Creating a tool for visualization and analysis of moving objects

Moseguí Saladié, Alexa

Curs 2015-2016

Director: COLOMA BALLESTER i GLORIA HARO

GRAU EN ENGINYERIA EN SISTEMES AUDIVISUALS

Treball de Fi de Grau
Creating a tool for visualization and analysis of moving objects

Alexa Moseguí Saladié

UNDERGRADUATE THESIS
Degree in Audiovisual Systems Engineering

ESCOLA SUPERIOR POLITÈCNICA UPF
Year 2015-2016

THESIS SUPERVISORS
Coloma Ballester, Gloria Haro
Acknowledgements

I want to express my most sincere gratitude to my supervisors Coloma Ballester and Gloria Haro for giving me the opportunity to stay at the GPI, where I have learned a lot, and for their guidance, patience, motivation and advising through this project. I also would like to thank Roberto Palomares and Pablo Arias for their time and patience, answering all of my questions and bearing with all of my crisis. Moreover, thanks to Maria Oliver for letting me use her segmented images in my project.

I owe a huge thank you to my family and friends for their unconditional support and for giving me the strength to keep on, I could not have done it without you.

Besides, I would like to thank Aina, Anna and Laia for walking with me during these five years and for staying together through all the nerves and stress of these past months.

Finally, special thanks to him, Christian, who have backed and advised me through this long journey. All this would have been much harder without you.
Abstract

In recent years, we have witnessed a significant increase in the use of technologies related to video manipulation. Thus, the analysis and automatic interpretation of a dynamic three-dimensional scene from a single video has become a primary objective within the computer vision community both for its scientific interest and its multitude of technological and social applications. Researchers need a powerful, but user-friendly, application to visualize and analyse their results, such as, tracking people or vehicles, detection of situations, automatic detection of objects, and so on.

This project develops a tool for visualizing these results in a 3D environment and checking the accuracy of the detection and tracking. Using a wide range of computer vision techniques, the video scene will be projected as a set of spatiotemporal tubes in 3D where the user will be able to interact with them and edit some features. The aim of this application is to ease the visualization of the video scene from all the possible angles and points of view.

Resum

En els últims anys, hem assistit a un increment important en l’ús de les tecnologies relacionades amb la manipulació de vídeo. Així doncs, el anàlisis i la interpretació automàtica d’una escena dinàmica tridimensional a partir d’un sol vídeo s’ha convertit en un objectiu primordial dins la comunitat de visió per ordinador, tan pel seu interès científic com per les moltes aplicacions tecnològiques-socials. Els investigadors necessiten una aplicació potent, però a la vegada intuïtiva, per visualitzar i analitzar els seus resultats provinents del seguiment de persones o vehicles, de deteccions de situacions, de deteccions automàtiques d’objectes, entre d’altres.

En aquest projecte hem desenvolupat una eina per la visualització en un entorn 3D d’aquests resultats. A més a més, l’usuari també podrà comprovar la precisió obtinguda en les deteccions i el seguiment dels objectes. Utilitzant un ampli ventall de tècniques en visió per ordinador, l’escena del vídeo serà projectada en un conjunt de tubs espaiotemporals 3D on l’usuari podrà interactuar amb l’entorn i editar algunes característiques determinades. Per tant, l’objectiu d’aquesta aplicació és facilitar la visualització d’una escena de vídeo en tots els angles i punts de vista possibles.
# Contents

## 1. INTRODUCTION
1.1 Motivation ........................................................................................................... 1  
1.2 Context .................................................................................................................. 2  
1.3 Related work ......................................................................................................... 2  
1.4 Goals ..................................................................................................................... 3  

## 2. STATE OF THE ART
2.1 3D visualization software ...................................................................................... 5  
2.1.1 Amira & Avizo .................................................................................................. 5  
2.1.2 Mistika ............................................................................................................ 6  
2.1.3 Mocha ............................................................................................................. 7  
2.2 Visualization algorithms ......................................................................................... 8  
2.2.1 Spatiotemporal video segmentation with long-range motion cues ................. 8  
2.2.2 Visualization of video motion in context of video browsing ......................... 8  
2.2.3 Viz-a-vis: Toward visualizing video through computer vision ...................... 9  
2.3 3D computer graphics software ........................................................................... 10  
2.3.1 Blender ............................................................................................................ 10  
2.3.2 VTK ............................................................................................................... 11  
2.3.3 Paraview ........................................................................................................ 12  
2.3.4 MedInria ....................................................................................................... 13  

