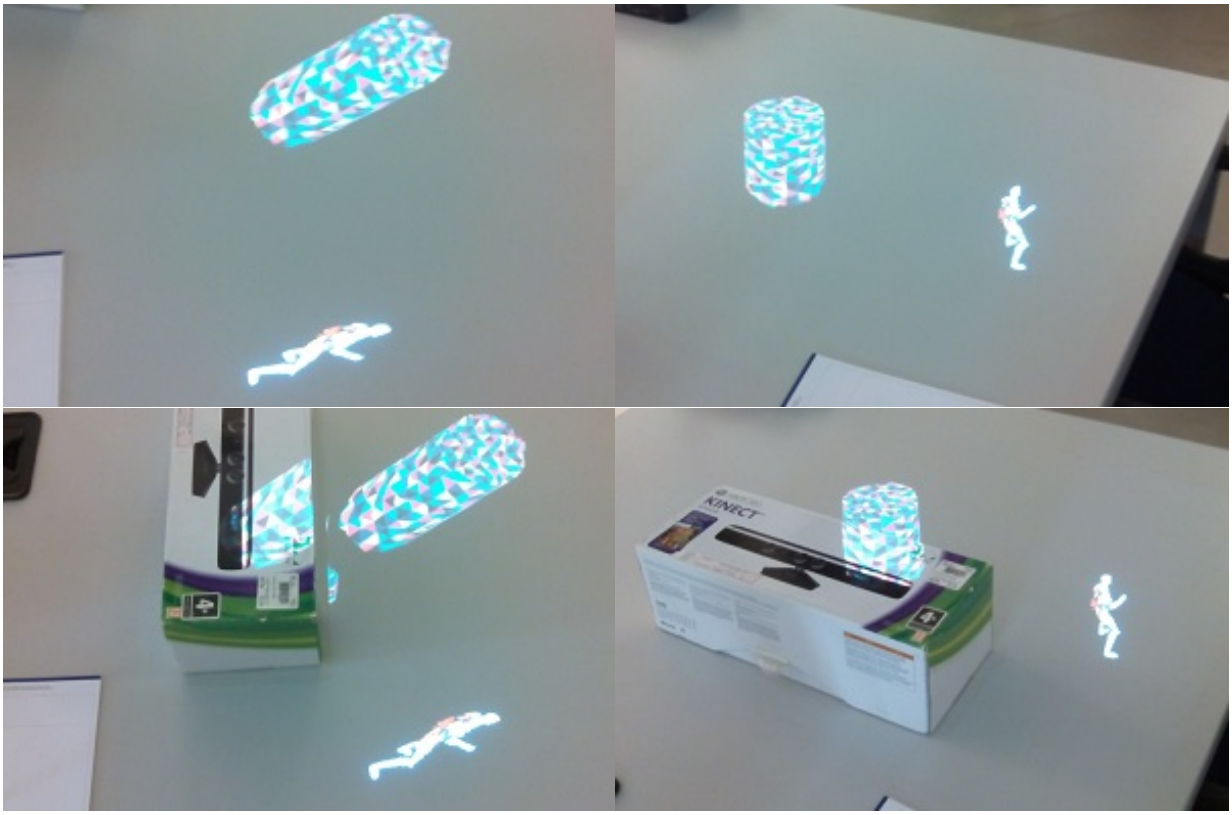


Figure 48: Demonstration where the real scene is reconstructed and used as the scenario of a game. The virtual objects are projected adapting to the surface reconstructed. The top images reconstruct a simple planar top of a table, and in the bottom ones a box is added to the scene. The left images are taken from the top and the right ones from the user perspective. As seen in these images, the image generated for projection adapts accordingly the objects added to the scene.

