UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

MARCELO SOUZA VASQUES

Environment Mapping applied to Medical Surgery Simulators

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engeneering

Advisor: Prof. Dr. Marcelo Walter

Porto Alegre
July 2018

ABSTRACT

Virtual surgical simulators are important to medical training on minimally invasive surgeries and should provide visual realism in order to accurately reproduce the environment of real medical operations. Due to the high cost involved in Global Illumination (GI) rendering, current simulators use local illumination models, compromising realism. Using data extracted from videos of real surgeries, we explain a new approach to obtain the visual data needed to add environment mapping techniques to a surgery simulator in a local illumination framework, approximating the results to a GI context. We also detail the implementation of a simple medical scene simulator using a liver object to validate the whole approach.

Keywords: Environment Mapping. Image Mosaicing. Medical Simulator. Laparoscopic Surgery. Local Illumination.

RESUMO

Simuladores virtuais cirúrgicos são importantes para o treinamento médico em procedimentos minimamente invasivos, devendo apresentar realismo visual para que os ambientes reais de operações médicas sejam reproduzidos com precisão. Contudo, devido ao alto custo envolvido na renderização de Iluminação Global, os simuladores atuais utilizam modelos de iluminação local, o que compromete o realismo. Usando dados extraídos de vídeos de cirurgias reais, nós explicamos uma nova abordagem para a obtenção de dados visuais necessários para que uma técnica de Environment Mapping seja adicionada a um simulador cirúrgico em uma estrutura de iluminação local, o que aproxima o resultado de um contexto de Iluminação Global. Além disso, nós também detalhamos a implementação de um simulador de cenas médicas simples usando um objeto de um fígado, a fim de validar toda a abordagem.

Palavras-chave: Environment Mapping, Image Mosaicing, Simulador Médico, Cirurgia Laparoscópica, Iluminação Local.

LIST OF ABBREVIATIONS AND ACRONYMS

GI	Global Illumination
MIS	Minimally Invasive Surgeries
SURF	Speeded Up Robust Features
SVD	Singular Value Decomposition
RANSAC	Random Sample Consensus
SFL	Surface Light Field
FMM	Fast Marching Method

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Medical applications are a relevant area of research in Computer Graphics due to its direct impact on prevention and treatment of health related issues (VIDAL et al., 2006; PREIM; BOTHA, 2013). Computer generated graphical tools are able to reproduce real medical environments with varied realism. Virtual surgical simulators, for instance, are important to medical training and can provide a safe environment to students and professionals when learning surgical techniques that are later used on real situations. Some types of surgeries can be even easier to virtually reproduce on a simulator, such as the minimally invasive surgeries (MIS), since they do not require a physical interaction between the surgeon and the patient, needing only a screen that shows the content provided by a camera that moves inside the patient's body, attached to the laparoscope, and some kind of tool used by the surgeon to manipulate the laparoscope. In other words, as defined by (NEYRET; HEISS; SéNéGAS, 2002), laparoscopic surgeries are "a non-invasive technique thats consists of introducing through small holes in the patient's abdomen several micro-instruments and an optic fiber connected to a camera and a light source". In Fig. 1.1, this surgical scenario is presented.

Figure 1.1: Doctors performing a minimally invasive surgery using a laparoscope attached to a monitor.

Source: <http://www.medstarhospital.co.in/service/viewservice/laparoscopic-surgery>

However, in order to actually offer a satisfactory immersion in the surgical simulator and to provide a credible application, visual realism is a crucial characteristic to be achieved.

To accurately reproduce the environment of real medical operations, it is necessary

to understand the lighting present in a real surgery, and how this light interplays with the tissues during the surgery. A local illumination model is a simple model where interactions are local to the object. Enhanced approaches can provide increased realism by using global illumination rendering, where the multiples bounces of light in the environment are considered. Nonetheless, due to the high cost involved in such techniques, current simulators use pure local illumination models, compromising immersion and realism. A conciliation of these two scenarios, however, enables a more realistic simulation with low processing cost. This combination is possible by adding a simulation of the light reflected inside the environment, a feature that is not part of a standard local illumination model, and brings the simulation closer to a GI context.

We propose a new approach to the lighting used on a surgical simulator by adding a precomputed environment mapping technique, which improves a local illumination context by considering the light contributions created by the background environment. Several videos from real surgeries are analyzed before the reproduction of the content on the simulator and, by composing a credible synthesized environment texture of the surgical scene, our method enables the reflections of the background to be calculated and displayed on the surface of the rendered object. The result is an application that can be run on current computers, with clear improvements to the pure local illumination model. This way, we present an executable medical scene simulator provided with a precomputed texture that requires to be created only once, for each surgery type.

The organization of this work consists of a brief description of important concepts explained in Chapter 2, altogether a summary of the current literature related to the objective. In Chapter 3, we detail the methods and the approach we implemented to obtain the synthesized texture of the background and the simulation of the medical scene. The results of these implementations are then presented and commentated in Chapter 4. We conclude this text with Chapter 5, where we summarize the steps described and open some possibilities for future work.

2 BASIC CONCEPTS AND RELATED WORK

In this chapter, we present the main definitions needed to understand this work. We begin by explaining basic concepts of Environment Mapping and Epipolar Geometry, which relates two images (or two views) by using geometry, and gives a precise perspective of the scene represented by the images. Later, we present the definitions of some important algorithms and methods used in our implementation, such as SURF, inpainting, SVD and RANSAC. Finally, we review some existing similar contributions to our approach.

2.1 Environment Mapping

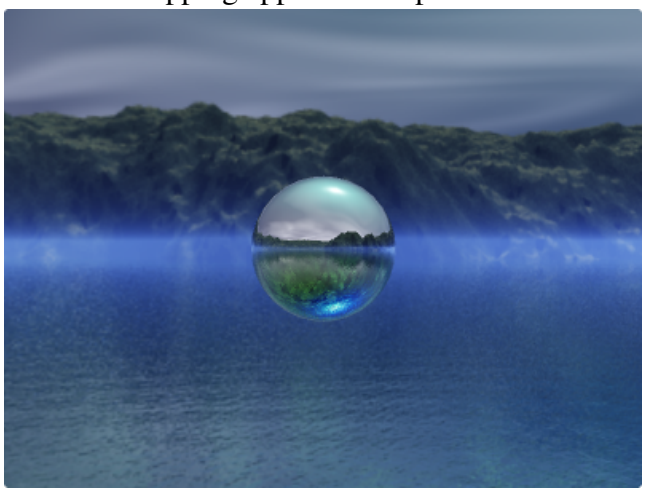
Environment mapping, which characterizes one of many different procedures researched on the graphical illumination field, can be summarized as a “reflection mapping” model (BLINN; NEWELL, 1976). It describes the reflected light that influences the object on every point of its visible surface. In other words, as explained by (LALONDE; EFROS, 2010), "an environment map is a sample of the plenoptic function at a single point in space," (ADELSON; BERGEN, 1997). Therefore, a way to represent this data is needed. Some authors resort to a field representation of the light, as in the Surface Light Field (SFL) technique (CHEN et al., 2002). However, extracting these informations from a video sequence is not a straightforward process, as described by (PALMA et al., 2013).

Other authors establish their methods on simpler and more direct spherical, cylindrical or cube mapping techniques, as presented by (SZELISKI; SHUM, 1997) and (HEIDRICH; SEIDEL, 1998). In Fig. 2.1, a classic result of a Environment Mapping application is shown with the use of a sphere.

2.2 Epipolar Geometry

As presented by (LIM; DE, 2007), a well-known image based rendering technique is image mosaicing, first introduced by (INAMPUDI, 1998), which describes the creation of a larger image by putting together several smaller images side by side, possibly with overlaps. This idea is important to this work because it enables the reconstruction of the background around an object, which is crucial to the effects of environment mapping, as

Figure 2.1: Environment mapping applied to a sphere that reflects the scene light.



Source: <http://rbwhitaker.wdfiles.com/local--files/reflection-shader/screenshot1.png>

shown on (LALONDE; EFROS, 2010), describing a Stereo Imaging approach.

Hartley and Zisserman (HARTLEY; ZISSERMAN, 2004) explain how the translation and rotation information can be found on a Stereo Imaging technique, and this can be achieved by firstly estimating the Fundamental and Essential matrices. They both require data that is not easily accessible to the algorithm, such as the camera parameters, but by estimating these values it is possible to output usable results. The Fundamental and Essential matrices are similar concepts, but the Fundamental matrix can be considered as the generalization of the Essential Matrix, in which the camera parameters are not present. We explain below both concepts.

2.2.1 Fundamental Matrix

Let us consider two images from the same scene, but with different viewpoints from the same object in space. The epipolar geometry establishes a few characteristics that can relate both images. For example, if the same point X in the scene space is represented on each image as the three-dimensional vectors x and x', respectively, in some way they are related and there is a way to convert the point in the first image to the one in the second image. This is where the Fundamental matrix comes in, and it is a unique 3×3 matrix of rank 2. The main condition that must be satisfied by the fundamental matrix F when dealing with points x and x' is, as described by (HARTLEY; ZISSERMAN, 2004),

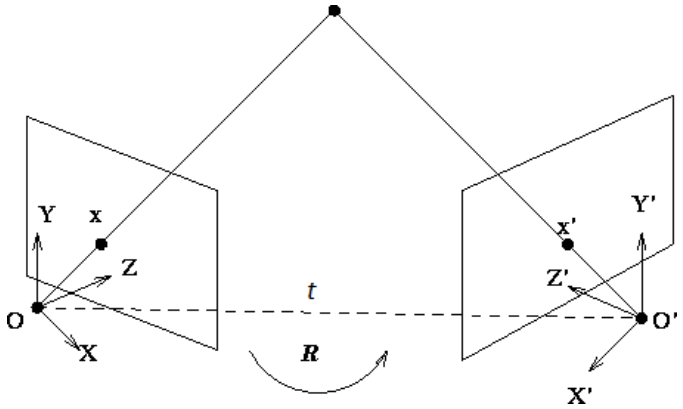
$$x'^T F x = 0$$

where x'^T is the transposed vector of the point x'.

2.2.2 Essential Matrix

The Essential matrix is the Fundamental matrix including the camera parameters. As explained by (HARTLEY; ZISSERMAN, 2004), the Essential matrix also contains only normalized image coordinates. The relation between the Fundamental and Essential matrices is given by $E = K^T F K$, where K is the intrinsic camera parameters matrix, K^T is its transposed version and F is the Fundamental matrix. In Fig. 2.2 we show the relation between the two views attached to the observers O and O' and the points x and x', together with the rotation R and the translation t.

Figure 2.2: The relationship between corresponding points x, x', the translation and rotation between them, shown as t and R.



Source: adapted from <http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT10/img21.gif>

Another important property of the Essential matrix is that it can be written as a relation of the rotation matrix R and the translation *t* as

$$E = R[t]_\times$$

where R is the rotation matrix and $[t]_\times$ is the skew-symmetric matrix representation of the vector t.

The Essential matrix is important in this work because it enables the possibility of estimating the translation and rotation data from the video frames used in the analysis. This way, it is possible to estimate the relationship from one camera position to another. Therefore, the image mosaic can be built relying on this information. After we select

the origin for the mosaic, we just need to position the pictures according to their estimated translation, overlapping one another. In Fig. 2.3, we show how an example of this technique can be used to create a larger image.

Figure 2.3: Greater image built by a mosaic of smaller images.

Source: <http://www.sci.utah.edu/~cscheid/spr05/imageprocessing/project2/imgs/big_yosemite_raw.jpg>

2.3 Speeded Up Robust Features (SURF)

Presented by (BAY et al., 2006), SURF was introduced as a new approach to solve the problem of finding corresponding points in a stereo image situation. The algorithm outperformed the existing methods for the same task and became one of the main mechanisms to achieve 3D reconstruction, image recognition and geometry estimation on image processing projects.

The SURF algorithm works on a three-steps process, which basically detects interesting points on both images and then elaborates a description of the neighbourhood in which these points are inserted. During the last step, the execution is responsible for a matching between the descriptors, which finally outputs the selected points for the whole algorithm.

During the interesting point detection, which is processed in grayscale, as the other parts of the algorithm, the method relies on integral images and on the Hessian matrix. The former is the sum of the values of the pixels from a rectangular region, while the latter bases its construction on a matrix composed by the convolution of second order derivatives of the Gaussian from a determined pixel.

The Hessian matrix filter used by the algorithm is responsible for the selection of the interesting points, and this is done by calculating the determinant of the Hessian

matrix for each pixel. A parameter called Hessian threshold can be applied here, being in charge of how high the determinant must be to correspond to a selected pixel. In other words, a higher threshold results on more distinctive pixels.

2.4 Inpainting

Filling blank spaces of an image can be achieved by using digital inpainting techniques, and one of the most efficient works in this field is the algorithm introduced by (TELEA, 2004). In contrast to comparable contributions, the author aims for a fast and reliable result, maintaining the lines of equal gray values when reconstructing the missing spaces. When running, the data contained in the neighborhood is essential to the first selected pixel, and that is why the execution starts by processing points located in the boundaries of the holes.