## 3. ALGORITHM DESCRIPTION
3.1 Overview of our algorithm .................................................................................... 15  
3.2 Adapting frames .................................................................................................... 16  
3.2.1 Convert to PNG ............................................................................................... 16  
3.2.2 Convert to Grayscale ...................................................................................... 17  
3.2.3 Changing the background colour ..................................................................... 18  
3.3 Object detection .................................................................................................... 19  
3.3.1 Image histogram ............................................................................................. 19  
3.3.2 Image thresholding ......................................................................................... 22  
3.4 Isosurface generation ............................................................................................ 24  
3.4.1 Isosurface ....................................................................................................... 24  
3.4.2 Mapping ......................................................................................................... 25  
3.4.3 Actors ............................................................................................................. 25  
3.5 World space projection ......................................................................................... 25  
3.5.1 Overview of rendering ................................................................................... 25  
3.5.2 Lights .............................................................................................................. 27  
3.5.3 Cameras ......................................................................................................... 27  
3.5.4 Coordinate Systems ....................................................................................... 29  

## 4. PROPOSED APPLICATION
4.1 App definition ....................................................................................................... 34  
4.1.1 Main idea ......................................................................................................... 34  
4.1.2 Used technologies .......................................................................................... 35  
4.2 Design and features ............................................................................................. 35  
4.2.1 Input data ........................................................................................................ 35  
4.2.2 Main window ................................................................................................... 36  
4.2.3 Upload original frames .................................................................................... 36  
4.2.4 Surface properties ......................................................................................... 37  
4.2.5 Video discretization ...................................................................................... 37  
4.2.6 Object visualization ....................................................................................... 37  

vii
4.2.7 Frame visualization ................................................................. 37
4.2.8 Screenshot .............................................................................. 38
4.3 Requirements ............................................................................. 38
4.3.1 Functional ................................................................................ 38
4.3.2 Non-functional ................................................................. 40
4.4 Mock-up ..................................................................................... 41
4.4.1 Interface design ..................................................................... 41
4.4.2 Layout scheme ......................................................................... 46
4.4.3 Final design .............................................................................. 49
4.5 Class diagram ............................................................................ 52

5. IMPLEMENTATION ................................................................. 55
5.1 VTK ....................................................................................... 55
5.2 Application pipeline .................................................................... 57
5.2.1 Frame processing .................................................................... 57
5.2.2 Object identification and isolation ........................................... 59
5.2.3 Volume generation .............................................................. 60
5.2.4 Volume displaying ............................................................. 61
5.2.5 Editable visualization features ............................................... 62

6. EVALUATION AND RESULTS ................................................. 69
6.1 Visualization of the results ........................................................ 69
6.2 Evaluation .................................................................................. 79
6.3 Limitations ................................................................................ 81

7. CONCLUSION ............................................................................ 83
7.1 Future work development .......................................................... 83

Appendices .................................................................................. 85

A. MARCHING CUBES ................................................................. 87
Chapter 1

INTRODUCTION

1.1 Motivation

In recent years, we have witnessed a significant increase in the use of technologies related to video understanding, editing and manipulation. Thus, the analysis and automatic interpretation of a dynamic three-dimensional scene from a video has become a primary objective within the computer vision community, both for its scientific and its multitude of technological and social applications. Some of these applications are tracking people or vehicles, detection of specific situations or automatic interpretation of the changes occurred in a scene.

On the other hand, automatic editing of a video is another great challenge of computer vision and has direct applications in the post-production industry and visual effects which usually use software packages that require user intervention. Some of these applications are inserting or deleting objects, improving video quality and creating new views of a scene.

Segmentation of a video into different objects is a key tool for the vast majority of the applications mentioned, usually accomplished at the first stages of them. Video segmentation can be defined as the process of grouping pixels of video in homogeneous spatiotemporal regions (usually called spatiotemporal tubes) exhibiting consistency both with respect to their appearance as their movement. These spatiotemporal tubes are aimed to correspond to objects moving in the real scene. Another basic element in many applications is the automatic completion of shapes, moving or not. If the problem refers to the completion of 2D, 3D shapes or spatiotemporal tubes, it is technically called binary inpainting or geometry-based inpainting. If the problem includes completing or filling-in texture information, it is usually called inpainting (also texture-based inpainting).

Consequently, visualization is an important part of computer vision. The researchers need powerful tools to test their investigations. Therefore, due to our human condition, the visual representations are easier to work with because we have really valuable visual abilities and we can even infer some structural 3D information from 2D planar images. However, the world around us is in 3D, all our environment is in 3D, and, in consequence, we understand better the interactions when can be visualized in 3D.