With respect to the frequency that a new frame is rendered, it can be higher or lower than the mesh updating rate, depending on the quality we need that the virtual objects have. If there are no virtual objects at all in the scene, the multipass render doesn't need to be activated, achieving higher frame-rates. With a voxel volume resolution of 96x96x96, the average frame-rate amongst the tested scenes is around 70 frames per second, much higher than the mesh updating rate in the game engine of 17 frames per second. When adding virtual objects to the scene, the inverse can also happen. If we need that the virtual objects are projected with good quality, without blurred zones and aliased edges, the use of a high resolution texture in the intermediate step of the multi pass renderer makes the frame-rate drop significantly. Using a texture of 1000x1000 pixels, the frame is updated just 12 times per second, a little below than the 17 updates of the mesh per second but maintaining interactive rates. If a texture of 2000x2000 pixels is needed, the frame is updated just 4 times per second, which breaks the interactivity.

5

CONCLUSIONS AND FUTURE WORK

The purpose of this work was to explore the architecture of a Live Projection Mapping system. The system handles projecting in dynamic surfaces and is able to merge virtual objects to the real scene, adapting the projection to the scene accordingly to the user position.

To achieve this goal the framework was divided in three independent modules that work together in the final architecture. The first is responsible for the calibration of the depth sensor and projector relatively to each other, the second should reconstruct the surface intended for projection, and the last one should handle the projection itself and the addition of virtual objects, adapting the projection to the real scene dependently of the viewer's position.

The implemented system can achieve interactive frame rates by making a balance of quality and speed. The surface reconstruction step requires a significant computation time, but by using lower resolutions on the voxel grid it performs at relatively good frame-rates. When running isolated and using resolutions for the voxel volume of 192x192x192 or below, it performs at around 20 frames per second. When invoked on the global system, the overhead of applying the resultant mesh on the game engine makes the frame-rate drop. However, it can obtain a minimum of 15 updates per second with resolutions of 160x160x160 or below.

To obtain a quick update of the surface we need to reduce the quality of the reconstruction. As the depth sensor used has low resolution, depending of the target scene, higher resolutions of the voxel volume does not necessarily lead to better results. For the majority of cases, lower resolutions of the volume results in reasonable results, thus, the performance is hardly a problem. The other component that contribute to lowering the frame-rates is the multi-pass renderer step to achieve a VDR on the virtual objects. The higher resolution we use for the intermediate texture in that step, the slower the process become, but with lower resolutions the resultant projection is blurred and aliased. The balance between quality and speed in this component is critical to achieve interactive frame-rates.

The system works well when handling opaque objects with whiter colors, without or with little texture. Projection on surfaces that are near parallel to the projector light are hardly noted, particularly in reflective surfaces. Surfaces almost parallel to the sensor are not reconstructed as well, and when dealing with projection of virtual objects, the holes in the mesh breaks the result in some cases, both in the projection and in the physical interactions.

The first improvement to the system to do as future work is to replace the sensor used (Kinect v1) by the Kinect v2, not only because it has better accuracy in the depth map, but also because it has a significantly better resolution of the RGB image. Although that image is not used in the run time application, the RGB sensor is used as intermediate in the calibration process. As explained in the results, the low resolution of the RGB sensor in Kinect v1 leads to restrictions in the sensor's position.

Another important addition to the system left for future work is the implementation of an head tracking module. By now the user position is simulated, not being updated as the user moves. As the reconstruction module, this module could easily be added to the system as a plugin.

The mesh generated from the reconstruction module is very limited, as it only represent the surface visible from the sensors viewpoint, resulting in holes that can ruin the result for certain perspectives. The method chosen allows to merge the input of multiple sensors, that if positioned around the scene can scan the entire surface. This will result in a mesh of the complete scene without holes.

Another interesting change in the reconstruction module is to replace the actual one with one using a data driven method. This way multiple known objects could be detected and added to the virtual world individually. The reconstructed objects will be more accurate than with a normal reconstruction method, having in mind that we need a previously made model of the objects that will be used.

For cases where a part of the scene is static, a reconstruction module with background subtraction was initiated but not completed due to time restrictions, allowing to separate the reconstruction of the surfaces that will be fixed during the execution and the dynamic ones. By applying the background subtraction only the surfaces on the foreground will be reconstructed in real-time, allowing to enhance the reconstructed surface of the static one in an off-line step. This process could be used in conjugation with a data driven method as described before, reconstructing the background in an off-line step and use a data driven method to detect and track the foreground objects in real-time.