The main point of the algorithm is to estimate the smoothness value of the image along its gradient, and this process is done by using a weighted average on a known image neighborhood of the pixel that is being inpainted. The method starts by analysing a gray image and then applies the same obtained data to its colored version. After the first selected pixel has its new value calculated, the information is propagated to the other missing points by using the fast marching method (FMM) (SETHIAN, 1996).

2.5 Singular Value Decomposition (SVD)

As explained by (STRANG, 2016) and (BLUM; HOPCROFT; KANNAN, 2018), any matrix can be decomposed into simpler pieces by using the SVD method. This process can be used on applications as an image compressor, for example. Basically, what the authors explain is that any $m \times n$ matrix can be factorized into three other matrices of low rank, which we call U, V^T and W:

$$A = UWV^T$$

This decomposition is based on the concept of writing the matrix A as a sum of other matrices, where U is called the set of left-singular vectors and V^T the set of right-singular vectors. While U is $m \times m$, V^T is $n \times n$. Lastly, the singular values are all contained in the diagonal of the matrix W, which is also $m \times n$.

2.6 Random Sample Consensus (RANSAC)

In 1981, (FISCHLER; BOLLES, 1981) introduced RANSAC as a method for fitting a model on a set of experimental values. It is particularly recommended for image processing analysis due to its capability of dealing with outliers, mainly because of wrong outputs coming from feature detectors. One of the algorithm main characteristics is the use of random subsets of data values to increase its chances of a correct result.

As the selection of data values is random, the method gives non-deterministic outputs. The main idea proposed by the authors is to start with a small subset of data values. In the next step, this sample set is used to compute the model, and then to find other data values that are within the same error margin tolerance. Using a threshold to evaluate the size of the resulting set, the method then utilizes the largest consensus data set found to compute the final model.

2.7 Related Work

In this section, we review related efforts on the topics addressed in this work. We first summarize similar contributions to the medical simulation field, as done by (NEYRET; HEISS; SéNéGAS, 2002) and (LIM; DE, 2007). We also present some results that explore different Environment Mapping techniques, such as the construction of the reflection model based on a single image (LALONDE; EFROS, 2010) and the composition of an image mosaic (SZELISKI; SHUM, 1997) as the reflected source. As our objective is analyzing video frames to build the environment mapping, we also bring some concepts from (PALMA et al., 2013), which explains how to extract scene data from video clips and introduces a different approach to store the light information of the scene, known as Surface Light Field.

2.7.1 Realistic Rendering of an Organ Surface in Real-Time for Laparoscopic Surgery Simulation

The purpose of (NEYRET; HEISS; SéNéGAS, 2002) is to describe a realistic rendering of a human liver, including its reactions to the surgery and collisions to the medical instruments. The authors go further when detailing the realistic aspects of their

work, describing techniques for cauterization, blood drops and whitening effects.

In their model, they use three layers of textures to achieve a satisfactory appearance to the surface of the liver object. They are basically described as the skin texture, the dynamic reactions and the reflection mapping layers. The resulting object rendering is the sum of all these layers, which are processed in real time and change during the execution.

2.7.2 Synthesizing Environment Maps from a Single Image

In this paper, Lalonde and Efros (LALONDE; EFROS, 2010) describe how to actually reproduce a location-dependent environment map on an object virtually inserted in a real scene projected in a single image. Several simplifications and estimations are introduced by the authors, such as assuming that the reflection is basically Lambertian, in which every surface point emits the same amount of light in all directions, and that the geometry of the scene is way simpler than it actually is. They also assume that the unseen side of the scene captured in the image is basically the same as the one shown in the picture. The estimated scene geometry is used to deform the original photo in some different ways. After mirroring the modified picture, the method consists on warping this image around a sphere, which is then ready to be placed inside the original scene.

2.7.3 On some Recent Advances in Multimodal Surgery Simulation: a Hybrid Approach to Surgical Cutting and the use of Video Images to Enhance Realism

The authors present a new new hybrid approach for a surgical cutting simulation (LIM; DE, 2007). To create a more realistic effect when seeing the surgery simulation, the article contains the description of image-based rendering techniques, which are used to enhance lighting presentation by extracting data from multiple images captured in video frames.

Image mosaicing is one of the approaches implemented and it is responsible for image registration and for seams elimination by blending. Then, using view-dependant texture mapping, each frame is analyzed: images from the same object, but with different viewing angles, can be blended to create a new view, by using weighted averages for pixels. The weight values are chosen to be inversely proportional to the angles. This last concept is used to provide a realistic glistening effect on the tissues.

2.7.4 Surface Light Field from Video Acquired in Uncontrolled Settings

The main idea introduced in this paper (PALMA et al., 2013) is to estimate the Surface Light Field from an object registered in some video frames. In order to obtain the SLF, they separate the diffuse component of the surface appearance from the other view dependent SLF lighting effects that are responsible for the residual color of the object. The estimation process of the SLF takes four steps to calculate the diffuse color and the coefficients associated with the spherical function. They also introduce an user-defined parameter to change the intensity of the residual component, which makes the algorithm less independent.

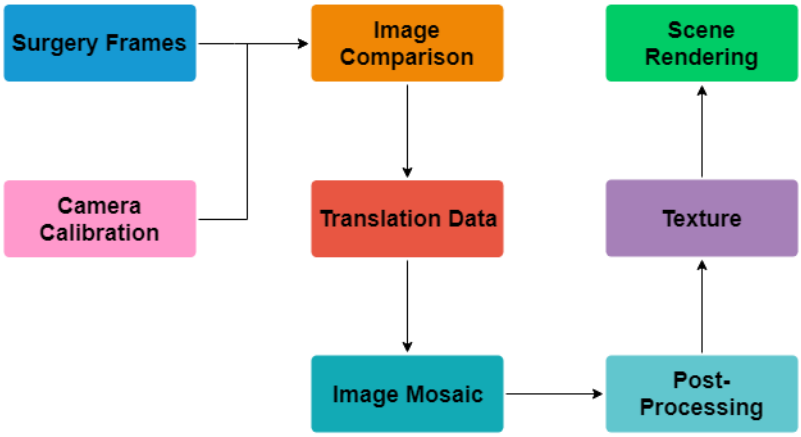
2.7.5 Creating Full View Panoramic Image Mosaics and Environment Maps

(SZELISKI; SHUM, 1997), however, bases its concepts on a fixed camera position, with only a variable rotation. The authors then detail their approach for estimating the focal length of the camera based on the calculated rotation matrix of the images. After describing the method they use to fill the holes in the panorama, which fulfills the space with another image, they also describe how to transform the obtained mosaic into an environment map of the scene, by utilizing a sphere mapping technique.

3 METHOD

In this chapter, the description of the steps applied during the implementation of this project are presented. The algorithms and their implementation details are explained alongside the methodology chosen for this work.

Figure 3.1: Pipeline with general steps implemented in the texture analysis and synthesis section of the program.



Source: The authors.

To illustrate the whole development, we present the implementation of the project in two parts, which are integrated to compose the whole program. In the first section, we introduce the methods used to construct the texture, characterizing the Texture Analysis and Synthesis step, which takes image frames as input and compose a new texture based on data extracted from an analysis of their characteristics. The general view of this part is shown in Fig. 3.1.

Then, we detail the concepts of the Medical Scene Simulator techniques that are used to render the human organs in a 3D scene environment, in real-time. This step bases itself on the texture output of the first part and aims to achieve a reproduction of the real medical applications for this project.

3.1 Texture Analysis and Synthesis

Since we target the synthesis of a convincing organic human texture, the algorithm first analyzes the data contained in images from real human body tissues. These images serve as input to texture synthesis during this first part of the process, as they are combined together as a mosaic and then refined to elaborate a more realistic composition.

The mosaic construction requires camera parameters so the translation between a pair of images can be estimated. The real distance between the different points of view shown by pairs of images is an information obtained after calculating the fundamental and essential matrices, which were explained earlier in this document.

To implement the image processing techniques, we used the OpenCV 3.2 library and wrote the application in C++, using Visual Studio. After the initial composition of the mosaic, the resulted texture is used directly in the rendering of the organ simulation, which is described in the next section.

3.1.1 Surgery Videos

The images used to synthesize the final texture were all extracted from videos of real surgeries. In Fig. 3.2, we show one frame from one of our videos. The medical procedures were recorded with laparoscopes inside the patients bodies. This way, the collection of videos includes more than two hours of content from different parts of the surgeries. However, to maintain the similarity between the images used as input for texture synthesis, we selected small clips from the videos that represent the same organ in the frames.

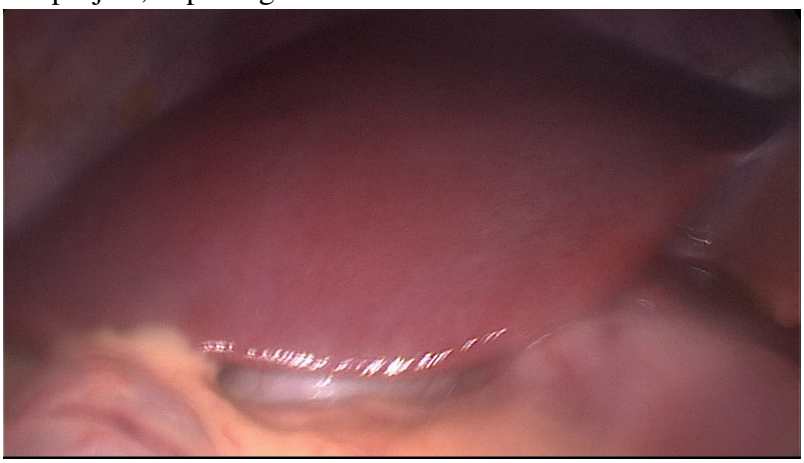
The available videos present challenges for image processing, since there is a few amount of light coming from the source attached to the laparoscope. These difficulties exist because of the low camera specifications generally chosen for real surgery environments, considering that image quality is not such an important requisite when compared to others such as the final size of the laparoscope equipment.

Some frames from the videos also contain artifacts that are not part of human tissue, such as the body of the laparoscope itself and dark borders from the lens of the camera. These unwanted parts of the frames can actually appear on the final synthesized texture, but we chose to avoid clips that had images with these types of artifacts during the processing of the frames.

3.1.2 Camera Parameters and Calibration

As described in the Basic Concepts chapter, to correctly estimate the Essential Matrix, it is first necessary to know the intrinsic camera parameters. The Essential Matrix

Figure 3.2: Frame extracted from one of the laparoscopic surgery videos used in the database for this project, depicting a liver.



Source: The author

is vital to this work because it provides useful information such as the rotation and the translation between two different points of view.

After knowing these parameters, we are able to correct both radial and tangential distortions, which are responsible for straight lines appearing curved inside the images, for example. They are called intrinsic because they are specific to a camera model construction. Therefore, if the camera model is changed, the parameters are also modified. This data is generally represented inside a 3×3 matrix, as shown below,

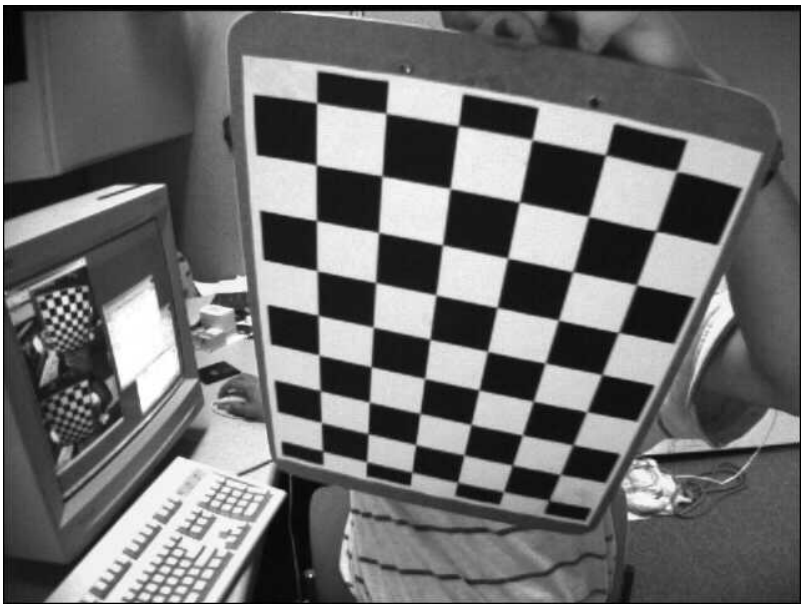
$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

where f_x and f_y are the focal length components and c_x and c_y are the values of the optical center of the camera.

Unfortunately, data such as the camera parameters are not easily know, particularly when we deal only with the recorded product of the camera equipment, without further complementary information. Since this is the exact scenario where this work is included, we prepared an experiment for a real laparoscope, implementing a program for the calibration of its camera. This goal, however, as presented in Chapter 4, was not achieved since we were not able to use a real laparoscope for calibration. The implemented calibrator was then tested on a regular smartphone camera.

The program we created follows the image processing pipeline for camera calibration suggested in the documentation of the OpenCV library (OpenCV, 2016), and bases its analysis on a known black-and-white chessboard pattern. A real representation of the

Figure 3.3: Example of a picture that includes a chessboard pattern with focal distortion, which can be used to calibrate a camera.