Nowadays, researchers frequently analyse their results watching their video segmentations frame by frame. While analysing individual frames independently might provide some cues, video provides more important information such as motion of objects in the scene, temporal continuity (with challenges such us temporally coherent results) and temporal object interactions.
Motivated by all these reasons, we are going to create an application for visualization and analysis of moving objects in a 3D environment.

1.2 Context

This project is part of a broader goal which aims to analyse a moving scene. The group of Image Processing and Computer Vision from UPF is working on the development of automatic segmentation tools and video inpainting techniques. Both problems work with objects or shapes which interact in space and time. The three-dimensional visualization of different objects and their interactions is a key to both validation of proposed algorithms as to simply and interpret a moving scene.

![Figure 1.1: Automatic object segmentation process.](image)

In the above diagram can be seen the automatic object segmentation process. The last step of this process is the visualization which is the section that we are going to focus on.

1.3 Related work

Although, as we have seen, visualization of moving objects is a decisive factor in computer vision, there are not open source and powerful visualizers nowadays. However, there are many papers devoted to visualization or the reconstruction of a moving scene. There are a wide range of techniques, from the generation of long-range motion cues [1], to the visualization of motion direction and motion intensity statistics by colour and brightness variations [2] or video visualization through computer vision [3].

There are also, some under license, but powerful, applications which aim to ease the visualization of 3D objects. Two of these applications are Amira [4] and Avizo [5] from FEI Company which permit the visualization and manipulation of 3D data. Moreover, SGO Company has created Mistika visualizer [6] which is focused on film post-production environments. Finally, Imagineer Systems has developed a tracking software called Mocha [7] which consist in a set of image tools.

In chapter 2, we are going to review all these techniques.
1.4 Goals

This project aims at creating a tool for visualization and analysis of a moving scene. More specifically, we will develop an application that will generate a 3D reconstruction of segmented video frames, also allowing to incorporate texture information. Figure 1.2 displays an example where a selected spatiotemporal tube is shown in green while the background is illustrated with the texture content of the original video frame. Using a wide range of computer vision techniques, the video scene will be projected as a set of spatiotemporal tubes in 3D where the user will be able to interact with them and edit some features. The objectives of this project are:

• To create a simple, intuitive and powerful application.
• To ease the visualization of a moving scene.
• To create a tool for the evaluation of video segmentations.
• To create an application enabling interact with the 3D frames reconstruction.
• To analyse the interactions between objects in a video.
• To evaluate the results of binary video inpainting.

The final application should be able to create a 3D frames reconstruction like this one:

![Figure 1.2: Example of 3D frames reconstruction result.](image-url)
Chapter 2

STATE OF THE ART

This chapter analyses different existing 3D software commercially available. Furthermore, visualization algorithms and software to create our application are reported as well. We describe their main idea and some illustrative examples of visualization results.

However, it is important to emphasize that there is no visualization app that does the same as ours. There are some applications that are able to reconstruct segmented frames but they have not the same characteristics nor are not as intuitive or simple as our tool and, additionally, they are not open source.

2.1 3D visualization software

2.1.1 Amira & Avizo

Amira [4] and Avizo [5] are 3D software created by FEI, a company that works with microscopy workflows providing images and answers in the low metric system (micro, nano and picometer scales).

Amira is focused on medical image and allows visualizing, manipulating and understanding data from all types of medical tests, such as MRI (magnetic resonance imaging), microscopy or tomography. It includes a wide range of image and video processing techniques to visualize microscopic details in medical images.

It can create a 3D reconstruction from a set of 2D images. In the example below it reconstructs part of a spine from MRI images and separates each bone.

![Figure 2.1: 3D reconstruction from 2D MRI images [4].]
On the other hand, Avizo is used for analysing scientific and industrial images. It covers fields as different as geoscience, semiconductors, minerals and mining or industrial manufacturing. One of its main strengths is industrial quality control. Taking several industrial pieces, Avizo is used for comparing these parts and find defects inside.

![Comparing two industrial parts with Avizo to find defects inside](image)

**Figure 2.2:** Comparing two industrial parts with Avizo to find defects inside [5].

As commercial products both software require the purchase of a license or an academic subscription. They are used by researchers and engineers in academia and industry around the world. Although not everyone can afford the license, they are a powerful tool for processing and analyzing image data.