As we can use multiple depth sensors to obtain data from all perspectives with the purpose of generating a mesh without holes, multiple projectors could also be used, projecting over the

surfaces that cannot be projected from the viewpoint of a single projector, handling as well occlusions between objects.

REFERENCES

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421. ACM, 1998.

Murat Arikan, Michael Schwärzler, Simon Flöry, Michael Wimmer, and Stefan Maierhofer. O-snap: Optimization-based snapping for modeling architecture. *ACM Transactions on Graphics*, 32(1):6:1–6:15, January 2013. ISSN 0730-0301.

Andreas Baak, Meinard Müller, Gaurav Bharaj, Hans-Peter Seidel, and Christian Theobalt. A data-driven approach for real-time full body pose reconstruction from a depth camera. In *Consumer Depth Cameras for Computer Vision*, pages 71–98. Springer, 2013.

Dirk Bartz and Michael Meißner. Voxels versus polygons: A comparative approach for volume graphics. In *Volume Graphics*, pages 171–184. Springer, 2000.

Hrvoje Benko, Ricardo Jota, and Andrew Wilson. Miragetable: Freehand interaction on a projected augmented reality tabletop. *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*, pages 199–208, 2012.

Hrvoje Benko, Andrew D. Wilson, Federico Zannier, and Hrvoje Benko. Dyadic projected spatial augmented reality. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 645–655, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3069-5. doi: 10.1145/2642918.2647402.

Matthew Berger, Joshua A. Levine, Luis Gustavo Nonato, Gabriel Taubin, and Claudio T. Silva. A benchmark for surface reconstruction. *ACM Trans. Graph.*, 32(2):20:1–20:17, April 2013. ISSN 0730-0301. doi: 10.1145/2451236.2451246.

Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Joshua A. Levine, Andrei Sharf, and Claudio Silva. State of the art in surface reconstruction from point clouds. *Eurographics STAR (Proc. of EG'14)*, 2014.

Oliver Bimber and Ramesh Raskar. *Spatial Augmented Reality: Merging Real and Virtual Worlds*. A. K. Peters, Ltd., Natick, MA, USA, 2005. ISBN 1568812302.

Oliver Bimber and Ramesh Raskar. Modern approaches to augmented reality. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. doi: 10.1145/1185657.1185796.

Oliver Bimber, Andreas Emmerling, and Thomas Klemmer. Embedded entertainment with smart projectors. *Computer*, 38(1):48–55, 2005.

Erik Bylow, Jürgen Sturm, Christian Kerl, Fredrik Kahl, and Daniel Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems (RSS) Conference 2013*, volume 9, 2013.

Andrea Canessa, Manuela Chessa, Agostino Gibaldi, Silvio P. Sabatini, and Fabio Solari. Calibrated depth and color cameras for accurate 3d interaction in a stereoscopic augmented reality environment. *Journal of Visual Communication and Image Representation*, 25(1):227–237, 2014.

J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383266.

R. Catanese. 3d architectural videomapping. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W2:165–169, 2013. doi: 10.5194/isprsarchives-XL-5-W2-165-2013.

Jiawen Chen, Dennis Bautembach, and Shahram Izadi. Scalable real-time volumetric surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(4):113, 2013.

ShengYong Chen. Three-dimensional object reconstruction, 2012. URL http://hwyc1000.com/csy/research/vision/model.htm#system.

Zhi-Quan Cheng, Yan-Zhen Wang, Bao Li, Kai Xu, Gang Dang, and Shi-Yao Jin. A survey of methods for moving least squares surfaces. In *Volume Graphics*, pages 9–23, 2008.

Leandro Cruz, Djalma Lucio, and Luiz Velho. Kinect and rgbd images: Challenges and applications. In *SIBGRAPI Tutorials'12*, pages 36–49, 2012.

C Cruz-Neira, Dj Sandin, and Ta DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. ... *of the 20th annual conference on* ..., pages 135–142, 1993.

Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996.

Herbert Edelsbrunner and Ernst P Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, 1994.

Cass Everitt. Projective texture mapping. *White paper, NVidia Corporation*, 4, 2001.