Source: OpenCV documentation.

pattern must be printed on a flat surface, such as a dense piece of paper, and the disposition of the rectangles inside the pattern needs to be informed as an input to the program.

The execution then analyses several photographs taken from the same chessboard pattern, each one from a different angle and point of view. An example of a printed chessboard pattern can be seen in Fig. 3.3. Knowing the size of the pattern on each image, the program then uses the OpenCV function to identify the corners of the rectangles inside the photos.

As the rectangles inside the chessboard pattern are all located in the same flat surface, their 3D corners positions have a Z value of zero when using the scene pattern local coordinate system. Therefore, after the correct identification of the patterns, it is possible to correlate both 3D object points and their equivalent 2D image points, thus estimating the distortion caused by the camera specifications, and therefore its intrinsic parameters. This data is then used directly as an input to the texture synthesizer.

3.1.3 Image Comparison

To build the whole mosaic, the first step after calibration is to compare each pair of images and find out what they have in common. We apply SURF on each image so the algorithm can found interesting keypoints in both images separately. The output of

Figure 3.4: Graphical representation of the keypoints selected from an image by the SURF algorithm, shown inside the red circles.

Source: https://docs.opencv.org/3.0-beta/doc/

this process can be seen in Fig. 3.4. The OpenCV (OpenCV, 2014) library has a built in SURF function that analyzes images with a parameter called minimum Hessian filter threshold, which, as explained before, corresponds to the distinctiveness level the point needs to achieve to be selected as output.

The comparison between frames is done on pairs of subsequent frames, as this project is based on stereo imaging approaches. In this process, SURF outputs both keypoints and the descriptors for a single photo. The description data is used when the program matches the results that come from the two images. As they include information about the keypoint and its neighbourhood, the descriptors are then utilized as the main input to the brute-force matching algorithm that OpenCV provides.

In this step of the program, each pair matched by the brute-force algorithm is loaded with a distance value assigned by the matching method. This value describes how similar the paired match is, with low values being the better options. Depending on the Hessian threshold chosen for SURF, which can give more or less keypoints as output, this distance value can be very significant to refine and reduce the number of matches, so the program can work with the few best corresponding points.

We implemented a function to find the minimum distance of all paired matches after checking all the results from SURF. This value is then used to create an interval of accepted corresponding pair distances, so the best matched pairs can be stored in a list. To estimate the Fundamental Matrix, these selected points are the foundation of the

geometric calculations that are described in the next subsection.

3.1.4 Fundamental and Essential Matrices

As mentioned before, the Fundamental and the Essential Matrices are used to compute the translation between two images. In this subsection, we detail exactly how this data is obtained based on a series of mathematical operations. In our approach, we first estimate the Fundamental Matrix, which is basically the Essential Matrix without the intrinsic camera parameters. Then, by using the approximation of these parameters that we obtain in the calibration process, we construct the Essential Matrix, which can be decomposed with methods such as the singular-value decomposition (SVD) into the data we need to estimate translation.

We already know that, by using algorithms such as SURF, it is possible to identify estimated corresponding points on both views shown by the images. Since they are described by different observation points, it is natural that the 3D space coordinates of these views have translation and rotation differences between one another. Assuming a point x in the first image has a corresponding x' in the second one, the correspondence between them must satisfy

$$x'^T F x = 0$$

where F is the Fundamental Matrix and x'^T is the transposed matrix of the point x'.

When talking about the Essential Matrix, however, the image coordinates have to be normalized before we assume its own properties. So, if x is a point in image A, we can call $\hat{x}$ its normalized version. This way, the same adapted definition valid for the Fundamental Matrix is also valid for the Essential Matrix E

$$\hat{x}'^T E \hat{x} = 0$$

where $\hat{x}'^T$ is the transposed matrix of the point $\hat{x}'$, which is the corresponding point in image B.

To obtain the normalized version of x, as we already know the estimated intrinsic camera parameters thanks to calibration, we can perform the following operation to undo the effects of the camera specifications on the coordinate

$$\hat{x} = K^{-1}x$$

where K^{-1} is the inverse of the matrix K, which contains the intrinsic camera parameters.

This way, substituting $\hat{x}$ and $\hat{x}'$, it is possible to use the property of the Essential Matrix to find

$$x'^{T}K'^{-T}EK^{-1}x = 0$$

Comparing this result to the property of the Fundamental Matrix, we can finally find a way to obtain the Essential Matrix based on F

$$E = K^{T}FK$$

This is exactly what is done after obtaining the Fundamental Matrix of the two images in the implementation. After the program identifies satisfactory keypoints matches, it finds an estimated version of the Fundamental Matrix by executing a function of the OpenCV library. In this case, this function uses the RANSAC method to improve the quality of the results. As described by the OpenCV documentation, this function runs the following equation for every pair of corresponding points, represented here as x' and x

$$x'^{T}Fx = 0$$

which is, in fact, the same basic property of the Fundamental Matrix we introduced before.

Once the Fundamental Matrix is estimated, the next step of the implementation is the application of the formula that relates both F and E, using the camera intrinsic parameters matrix, which we are calling K. It is a simple process and at the end we have the Essential Matrix, ready to be decomposed into the translation data.

3.1.5 Extracting Translation from the Essential Matrix

As explained before, some of the information that can be extracted from the Essential Matrix are the rotation matrix and the translation vector, obtained by using concepts of epipolar geometry. However, as detailed in the next chapter, we take into consideration only the translation values when building the image mosaic, which proved to be enough to achieve satisfactory results. The data extraction can be done because E can be written

as

$$E = R[t]_\times$$

where R is the rotation matrix and $[t]_\times$ is the skew-symmetric matrix representation of the vector t.

The problem with this form of the Essential Matrix is that it has five degrees of freedom, which means we have to decompose the matrix into a different state such that we can solve for the variables that compose the rotation and translation values. This process can be achieved by applying the singular-value decomposition (SVD) method to E.

When we apply SVD on the matrix E, the decomposed result consists of three different matrices. The OpenCV library has a class named SVD that performs the decomposition and stores the resulting matrices into three different variables, which we call here U, V^T and W, as a representation of the naming convention used to represent the singular vectors in the SVD method. However, these three matrices do not correspond directly to the rotation and translation values. To obtain them, we need to combine these three results.

As shown in (OLSSON, 2013) and (HARTLEY; ZISSERMAN, 2004), the Essential Matrix has to be decomposed into two matrices, so it can look like the first form presented in this subsection. This way, E can be written as

$$E = SR$$

where S is skew-symmetrical and R is a rotation.

The authors use two predefined matrices to achieve the decomposition. One of them, the matrix we introduced as W earlier, is orthogonal and a representation of the rotation in unitary form

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The other matrix, which they call Z, is composed of the following values

$$Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

It is proven in (OLSSON, 2013) and (HARTLEY; ZISSERMAN, 2004) that E can be decomposed into matrices S and R by starting with W and another matrix, which they call Z. Nevertheless, what really matters here is W. Since we already decomposed the Essential Matrix using SVD, we can recover the rotation matrix by multiplying W and two of the matrices resulted from the SVD execution. These matrices are U and V^T, which means

$$R = UWV^T$$

or

$$R = U \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T$$

and

$$S = UZU^T$$

After having the rotation matrix R, the next step is to find the translation vector t. Fortunately, this process is easier and, as shown by (OLSSON, 2013), t is the null space of S, which corresponds to the third column of U. Since this column has three values, it can be seen now as a 3D vector.

3.1.6 Image Mosaicing

Based on the Essential Matrix, as explained in the last subsection, we obtain the translation that describes the estimated distance between the different points of view, in a unitary vector, since the Essential Matrix is already normalized into image coordinates. It is important to clarify that this data is part of a stereo-imaging analysis, which means the translations between the frames are always based on single pairs of images. We can have as many frames as it is necessary, but they are always analyzed in subsequent combinations of pairs of images.

The translation data is stored in a list, and the next step is to bring the spatial coordinates from each pair of images into a whole scene. As they are all frames from the same object, which is, in this case, an human organ, the spatial scene depicts a closed

Figure 3.5: Artificially created desired result of the image mosaic, with three images that broaden the view of the scene background.

Source: The authors.

contiguous environment. Thus, the algorithm uses the translation values to render a plane with all the analyzed images placed where the estimated positions of their points of view should be. The algorithm is susceptible to errors that come from wrong point description or detection during the object recognition step performed by SURF. In this case, some images can be placed on a wrong position inside the mosaic, interfering in the final texture. It is not an intermittent problem, but in a few results some images can be misplaced due to this effect.

This process results on a mosaic composed by different images that represent a larger view of the scene, with a larger background. This mosaic is rendered using OpenGL in a 3D space, and each image is positioned accordingly to its calculated translation, starting with the first processed photo, which is interpreted as the origin. Here, a single image corresponds to the texture of a 2D rectangle drawn in its estimated position inside the mosaic plane. An example of the desired result of this process is presented in Fig. 3.5, where three images of a gift box are placed on top of one another according to the box movements. All planes are scaled equally to fit inside the rendering space delimited by OpenGL, as well as their translations.

3.1.7 Post Processing Improvements

After the processing of the images and the construction of the mosaic, we obtain a fully synthesized texture by capturing the view of the camera in OpenGL. This method, however, has a disadvantage, since the produced image still has some blank spaces that are not filled by the positioned photos. This problem is even more visible on the borders

of the texture, as the captured view is a rectangle and the translated images generally overlap one another with different offsets. Therefore, some post processing techniques are needed to refine the final result.

We implemented three methods to improve the quality of the resulting texture, and they are shown below, in the same order as they are executed in the implementation:

1. Alpha Blending
2. Margin Reduction
3. Inpainting

The first one, alpha blending, is not technically processed after the capturing of the camera view, but during the composition of the mosaic. It is a very simples process, and started with the necessity of making the images inside the mosaic appear more merged when combined. This is achieved by the compilation of a pair of vertex and fragment shaders that force the single image textures to have a variable alpha value. This way, the frames can all blend together.

Another problem to the captured view of the mosaic is that it may have uneven margins, filled with blank spaces. In this case, our OpenGL renderer fills the void with black, which is an easy color to identify. Thus, we created a function that starts with a rectangular window with the same size of the captured image. The execution iterates, on each margin, always jumping one pixel towards the center of the image matrix. If the whole row or column of pixels is black, the margin of the new window is reduced. By the end of the function, this new reduced window is used to crop the texture, thus minimizing the black pixels in the margins.

Inpainting techniques are well known for their capacity of recovering damaged pictures. It is an interesting method to fill holes in images with missing portions, which is exactly our problem here with the texture. The algorithm introduced by (TELEA, 2004) is one of the most used forms of inpainting. Fortunately, it is available on OpenCV with a customizable neighbourhood parameter that checks smaller or bigger regions of the borders of the holes.

However, to correctly run the OpenCV method, the function expects a mask corresponding to the holes to be filled with synthesized patches. Therefore, we implemented an algorithm to check for the black pixels of the image, which are the pixels we want to be painted, and then invert their values to white. This composes a mask of white pixels of the same size of the original texture, as expected by the OpenCV function.

After all three post-processing methods, the final texture is finally ready to be utilized on our rendered simulation. This is the next step for this project, and its execution is started after the synthesized texture is obtained, as it is explained in the next section.

3.2 Medical Scene Simulator

To correctly show the processed texture, and to demonstrate the potential applications for our implementation on real surgery simulators scenarios, we built a fully functional 3D renderer based on OpenGL 4.5. To achieve this, we used the FreeGLUT 3.0 and GLEW 2.1 libraries, along with OpenCV 3.2. Again, we used Visual Studio and chose C++ as the programming language to build the application.

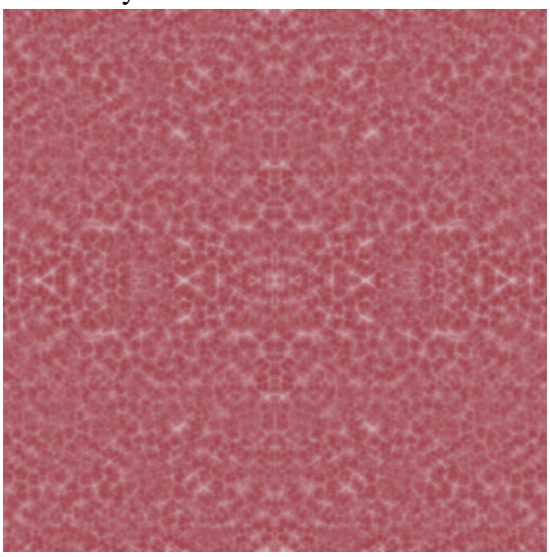
The renderer is composed of a few main elements, which are the object rendering, the shaders construction, the illumination, and the skybox or skysphere rendering. These portions are all part of the application that renders the scene in real time. In the next subsections, we explain how each of these components work.

3.2.1 Organ Rendering

The organs loaded and shown on screen during the program execution are the result of a OBJ loader we implemented, which fully supports vertices, faces, normals and textures coordinates description. The implementation is provided with an $.obj$ file that contains the whole data representing the organ. Then, along with its base texture, which can be observed in Fig. 3.6, the program loads and calculates the texture and lightning contributions on each vertex, so it can render the result on screen. The texture mapping, however, is handled by the shaders.