### 2.1.2 Mistika

Mistika software [6] is focused on film post-production environments. It is a powerful visualizer with a wide range of edition tools such as colour grading, visual effects (VFX), audio S3D, full finishing, among others.

It is developed by SGO, a European developer founded in 1993, which creates and develops post-production software for industries. SGO describes Mistika as one system total solution. It aims to find all kind of solutions for film post-production environment.

Mistika is designed to be used by post-production and filmmakers professionals. Its interface is produced to augment its potential with a pen and a tablet and specific mouse and keyboard. The usual working panel can be seen in Figure 2.3.

As powerful as this tool can be, it is not thought to be used by researchers and has a non-intuitive interface for them. Thus, this is not a useful application for our users.
2.1.3 Mocha

Mocha [7] is a tracking software developed by Engineering Systems. As explained in their official web page: ‘Built upon a renowned Planar Tracking image engine, Mocha follows ‘pixel patterns’ trough the most difficult conditions including objects that go off-screen, objects that are partially obscured, even out of focus footage’.

As in Mistika case, Mocha is aimed to artists and filmmakers, therefore has specific tools for this kinds of industries. It has been used for well-known film productions like Harry Potter or Game of Thrones.

This software requires the purchase of a license in order to use them, so it is not open source. By the fact that has so many tools like tracking graphics or 3D camera tracking and animation it is not aimed at researchers. The main aim of this tool is not 3D reconstruction of frames, thus it will be difficult for our users to use this application for that purpose.

An example of Mocha result can be observe in the following example:
2.2 Visualization algorithms

Apart from visualization software there are also several papers in the related literature aimed to visualization of video frames. In this section we are going to explain the main idea of three of such papers with a visualization aim. Although the objective of these papers is not the same as ours we can get inspired by some of the methods described below.

2.2.1 Spatiotemporal video segmentation with long-range motion cues

The objective of this paper [1] is video segmentation and share goals similar to ours. It aims to capture long-range temporal interactions among objects. The resultant visualization of objects interactions will be spatiotemporal regions respecting segmented object boundaries over time.

This method uses its own spatiotemporal video segmentation algorithm which incorporates motion cues from the past and future frames. Then creates a new track clustering cost function that includes occlusions and motion similarities along the tracks. The resultant visualization of the tracking can be seen in the following example:

![Resultant visualization of video tracking][1]

2.2.2 Visualization of video motion in context of video browsing

Taking video browsing as its objective, this paper [2] uses visualization of motion direction and motion magnitude statistics by colour and brightness variations for browsing motion patterns. This method allows the users to identify scenes of interest given a prototype scene.
The motion in the videos is represented by vector motion histograms (one per frame) using a motion vector classification scheme. The histograms are transformed later into a colour coding using the HSV colour values for visualization purposes. An example (150 seconds of a ski jumping competition video) of the resultant video scheme is:

![Resultant video scheme](image)

The user can navigate interactively through the video in order to interlink high-level semantics with the visualization of low-level motion information for a specific segment.

### 2.2.3 Viz-a-vis: Toward visualizing video through computer vision

The paper [3] aims to automatically analyse the human behaviour from spatiotemporal reconstructions of video frames. This method creates a system for activity analysis in natural settings over variable periods of time.

As described in the paper, they ‘focus on bringing the semantic gap between high level human analyses and low level machine sensing’. Taking a sequence of images from an overhead video from a place of interest as an input, they analyse the motion of this sequence, create a spatial and temporal aggregation and a semantic aggregation and these results for a visualization of the activity. These visualization results can be seen in the following example:

![Viz-a-vis overview](image)
2.3 3D computer graphics software

Another important topic to discuss is the 3D computer graphics software that we are going to use to program our application. There are a lot of open source 3D computer graphics software so we need to go through all their features to decide the most appropriate one.

2.3.1 Blender

Blender [8] is free and open-source software used in 3D computer graphics for creating animated films, 3D printed models and video games, among a lot of others purposes. Apart from all the uses mentioned above, it can also be used as a Python scripter to create your own plug-ins.

It has a huge community of programmers that share their modifications of the software and its new plug-ins. The Blender Network provides support and social networking for Blender professionals. Some of the most important user communities are Blender Artists and Blender Nation which actively update tutorials and information and have excellent forums for support.

In terms of our application, this software reads video segmentations in 3D and allows visualizing 2D+time tubes. Blender also works really well with point-clouds and 3D meshes. Since it uses Python, it is possible to create plug-ins and use Python libraries such as ‘Numpy’ and ‘BMesh’ which are really useful in computer vision applications.