Gabriel Falcao, Natalia Hurtos, and Joan Massich. Plane-based calibration of a projector-camera system. *VIBOT master*, 9(1):1–12, 2008.

Gabriele Fanelli, Thibaut Weise, Juergen Gall, and Luc Van Gool. *Real time head pose estimation from consumer depth cameras*, volume 6835 LNCS, pages 101–110. 2011.

Open Source Robotics Foundation. Intrinsic calibration, 2015. URL http://wiki.ros.org/openni_launch/Tutorials/IntrinsicCalibration. Accessed: 2015-10-28.

Jason Geng. Structured-light 3d surface imaging: a tutorial. *Adv. Opt. Photon.*, 3(2):128–160, Jun 2011. doi: 10.1364/AOP.3.000128.

Anselm Grundhofer and Oliver Bimber. Real-time adaptive radiometric compensation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(1):97–108, 2008.

Alexander Hornung and Leif Kobbelt. Robust reconstruction of watertight 3 d models from non-uniformly sampled point clouds without normal information. In *Symposium on Geometry Processing*, pages 41–50, 2006.

iSandBox. isandbox, 2013. URL http://isandbox.ru/. Accessed: 2015-10-21.

Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047270.

Brett Jones, Rajinder Sodhi, Michael Murdock, Ravish Mehra, Hrvoje Benko, Andrew Wilson, Eyal Ofek, Blair MacIntyre, Nikunj Raghuvanshi, and Lior Shapira. Roomalive: Magical experiences enabled by scalable, adaptive projector-camera units. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 637–644, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3069-5. doi: 10.1145/2642918.2647383.

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. *Poisson surface reconstruction*, pages 61–70. 2006.

Oliver Kreylos. Augmented reality sandbox.

Florent Lafarge and Pierre Alliez. Surface reconstruction through point set structuring. In *Computer Graphics Forum*, volume 32, pages 225–234. Wiley Online Library, 2013.

Robert Lange and Peter Seitz. Solid-state time-of-flight range camera. *Quantum Electronics, IEEE Journal of*, 37(3):390–397, 2001.

Johnny C Lee, Paul H Dietz, Dan Maynes-Aminzade, Ramesh Raskar, and Scott E Hudson. Automatic projector calibration with embedded light sensors. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 123–126. ACM, 2004.

Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J Mitra. Globfit: Consistently fitting primitives by discovering global relations. In *ACM Transactions on Graphics (TOG)*, volume 30, page 52. ACM, 2011.

William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.

Microsoft. Kinect for windows features. URL http://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx. Accessed: 2015-01-15.

F R de Miranda, R Tori, C E S Bueno, and L P Trias. Designing and implementing an spatial augmented reality x-ray. *Revista de Informática Teórica e Aplicada*, 15(3):47–74, 2009.

Daniel Moreno and Gabriel Taubin. Simple, accurate, and robust projector-camera calibration. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 464–471. IEEE, 2012.

John Morgan. Real-time 3d reconstruction using signed distance functions, 2014. URL http://www.personal.psu.edu/users/j/p/jpm5375/assignment6.html/. Accessed: 2015-09-02.

Thiago Motta, Manuel Loaiza, Alberto Raposo, and Luciano Soares. Kinect projection mapping. *SBC*, 5(5):20, 2014.

nVidia. Generating complex procedural terrains using the gpu, a. URL http://http.developer.nvidia.com/GPUGems3/gpugems3_ch01.html. Accessed: 2015-09-05.

nVidia. The cg tutorial, b. URL http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter09.html. Accessed: 2015-09-05.

Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. In *ACM SIGGRAPH 2005 Courses*, page 173. ACM, 2005a.

Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. An integrating approach to meshing scattered point data. In *Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 61–69. ACM, 2005b.

openMVG. open multiple view geometry, 2013. URL http://imagine.enpc.fr/~moulonp/openMVG/coreFeatures.html. Accessed: 2015-10-28.

openMVG. Cameras, 2015. URL http://openmvg.readthedocs.org/en/latest/openMVG/cameras/cameras/. Accessed: 2015-10-28.