We use three matrices to control rendering and viewing coordinates, which are known as Model, View and Projection matrices. They are responsible for coordinate transformations of the object, world and camera scopes, respectively. To handle these matrices correctly, we included the OpenGL Mathematics (GLM) library to the project, which has many built-in matrix functions, such as scaling, translating and rotating.

Figure 3.6: Scientifically accurate base texture used for the liver object.



Source: Local database.

3.2.2 Shaders and Illumination

We already explained that, during the construction of the image mosaic, each image plane is attached to a pair of vertex and fragment shaders responsible for the alpha blending. In modern OpenGL, these pairs of shaders work together inside the rendering pipeline that is intrinsic to the library. While the vertex shader outputs vertex data to build the object primitives, the fragment shader works with fragments coming from the rasterization process (OpenGL, 2017).

We also attach different pairs of shaders to each object rendered. After they are compiled and linked to an object during execution, they are basically responsible for texture mapping, using data such as vertex position. However, there is a more complex case, which is the organ rendering. A rendered object can have different contributions of colors on its surface, which are categorized as emissive, ambient, diffuse and specular, and are linked to the lighting of the scene, as described in (VERTH; BISHOP, 2015).

Emissive light comes from an emissive source, such as a light bulb, and can be seen as an added constant value to the surface. Ambient light contribution is also constant and is based on the light's ambient color and the color of the ambient's material, which is, in our case, the mapped texture of the environment. It can be written as

$$C_A = i_L L_A M_A$$

where C_A the ambient contribution, i_L is the illuminance, L_A is the light's ambient color

and M_A is the ambient's material.

Diffuse light depends on the angle of the incident light and can be seen as

$$C_D = i_L L_D M_D max(\hat{L} \cdot \hat{n}, 0)$$

where C_D the diffuse contribution, i_L is the illuminance, L_D is the light's diffuse color, M_D is the diffuse material, $\hat{L}$ is the light direction and $\hat{n}$ is the surface normal.

Specular light is responsible for the shining effect on the object's surface, which means it is the white highlight reflection, and it can be seen as

$$C_S = \begin{cases} i_L L_S M_S max(\hat{r} \cdot \hat{v}, 0)^m, \text{ if } \hat{L} \cdot \hat{n} > 0 \\ 0, \text{ otherwise} \end{cases}$$

where C_S the specular contribution, i_L is the illuminance, L_S is the light's specular color, M_S is the specular material, $\hat{r}$ is the reflection vector, $\hat{v}$ is the viewpoint vector, m is the shininess coefficient, $\hat{L}$ is the light direction and $\hat{n}$ is the surface normal.

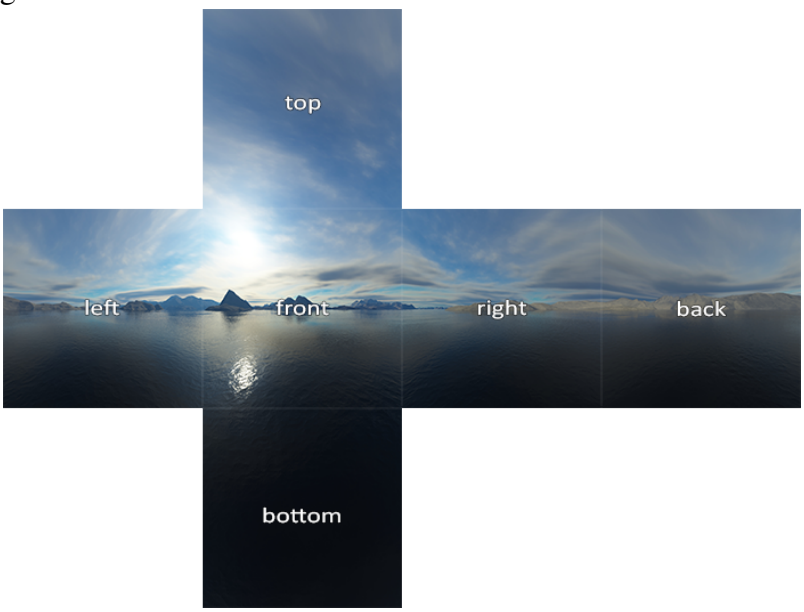
Inside the fragment shader attached to the organ object, these light contributions are all added together and stored in a variable, considering we place one emissive light source right above the object. This final value is the output of the shader, which means it is calculated for each fragment, and corresponds to the fragment color on the object's surface.

3.2.3 Skybox and Skysphere

Considering that the real organ has a complete tissue environment around itself, and to provide a convincing scene that surrounds the rendered object, we implemented two distinct forms of rendering the background in OpenGL, which are techniques known as skybox and skysphere (or skydome). Both options project a texture on a 3D model drawn around the camera view. In a skybox, this model is a cube, while in a skysphere situation, as the name suggests, it is a sphere.

OpenGL supports a texture format for texturing a cube in a process known as cube mapping. However, this cube map format expects six different images, one for each face of a cube. An example of the composition of this texture format can be seen on Fig. 3.7. As we use the final texture synthesized in our image processing step as an input to build this background, we had to implement an algorithm to break this single image in

Figure 3.7: Images composition of a cubemap texture, showing the division of the six faces in a single texture.

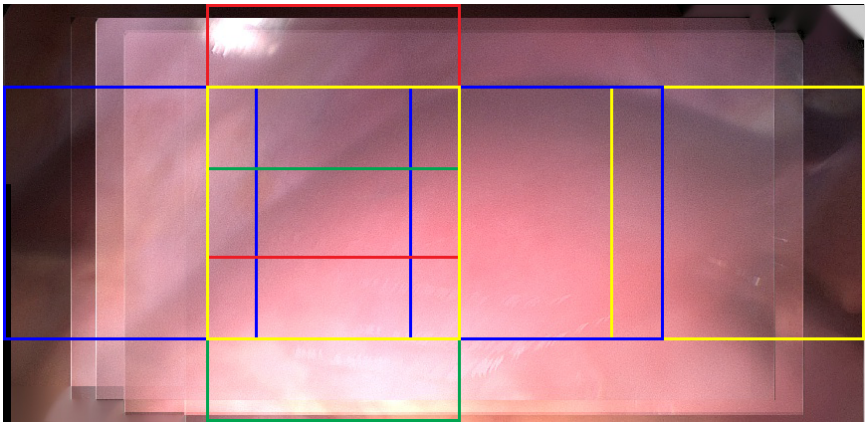


Source: https://learnopengl.com/Advanced-OpenGL/Cubemaps

six smaller portions, and then crop the texture. The result is very similar to the example shown in Fig. 3.7, but with each one of the six squares being a part of the synthesized texture. As each face has a width and a height that are powers of two, these values do not always divide the texture into equal parts, which means the squares have to overlap one another to fit inside the image. This cropping process is shown in Fig. 3.8, where each square is represented by a different color.

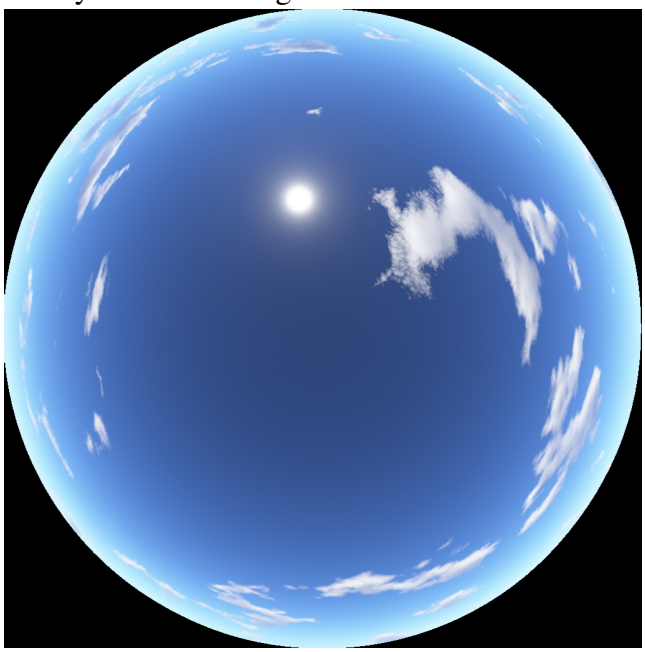
However, the support offered by the library has some limitations: the width and height of each image used on a single face have to be the same and a power of two. These requisites were added to the cropping algorithm of the whole synthesized texture, with

Figure 3.8: Cropping example of the texture into six squares with the same size to build a cubemap. Each square is used as a face of the cube.



Source: The authors.

Figure 3.9: Skydome rendering seen from the outside of the sphere.



Source: http://www.tutorialsforblender3d.com/Skybox

some accepted overlap between the cropped portions. To find the closest power of two to the fixed size of each image face, we used the following formula, which is easily deducted from the basic properties of the logarithm

$$x = (ceil(\frac{\log y}{log2}))^2$$

where x is the next power of two, $ceil$ is a function that rounds the number to the next integer and y is the fixed size of each image portion.

To render the cube, we used four 3D vertices for each face, resulting on an allocated model with a total of 24 vertices. The program also compiles and attaches a specific pair of shaders that, thanks to the support of OpenGL to cube mapping textures, describes a simple texture mapping to the object's vertices.

The skysphere, however, is somewhat more difficult to render, as it requires a sphere model to be loaded. Fortunately, our .obj loader can handle this task easily, and a simple sphere model exported from a program such as Blender is adequate to build the skysphere. A representation of a sphere mapped with a sky texture is shown in Fig. 3.9. Also, the shaders attached to the skysphere are rather more complex, as they perform the mapping of a 2D texture onto a 3D spherical object. This operation is purely mathematical and characterizes a sphere mapping technique.

Again, the objects depend on the model, view and projection matrices when ren-

dered. Since they need to cover the background as a whole, a high scale value is applied to the model matrix, which is handled by the functions available in the OpenGL Mathematics (GLM) library. Both skybox and skysphere options are accessible when running the program, and the user can choose whichever one is desired. They had been both implemented so the comparison between two different approaches could be possible.

4 EXPERIMENTAL RESULTS

In the previous chapter we explained the methods implemented for both texture synthesis and the rendering of a human organ. In this chapter, we present our results together with an exploration of possible parameters and inputs for the many steps of our solution.

We start by examining the videos from our database and selecting the best frames. Then, we obtain the camera parameters from a testing environment and proceed to adjust the results as the program extracts important data from the images, resulting on the translation values. The implementation uses these values to built a mosaic and, after processing the images we obtain a final texture than can be used in the rendering step. Again, we present and ponder different options to render the simulation.

4.1 Selection of Frames from Video

As described in the last chapter, the videos that compose our local database of recorded laparoscopic surgeries include many segments of hardly recognizable tissues and many frames with unwanted artifacts. This way, the process of selecting clips from the videos was entirely visual based on each individual frame.

After finding a satisfactory portion of a video, we still had to find suitable frames for the algorithm. In Fig. 4.1, we present six images from a real liver that we chose to run our tests with. The wanted frames ideally have a clear view of the main organ, and need to present a translation of the observer between one another. These requirements inflict on another important decision, as the number of input images can interfere with the construction of the mosaic and, therefore, on the final texture.

In Fig. 4.2, its is possible to observe the results of both six and eight images as inputs. The selected frames were from the same video clip and show the same organ from different points of view. However, as adding a pair of images interferes on the performance, we decided to run the program preferably with six frames as input.

Figure 4.1: Six frames of a human liver selected from a surgery video, showing different points of view from the same scene.

Source: Local database.

4.2 Calibrating a Camera

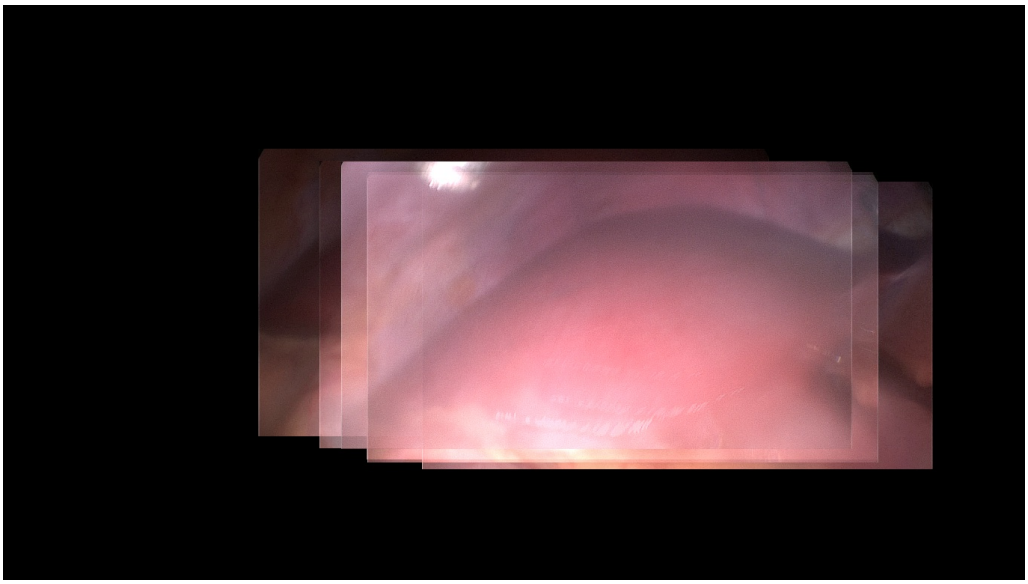
As explained before, a implementation of a camera calibrator written in C++ is part of this project. This program is independent of the main implementation and must be executed before the texture synthesizer and renderer every time the camera model used to record the frames changes. This way, the main application runs with updated intrinsic camera parameters and is able to present more accurate results.

Our main idea here was to calibrate the camera of a real laparoscope, and we contacted a doctor that performs laparoscopic surgeries to conduct the experiment. However, due to scheduling issues and restrictions to use the equipment, we were unable to achieve our goal of using the real medical tool. So, as a way to validate the results from the program, we present in this section a calibration experiment made with a regular domestic camera present in a smartphone available in the market.

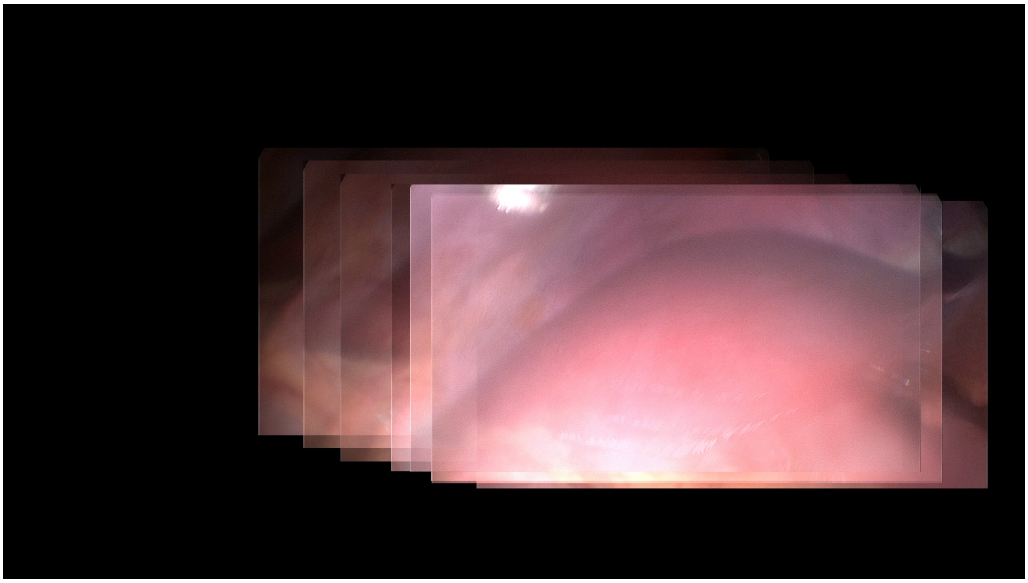
The first step of the calibration process is to print a chessboard pattern. In this case, we used a 9x6 pattern, which means there are nine rectangles of width and six of height. This value is arbitrary, and the built calibrator accepts different pattern sizes, since this value is informed as an input before the program execution. Then, it is necessary to take a reasonable number of pictures from the pattern, in different angles and positions. In Fig. 4.3, the ten images we used to calibrate the camera are presented.

The calibrator then outputs the camera intrinsic parameters matrix in the exact format we described before, which is shown in Equation 3.1. Running the program with

Figure 4.2: Image mosaicing results using different number of images as input.
(a) 6 images

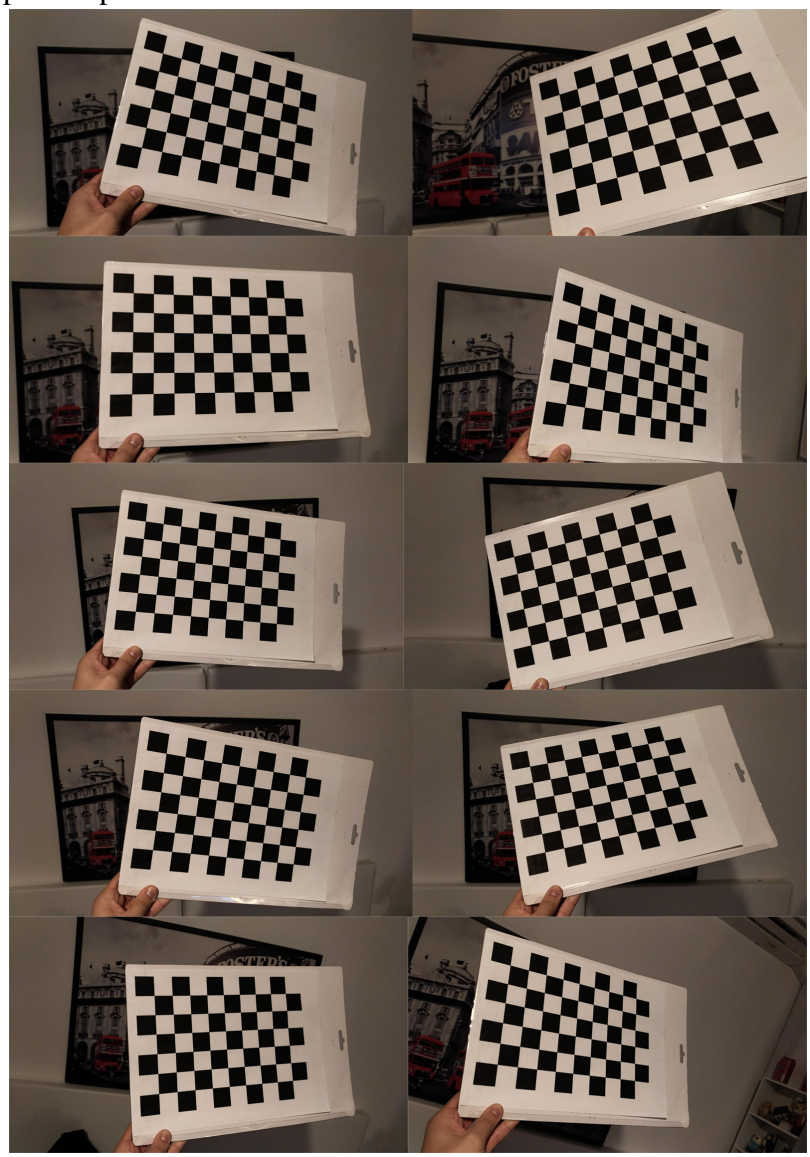


(b) 8 images



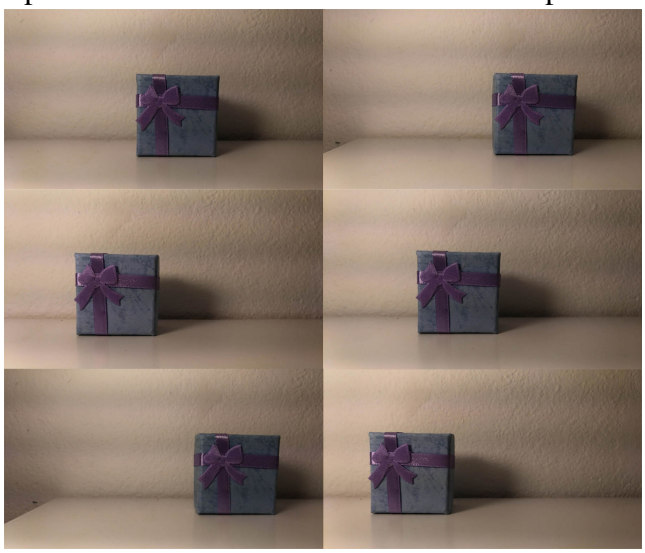
Source: Local database.

Figure 4.3: Pictures taken of a 9x6 chessboard pattern to perform camera calibration, showing the pattern positioned in different orientations.



Source: Local database.

Figure 4.4: Six pictures from the same scene taken with a regular smartphone camera. They were used as input to test the results of the calibrated implementation.



Source: Local database.

all the ten images presented in Fig. 4.3, the output is the following matrix

$$\begin{bmatrix} 3650 & 0 & 2357.9 \\ 0 & 3673.4 & 1273 \\ 0 & 0 & 1 \end{bmatrix}$$

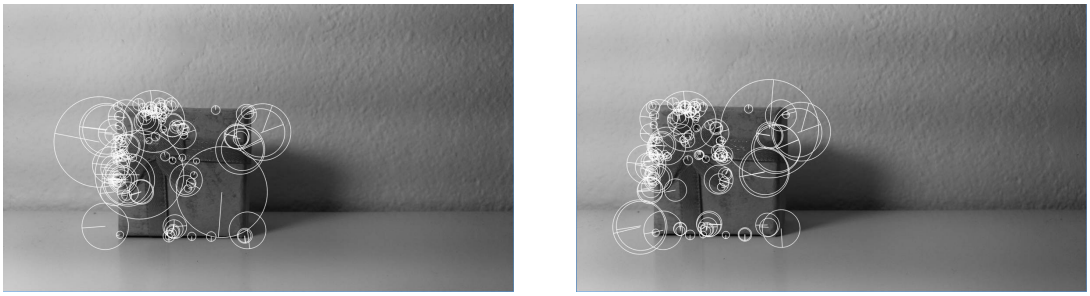
4.3 Choosing a Hessian Threshold and Finding Keypoint Matches

One of the main parameters for the SURF function in OpenCV to correctly detect keypoints is the minimum Hessian threshold. This value is basically responsible for the level of distinctiveness a point has to have to be chosen as a keypoint. In our tests, we experimented with very different threshold values, and the results were divergent. It is important to clarify that, for these tests based on the intrinsic camera parameters, we used photos taken with the same regular camera mentioned before. Therefore, they are not medical pictures and are present in this work just for validation purposes. Six of the pictures used for these experiments can be observed in Fig. 4.4.

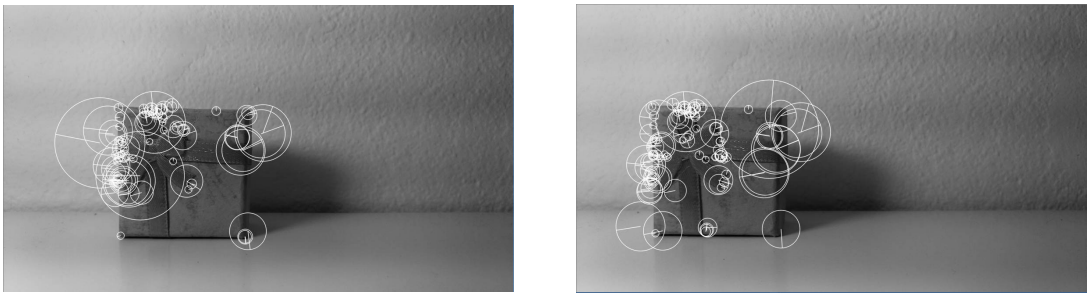
In Fig. 4.5, it is possible to see the results of the keypoints chosen for each image with different Hessian threshold values. The circles around each point in the images represent the neighborhood area the algorithm has informations about. There is a clear discrepancy between each run, and this interferes later on the construction of the mosaic. For example, some circles around the selected points are present in the images with a Hessian

Figure 4.5: SURF keypoints selection with different Hessian threshold configurations. The selected points are shown inside the circles.

(a) Image 1 with Hessian 600 (b) Image 2 with Hessian 600

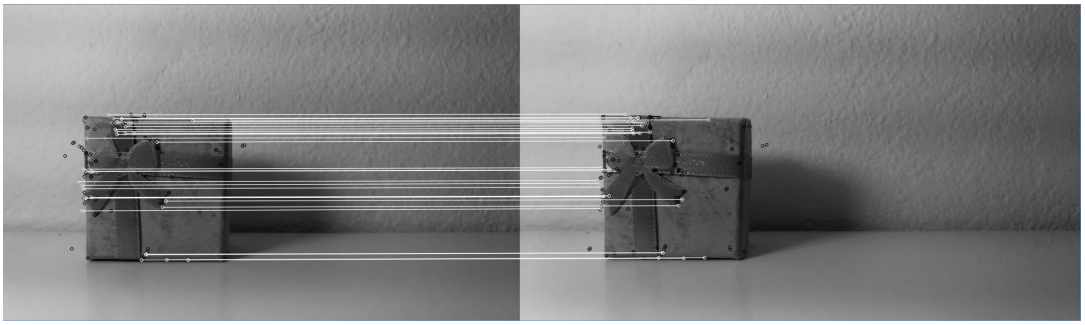


(c) Image 1 with Hessian 785 (d) Image 2 with Hessian 785



Source: Local database.

Figure 4.6: Matched pairs of selected points in a stereo image example and their distances, represented by the lines.



Source: Local database.

threshold of 600, but not in the ones with 785. This is expected because, as mentioned before, a higher threshold inflicts on fewer outputted points. The brute-force matcher then examines the keypoints and their descriptors to find the best matches between pairs of points. Fig. 4.6 shows an example of the selected matches and the processed distance for each pair of points from a pair of images.