Panasonic. How focal length affects viewing angle. URL http://av.jpn.support.panasonic.com/support/global/cs/dsc/knowhow/knowhow12.html. Accessed: 2015-10-20.

Soon Yong Park and Go Gwang Park. *Active calibration of camera-projector systems based on planar homography*, pages 320–323. 2010.

Thammathip Piumsomboon, Adrian Clark, and Mark Billinghurst. Physically-based Interaction for Tabletop Augmented Reality Using a Depth-sensing Camera for Environment Mapping. In *Proc. Image and Vision Computing New Zealand (IVCNZ-2011)*, pages 161–166, Auckland, Dec 2011.

PointCloudLibrary. Using kinfu large scale to generate a textured mesh. URL http://pointclouds.org/documentation/tutorials/using_kinfu_large_scale.php#using-kinfu-large-scale. Accessed: 2015-09-02.

Qualcomm. Qualcomm: Vuforia. URL https://www.qualcomm.com/products/vuforia. Accessed: 2015-01-03.

Ramesh Raskar, Greg Welch, Kok-Lim Low, and Deepak Bandyopadhyay. Shader lamps: Animating real objects with image-based illumination. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 89–102, London, UK, UK, 2001. Springer-Verlag. ISBN 3-211-83709-4.

Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

Joaquim Salvi, Jordi Pages, and Joan Batlle. Pattern codification strategies in structured light systems. *Pattern Recognition*, 37(4):827–849, 2004.

Ruwen Schnabel, Patrick Degener, and Reinhard Klein. Completion and reconstruction with primitive shapes. In *Computer Graphics Forum*, volume 28, pages 503–512. Wiley Online Library, 2009.

Andrei Sharf, Thomas Lewiner, Ariel Shamir, Leif Kobbelt, and Daniel Cohen–Or. Competing fronts for coarse–to–fine surface reconstruction. *Computer Graphics Forum*, 25(3):389–398, 2006. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2006.00958.x.

Chao-Hui Shen, Hongbo Fu, Kang Chen, and Shi-Min Hu. Structure recovery by part assembly. *ACM Trans. Graph.*, 31(6):180:1–180:11, November 2012. ISSN 0730-0301. doi: 10.1145/2366145.2366199.

Martina Spickova. Comparison of surface reconstruction methods for mobile robotics. Bachelor's thesis, Czech Technical University In Prague, 2013.

Mengwen Tan, Weipeng Xu, and Dongdong Weng. isarprojection: A kinectfusion based handheld dynamic spatial augmented reality system. In *Proceedings of the 2013 International Conference on Computer-Aided Design and Computer Graphics*, CADGRAPHICS '13, pages 425–426, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-1-4799-2576-6. doi: 10.1109/CADGraphics.2013.79.

thefullwiki. Structured light 3d scanner, 2009. URL http://www.thefullwiki.org/Structured_Light_3D_Scanner. Accessed: 2015-10-28.

Inna Tishchenko. Surface reconstruction from point clouds. Bachelor's thesis, Swiss Federal Institute of Technology in Zurich, 2010.

vvvv. How to project on 3d geometry, 2013. URL http://vvvv.org/documentation/how-to-project-on-3d-geometry. Accessed: 2015-08-04.

Chris Walker. Stereo vision basics, 2014. URL http://chriswalkertechblog.blogspot.pt/2014/03/stereo-vision-basics.html. Accessed: 2015-08-08.

Min Wan, Desheng Wang, and X Tai. Surface reconstruction with feature preservation based on graph-cuts. *UCLA CAM Report*, 12:58.

wikipedia. Marching cubes, 2010. URL https://en.wikipedia.org/wiki/Marching_cubes. Accessed: 2015-10-28.

Andrew D Wilson and Hrvoje Benko. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 273–282. ACM, 2010.

Yi Xu and Daniel G Aliaga. Robust pixel classification for 3d modeling with structured light. In *Proceedings of Graphics Interface 2007*, pages 233–240. ACM, 2007.

Z. Zhang. *Camera Calibration*, pages 4–43. Prentice Hall Professional Technical Reference, 2004.

Zhengyou Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.

Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. Elastic fragments for dense scene reconstruction. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 473–480. IEEE, 2013.