4.4 Extracting the Fundamental Matrix

To extract the Fundamental Matrix and then finally find the Essential Matrix of both images, our implementation executes an OpenCV function. This function expects parameters such as the arrays of corresponding points from each image and the specification of the chosen method to compute the matrix, which is, in this case, RANSAC. In Chapter 3, we explained that this function basically solves the main property of the Fundamental Matrix for some random amount of pairs, because of the RANSAC parameter.

The results we obtained vary by pairs of images, but an example of the obtained values is shown below

$$F = \begin{bmatrix} -2.569463449181994e^{-7} & -2.975056774109661e^{-5} & 0.01324578091234546 \\ 3.029177635432596e^{-5} & -3.889709532032659e^{-7} & -0.03095817890928954 \\ -0.0130081632703419 & 0.02834225743374488 & 1 \end{bmatrix}$$

Using the definition that says it is possible to find the Essential Matrix by multiplying the Fundamental Matrix, the camera intrinsic parameters matrix and its transposed version, we can then discover E. An example of the Essential Matrix calculated after the Fundamental Matrix values presented before is the following

$$E = \begin{bmatrix} -3.42334545273651 & -398.9032922771125 & -92.1066125060523 \\ 406.1599570743928 & -5.248722527211828 & 146.8317528921705 \\ 91.06595476733291 & -155.3913796453423 & -2.204291861422835 \end{bmatrix}$$

4.5 Obtaining Translation Values

Having the Essential Matrix, the translation and rotation values between both images is just a step away. This step is the singular-value decomposition (SVD), which is responsible for decomposing the Essential Matrix into the values the program expects. This approach, however, created two main difficulties for our work.

The first one is that the translation values are unitary and independent of a real scale, as expressed by (TRON; DANIILIDIS, 2017). The solution we found for this problem consists of knowing the size of the main real object in the metric system, and

then finding a correspondence on the distance between two 3D points inside the image. This way, we were able to scale the translation and then position the images correctly.

The second one is that the decomposition of the Essential Matrix can result on four possible values that satisfy the solution. The rotation found can be either R1 or R2, but only one of them being the correct value, while the translation t can be either positive or negative, which means the number magnitude, ignoring the sign, is final. Regarding this problem, we decided to always use the first translation solution presented by the decomposition. The rotation, however, is not used during the construction of the mosaic due to this uncertainty of the correct value. This decision does not necessarily inflict on wrong mosaic results because the translation between images still positions corresponding points on top of each other inside the mosaic, projected on the same plane, as it can be observed on the results we present later.

The values we obtained after the SVD, for the same Essential Matrix shown before, are presented below

$$U = \begin{bmatrix} -0.2804595372457954 & 0.8927170286454408 & 0.3527020764531015 \\ -0.9068159011793031 & -0.366889494264278 & 0.2075500430423245 \\ -0.3146861441758164 & 0.2616264622796354 & -0.9124276546109286 \end{bmatrix}$$

$$VT = \begin{bmatrix} -0.895421977138919 & 0.3742941785516412 & -0.2410878486340214 \\ -0.2934294029337587 & -0.9033894264865694 & -0.3127086976823722 \\ -0.3348412584353464 & -0.209263976868021 & 0.9187436637249879 \end{bmatrix}$$

$$W = \begin{bmatrix} 442.2600761404345 \\ 437.0613004691814 \\ 9.848823674209765e^{-14} \end{bmatrix}$$

Finally, it is possible to obtain both rotation and translation values, which are, for this case,

$$R = \begin{bmatrix} -0.9997527285087098 & 0.006966767242554558 & 0.02111743340411448 \\ -0.007061849708865572 & -0.9999652461763129 & -0.004431333685863598 \\ -0.02108582742219356 & 0.00457936608431142 & -0.9997671815418764 \end{bmatrix}$$

$$T = \begin{bmatrix} 0.3527020764531015 \\ 0.2075500430423245 \\ -0.9124276546109286 \end{bmatrix}$$

It is important to specify that all these results are specific to one pair of images. The values shown above are all extracted from the same pair and are obtained sequentially. They are present in this section just for demonstrative purposes.

4.6 Building the Image Mosaic

The whole process described in the previous sections is repeated for each pair of images, since each stereo imaging situation has different Fundamental and Essential Matrices. The iteration starts with the first inputted image and goes on for each subsequent pair of pictures. Each translation obtained by the program is stored in a list, which is then accessed when composing the mosaic by using rendering with OpenGL functions.

Each image is a rectangular object with a texture applied to its surface. These planes have fixed size to fit inside the rendered window, and they are scaled and positioned, according to their obtained translations, inside the $[-1, 1]$ interval used by OpenGL to delimit the window boundaries. The main idea here is to overlap points that correspond inside different images to broaden the background. The rendered result is captured and stored inside a new image, which can be observed in Fig. 4.7. This outcome, however, is not the final texture, as it still needs to be post-processed.

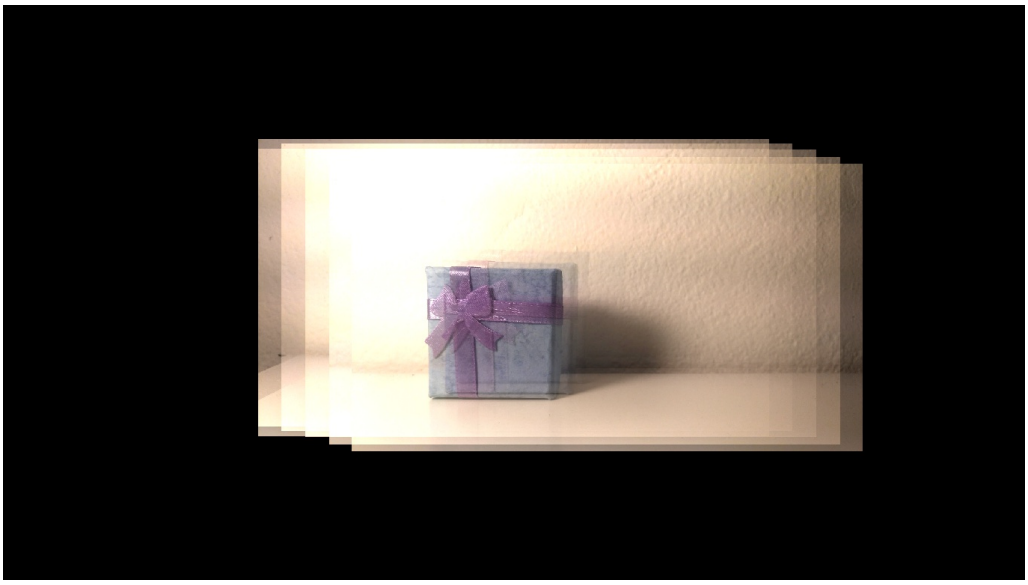
4.7 Processing the Synthesized Texture

As described in Chapter 3, there are three post-processing techniques applied to the obtained texture after the composition of the mosaic. One of them is not post-processed, but actually applied during the image mosaicing, which is alpha-blending. Therefore, this effect is already seen in Fig. 4.7.

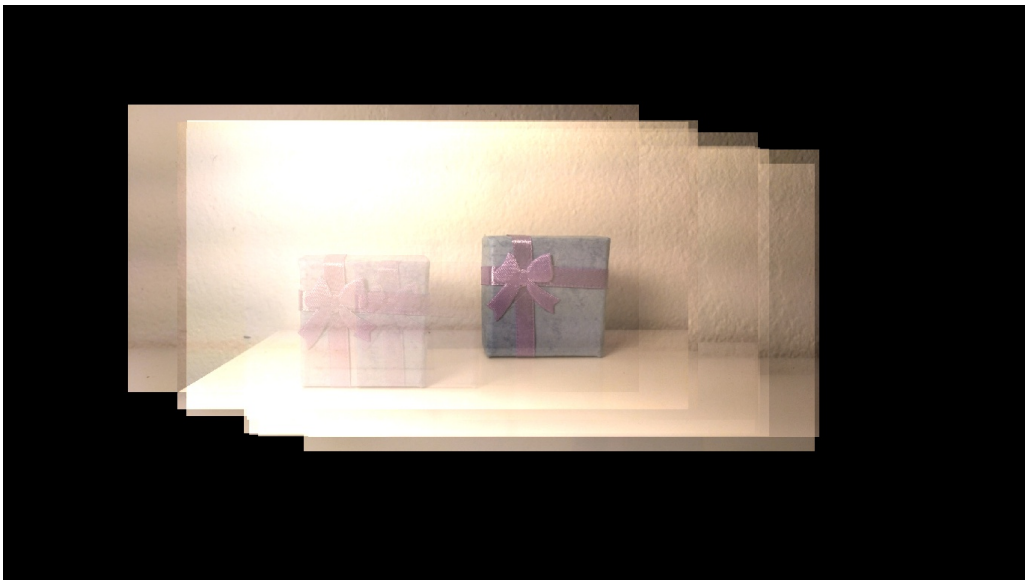
The second one is the margin reduction technique, which is responsible for reducing the rectangular size of the obtained image by enclosing the pixels with color information until the minimum window is achieved. This method, however, still leaves black holes in the processed image due to the uneven translation of each photo, so it is necessary

Figure 4.7: Image mosaicing experimental results using different number of images as input. They take into consideration the camera parameters obtained by the calibration of a regular smartphone camera.

(a) 6 images



(b) 10 images



Source: Local database.

Figure 4.8: Mosaic after the margin reduction processing, which eliminates the black margins outside the minimum rectangular window possible.

Source: Local database.

to fill these pixels with texture synthesis, as it can observed in Fig. 4.8.

The last post-processing method is the inpainting technique, which is used to fill missing spaces of real pictures. In our case, we chose to apply the algorithm described by Telea, which is available in the OpenCV library as a function. The program creates an image mask based on the black pixels identification and then passes this image, having the same size as the original reduced texture, as a parameter. The result can be observed in Fig. 4.9.

4.8 Rendering the Organ

For this task, the images used for testing the program were extracted from one of the laparoscopic surgery videos included in the database. Unfortunately, we were not able to calibrate a real laparoscope camera, so we could use its intrinsic camera parameters. Instead, just for experimental results, the program was provided with the same parameters obtained in the sections above, from a regular camera. The final texture synthesized from the images can be seen in Fig. 4.10c.

We chose to render a liver because we had a clear cut from one of the videos of a liver. This way, we loaded a liver model on the built renderer, alongside its base texture, which is shown in Fig. 4.11b. The simulator drawns the organ with its main texture before the skybox, using the camera view as reference. For this scenario, we positioned

Figure 4.9: Texture without holes, filled by the inpainting technique.

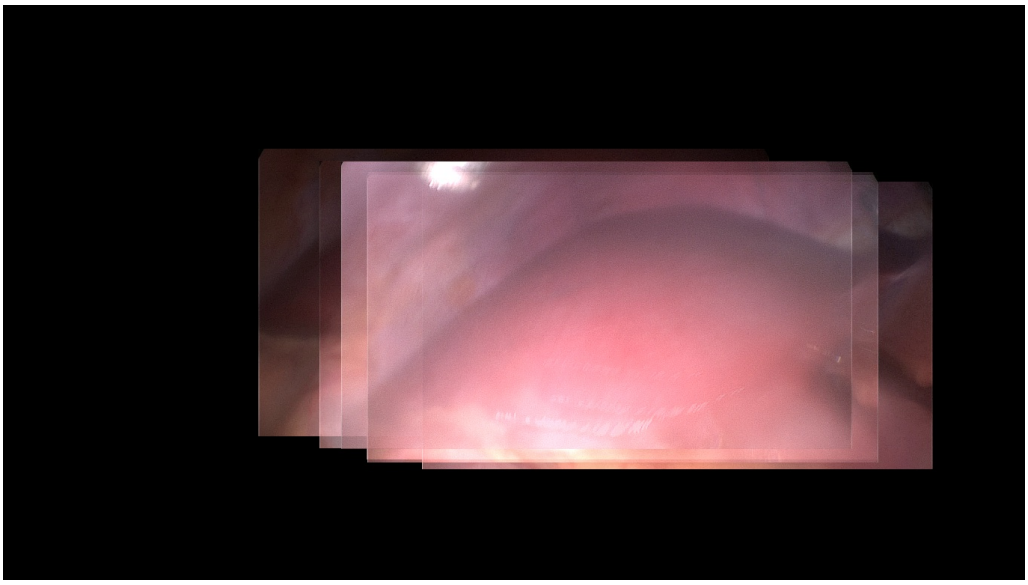
Source: Local database.

one light source located above the object. All light contributions configured in the shader, as explained in Chapter 3, are described in Table 4.1. We used this values empirically, considering that a wet organ has a strong specular aspect. These values, added to the constructed texture of the scene environment, lead the results towards a simulated Global Illumination approach.

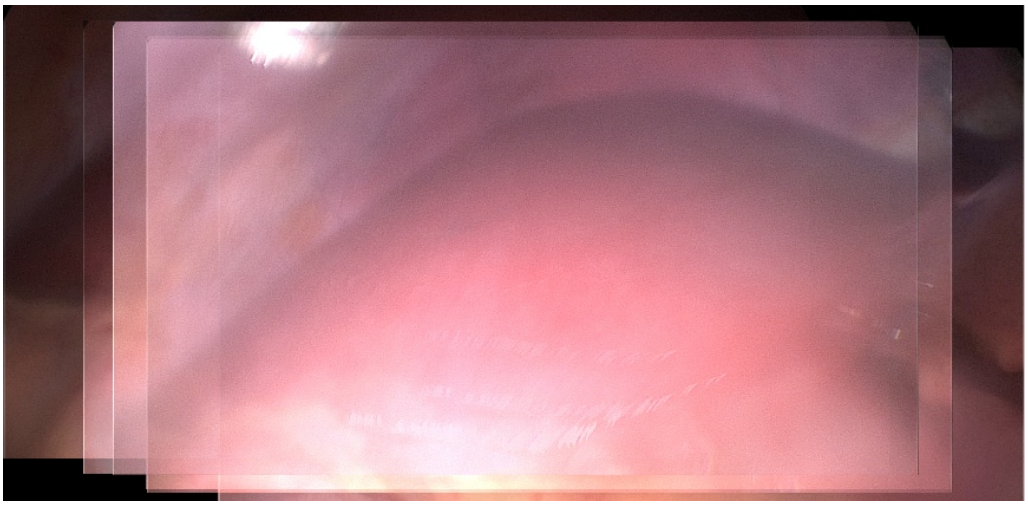
The camera view of the rendered scene, which runs in real time, including a skybox with a cubemapped texture, is reproduced in Fig. 4.11. This scene can be navigated with a moveable camera view. In this step, we are able to see the final environment mapping applied to the organ, thanks to the vertex and fragment shaders. The main part of the code used for this process can be seen in Appendix A. Both base and our synthesized texture are combined together on the surface of the liver, and a comparison between the object with and without the environment mapping texture can be observed in Fig. 4.11.

A different comparison can be observed in Fig. 4.12, where the liver object is shown from another point of view. The different faces of the skybox are visible in the background, and this interferes on the surface of the liver in Fig. 4.12c. In the images, the results with the synthesized texture show clearer colors on the liver surface, thanks to the reflection of the simulated illumination of the environment. In Fig. 4.11b, which depicts the liver without the generated texture, the light on the surface of the organ can be seen as more constant, which is a result of the absence of the interaction with the background light.

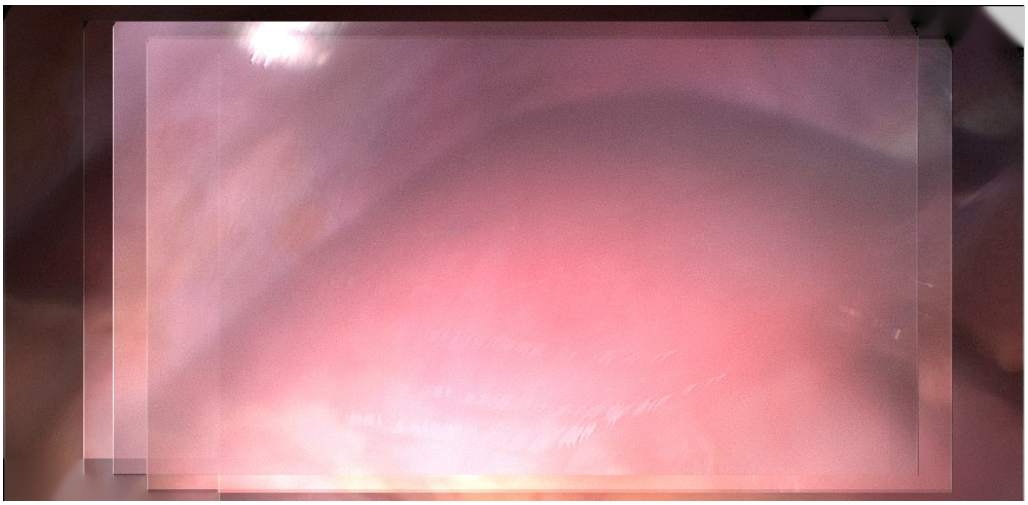
Figure 4.10: Texture synthesis steps using frames from a recorded human liver surgery.
(a) Resulting mosaic



(b) Margin reduction



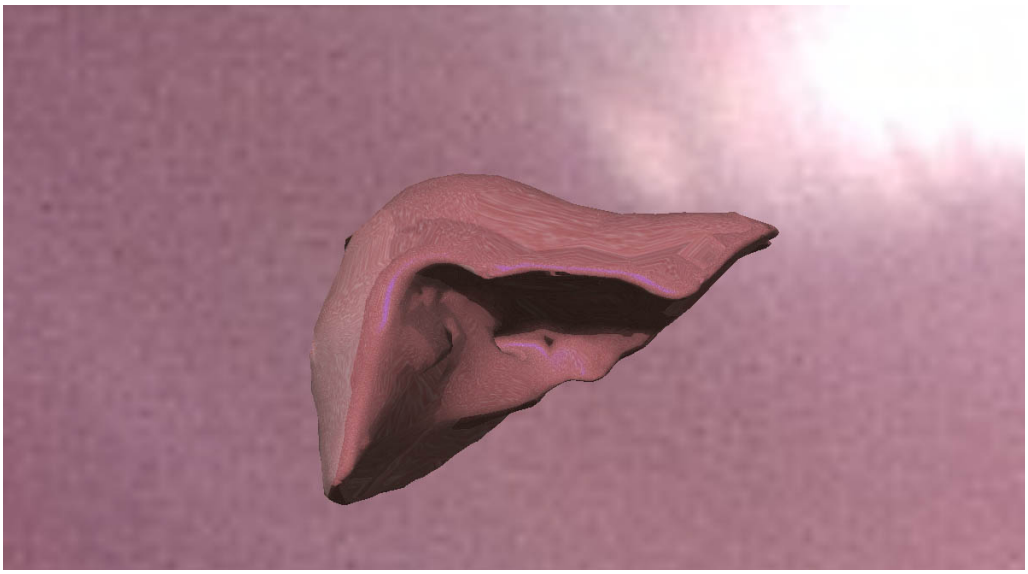
(c) Inpainted texture



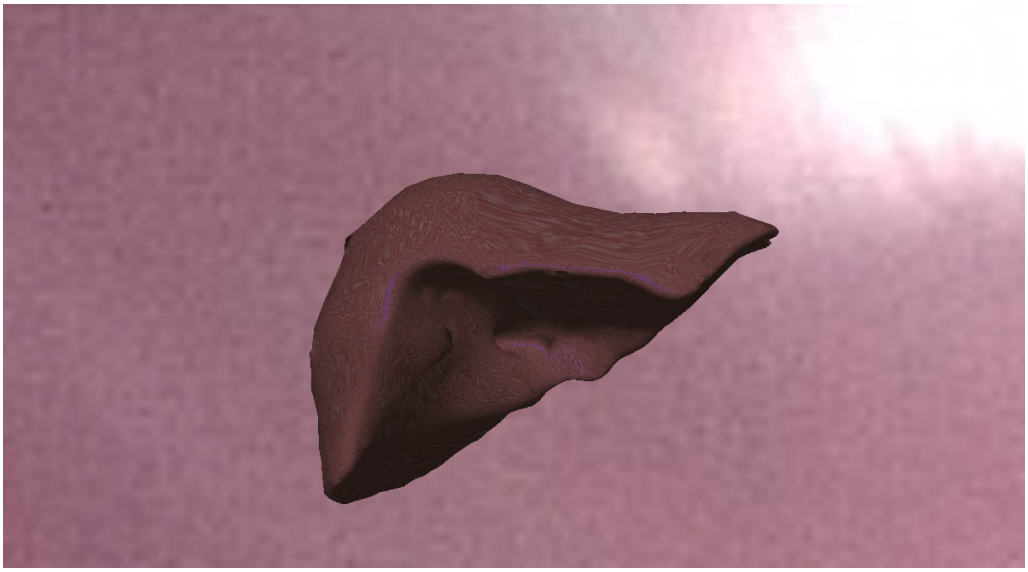
Source: Local database.

Figure 4.11: Rendering of the liver with background and varying textures, comparing the results of the generated and the base texture.

(a) Result with Environment Mapping



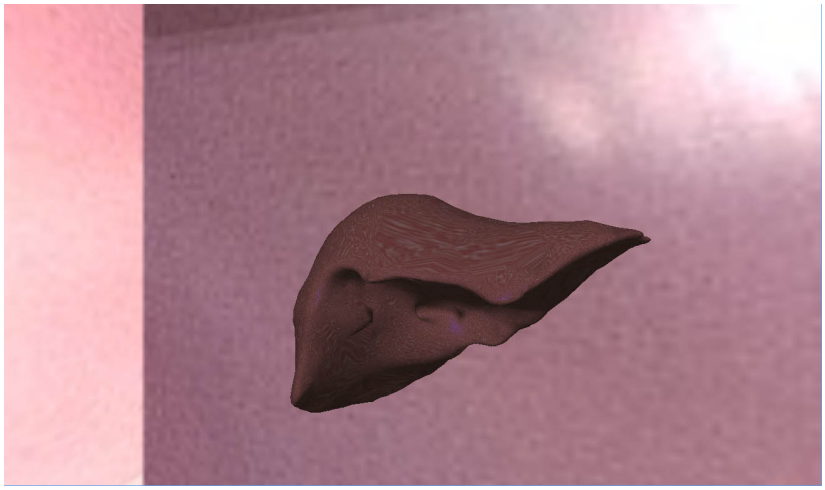
(b) Result with just the base texture



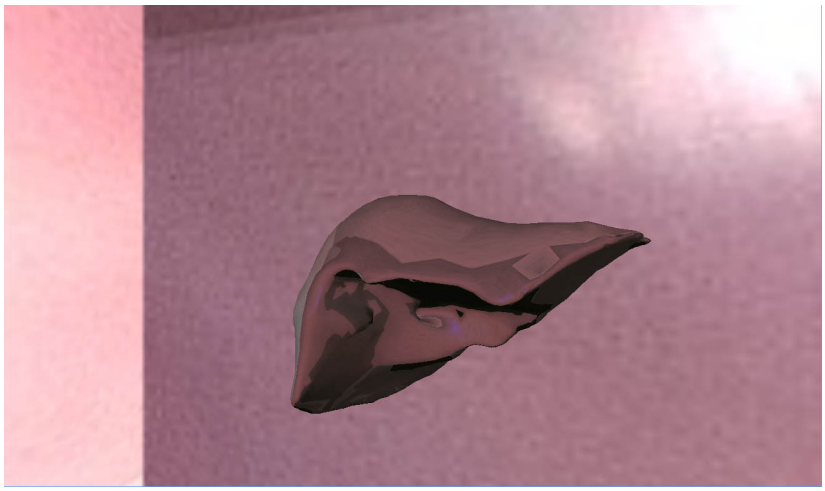
Source: Local database.

Figure 4.12: Simulation renderings of the liver with base, synthesized and a combination of both textures.

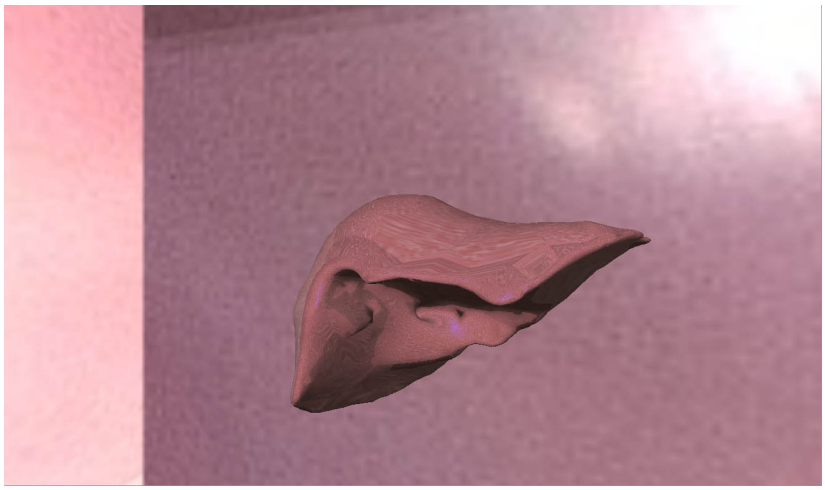
(a) With only base texture



(b) With only synthesized texture



(c) Combined result



Source: The authors.

Table 4.1: Global Illumination light contributions

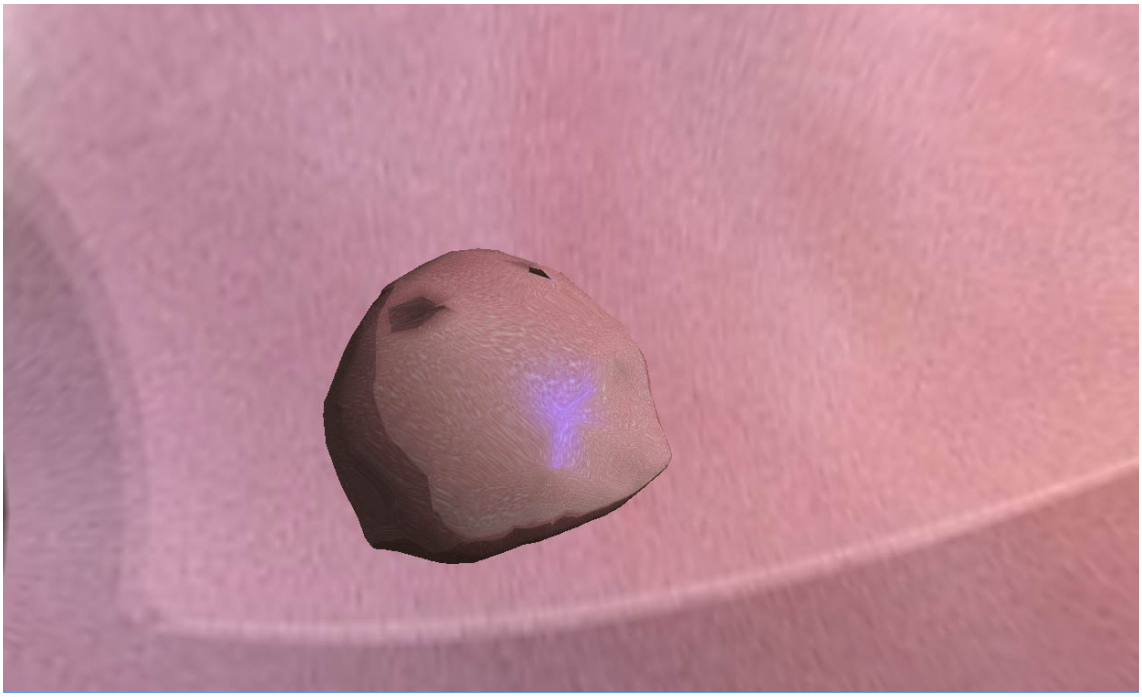
Light	*Contribution*
Emissive	0%
Ambient	20%
Diffuse	40%
Specular	40%

Source: The authors.

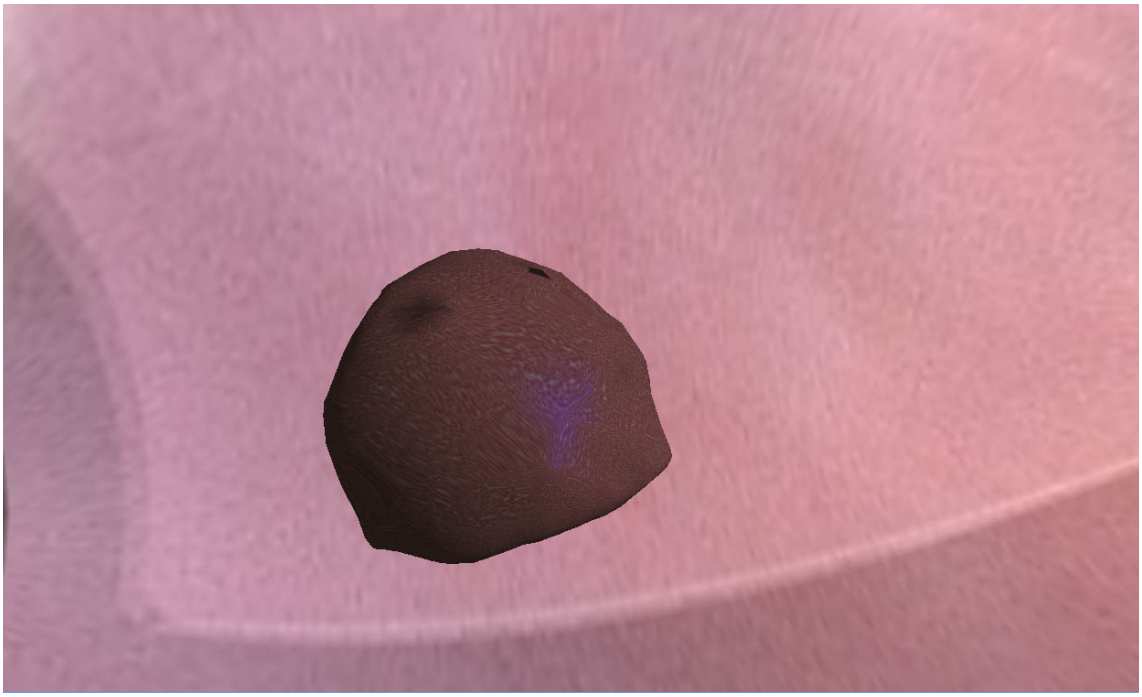
4.9 Rendering the Background

As a way to provide visual feedback from the environment surrounding the organ, we also implemented two forms of rendering the background. In a few words, they are just background objects with the synthesized textured applied to them. In Fig. 4.13, the results of the rendering with and without the synthesized textures are shown with a skydome in the background, in a different angle from the previous figures for comparison effects. In Figs. 4.12 and 4.11, the liver is rendered in a skybox environment.

Figure 4.13: Comparison between textures in a skydome scene.
(a) Liver with generated texture.



(b) Liver with only its base texture.



Source: The authors.

5 CONCLUSION

In this work, we reviewed the mathematical concepts behind a stereo imaging analysis and described the construction of both a texture synthesizer and a medical scene renderer. Lastly, we presented the results of each step of the implemented software and provided different combinations of parameter configurations and scene settings for comparison.

We started by explaining how the notions of the epipolar geometry establish the bases for the Fundamental and Essential Matrices estimation. When correlating two points from different views of the same scene by using SURF to find these matrices, the camera intrinsic parameters are an elementary piece of the solution. Thus, we detailed the implementation of a camera calibrator dependent of images of a chessboard pattern to provide accurate results of these parameters. Having these information, our program was able to find translation values between pairs of frames by decomposing the Essential Matrix through the SVD method. This values were then used to compose an image mosaic.

Before advancing to the synthesized texture mapping onto the organ object, we also proposed and examined the post-processing methods applied to the output image. Alpha blending, margin reduction and inpainting were used to improve the quality of the created texture. During the rendering phase, when the texture was mapped to the organ, we pondered the contributions of the lighting environment over the surface of the loaded model. The visualization of the effects of the environment texture on the organ was also enriched by the placement of a background, which is available as both a skydome or a skybox. All these rendering features are part of an application that runs in real time, in a current computer.

For future work, we would like to experiment with different techniques to elaborate the image mosaic, such as the stitching method (BROWN; LOWE, 2006) used to create panoramas. The result of the synthesized texture could also have its quality improved by adding more post-processing algorithms like smoothing effects, for example. The rotation data was also not used in the mosaic due to its uncertainty. If correctly validated, this information could be added to the texture to create visual depth.

Although we tried, we were not able to calibrate a real medical laparoscope, and this remains as a suggestion for future steps. More validation efforts could also be done with users, which is something we were not able to perform. Lastly, the implemented

medical simulator does not show a complex organism, having just one organ. The results could be applied to a more advanced simulator in order to have its results tested in a more realistic scenario.

As a conclusion, we satisfactorily presented a technique to simulate an environment mapping approach based only on images originated from videos of surgeries. The created results seem similar to the tissues that are present inside the body, and this is supported by the similarity of the overall appearance of the human tissue, although more validation regarding the quality of the results is needed. We also concluded that, as a simple image texture that can be utilized in different scenarios, the generated outputs are able to be applied to an existing simulator. However, above all, we introduced a synthesis method responsible for creating a texture that can be used to simulate a common effect of the Global Illumination model on a local illumination context.

REFERENCES

ADELSON, E. H.; BERGEN, J. R. The plenoptic function and the elements of early vision. **Computational Models of Visual Processing**, p. 3–20, 1997.

BAY, H. et al. Speeded-up robust features (surf). **Computer Vision and Image Understanding**, v. 110, p. 346–359, 2006.

BLINN, J. F.; NEWELL, M. E. Texture and reflection in computer generated images. **Communications of the ACM**, v. 19, p. 542–547, 1976.

BLUM, A.; HOPCROFT, J.; KANNAN, R. **Foundations of Data Science**. First. [S.l.]: Cornell University, 2018.

BROWN, M.; LOWE, D. G. Automatic panoramic image stitching using invariant features. **International Journal of Computer Vision**, v. 74, p. 59–73, 2006.

CHEN, W.-C. et al. Light field mapping: efficient representation and hardware rendering of surface light fields. **SIGGRAPH**, p. 447–456, 2002.

FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. **Communications of the ACM**, v. 24, n. 6, p. 381–395, 1981.

HARTLEY, R. I.; ZISSERMAN, A. **Multiple View Geometry in Computer Vision**. Second. [S.l.]: Cambridge University Press, ISBN: 0521540518, 2004.

HEIDRICH, W.; SEIDEL, H.-P. View-independent environment maps. **Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware**, p. 39–ff, 1998.

INAMPUDI, R. Image mosaicing. **IGARSS**, 1998.

LALONDE, J.; EFROS, A. A. **Synthesizing Environment Maps from a Single Image**. 2010.

LIM, Y.-J.; DE, W. J. S. On some recent advances in multimodal surgery simulation: A hybrid approach to surgical cutting and the use of video images for enhanced realism. **Presence: Teleoperators and Virtual Environments**, v. 16, p. 563–583, 2007.

NEYRET, F.; HEISS, R.; SéNéGAS, F. Realistic rendering of an organ surface in real-time for laparoscopic surgery simulation. **The Visual Computer**, v. 18, p. 135–149, 2002.

OLSSON, C. Lecture 6: Camera computation and the essential matrix. 2013.

OpenCV. **Open Source Computer Vision API Reference**. 2014. Available from Internet: <https://docs.opencv.org/3.0-beta/modules/refman.html>.

OpenCV. **Open Source Computer Vision documentation**. 2016. Available from Internet: <https://docs.opencv.org/3.2.0/index.html>.

OpenGL. **OpenGL Fragment Reference**. 2017. Available from Internet: <https://www.khronos.org/opengl/wiki/Fragment>.

PALMA, G. et al. Surface light field from video acquired in uncontrolled settings. **Digital Heritage International Congress**, p. 31–38, 2013.

PREIM, B.; BOTHA, C. P. **Visual computing for medicine: theory, algorithms, and applications**. [S.l.]: Newnes, 2013.

SETHIAN, J. A. A fast marching level set method for monotonically advancing fronts. **Proc. Natl. Acad. Sci. USA**, v. 93, p. 1591–1595, 1996.

STRANG, G. **Introduction to Linear Algebra**. Fifth. [S.l.]: Wellesley-Cambridge Press, ISBN: 978-09802327-7-6, 2016.

SZELISKI, R.; SHUM, H.-Y. Creating full view panoramic image mosaics and environment maps. **SIGGRAPH**, p. 251–258, 1997.

TELEA, A. An image inpainting technique based on the fast marching method. **Journal of Graphical Tools**, v. 9, n. 1, p. 25–36, 2004.

TRON, R.; DANIILIDIS, K. The space of essential matrices as a riemannian quotient manifold. **SIAM J. IMAGING SCIENCES**, v. 10, n. 3, p. 1416–1445, 2017.

VERTH, J. M. V.; BISHOP, L. M. **Essential Mathematics for Games and Interactive Applications**. Third. [S.l.]: A K Peters/CRC Press, ISBN: 1482250926, 2015.

VIDAL, F. P. et al. Principles and applications of computer graphics in medicine. **Computer Graphics Forum**, v. 25, n. 1, p. 113–137, 2006.

APPENDIX A — LIGHT CONTRIBUTIONS SHADER CODE

In this Appendix, we present the GLSL code used to calculate the different light contributions that compose the final color of the object surface. This is implemented inside the fragment shader and outputs the color value of each fragment in the rendered object.

```
#version 330

in vec3 normal_vector;
in vec3 light_vector;
in vec3 halfway_vector;
in vec3 texture_coord;
in vec2 sampler_coord;
uniform samplerCube cubemap;
uniform sampler2D organtex;
out vec4 fragColor;

void main (void) {
vec3 normal1         = normalize(normal_vector);
vec3 light_vector1   = normalize(light_vector);
vec3 halfway_vector1 = normalize(halfway_vector);

vec4 c = texture(cubemap, texture_coord);
vec4 o = texture(organtex, sampler_coord);

vec4 co = c + o;

vec4 emissive_color = vec4(0.0, 1.0, 0.0, 1.0);
vec4 ambient_color  = vec4(1.0, 1.0, 1.0, 1.0);
vec4 diffuse_color  = vec4(1.0, 1.0, 1.0, 1.0);
vec4 specular_color = vec4(0.0, 0.0, 1.0, 1.0);

float emissive_contribution = 0.0;
float ambient_contribution  = 0.20;
```

```
float diffuse_contribution  = 0.40;
float specular_contribution = 0.40;

float d = dot(normal1, light_vector1);
bool facing = d > 0.0;

fragColor = emissive_color * emissive_contribution +
    ambient_color  * ambient_contribution  * co +
    diffuse_color  * diffuse_contribution  * co * max(d, 0) +
                     (facing ?
    specular_color * specular_contribution * co *
    pow(dot(normal1, halfway_vector1), 80.0) :
    vec4(0.0, 0.0, 0.0, 0.0));
fragColor.a = 1.0;
}
```