Furthermore, thanks to its extensive community, Blender is updated frequently. They offer support in your doubts and have plenty of examples and forums uploaded.
On the other hand, as free and open-source software, there are some services that are not implemented yet. Users create plug-ins that are important by their own projects and as a consequence, in some fields there might be lacks of them.

Blender is usually used by artists to create animated films or video games therefore it is not actually designed to be used by image processing researchers. For instance, it does not allow to enter PGM, TIFF or other flexible formats usually used in image processing and computer vision.

As previously mentioned, it is possible to import some Python libraries in Blender but not all. In fact, PIL and PILLOW, which are used in image processing for their useful functions, are not compatible with this software.

### 2.3.2 VTK

VTK (Visualization Toolkit) [9] is a set of libraries developed by Kitware consisting of 3D computer graphics, image processing and visualization based on object-oriented programming. Being free and open-source software has risen its popularity and has extended to almost every 3D field, such as medicine and mechanics.

As said in VTK official website, ‘it consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java and Python. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor and volumetric methods. Moreover, it includes advanced modelling techniques such as implicit modelling, polygon reduction, mesh smoothing, cutting, contouring and Delaunay triangulation’.

It supports GUI toolkits such as Qt and Tkinter and it is capable of executing image operations in 2D and 3D in a few code lines. However, due to its power, it is necessary to allocate memory in the PC to benefit from all their functionalities.

One of VTK’s features is that it processes the data through a pipeline. This pipeline consists of data objects and components (sources, filters and sinks).
In the following image there is an example of a VTK volume:

![VTK example of hidrogen visualization.](image)

Figure 2.12: VTK example of hidrogen visualization.

### 2.3.3 Paraview

Paraview [10] is an open-source, multi-platform data analysis and visualization application. Its main advantage versus others software is the possibility of analysing extremely large datasets using distributed memory computing resources.

This software is based on VTK so it is not surprising that it was also developed by Kitware Inc. It has a user interface written using Qt which is compatible with Mac, Windows and Linux.

![ParaView logo](image)

Figure 2.13: ParaView’s logo
Furthermore, it is integrated with Python allowing writing your own scripts to get full access to all Paraview’s large data visualization and analysis capabilities. As we can read in its website, ‘Python scripts can be played back with or without the GUI in order to create reproducible, easily customizable, and scalable visualizations’.

Although Paraview is powerful software, due to being based on VTK, this last is a better option for creating applications. Paraview is restricted to Python interface while VTK gives you access to much more features.

![Figure 2.14](image1.png)

**Figure 2.14:** ParaView’s interface with 3D particle simulation.

### 2.3.4 MedInria

MedInria [11] is a visualization software focus on medical image. It is free and open-source and it is developed within Inria, in France. As it is explained in their website [11]: ‘Trough an intuitive user interface, medInria offers from standard to cutting-edge processing functionalities for your medical images such as 2D/3D/4D image visualization, image registration, diffusion MR (Magnetic Resonance) and tractography’.

It uses VTK on the code side among others (ITK, DTK and DCMTK). It has a lot of functionalities that would be useful for our application and, although it is focused on medical image, there are similarities between our 3D frame reconstruction and 3D MR reconstruction.

![Figure 2.15](image2.png)

**Figure 2.15:** Tractography overlapped with 3D image in MPR.
Chapter 3

ALGORITHM DESCRIPTION

This chapter describes all the steps taken by our algorithm to create a 3D representation and a reconstruction of frames. Starting from the beginning, where we create a 3D image from a set of 2D frames, passing by the generation of the surface of the 3D (2D+time) objects appearing in the scene, and ending with the projection of the volume in world space.

3.1 Overview of our algorithm

The purpose of our algorithm is to create a 3D reconstruction of segmented frames of a moving scene. In particular, it is intended to allow the visualization and analysis of moving objects and their interactions with each other.

We briefly summarize the main steps of our algorithm:

Step 1: Depending on the format of the images, the algorithm will have to adapt the frames and create a 3D image to be read (see section 3.2).

Step 2: Once the image is readable, different moving objects must be detected, see section 3.3. We are going to use histogram technique as the image is already segmented and we have a colour for each object. Afterwards, a 3D image is going to be created for each moving object.

Step 3: In order to generate a 3D volume, we use an existing isosurface algorithm called Marching Cubes described in [12], explained in detail in section 3.4.

Step 4: Finally, the 3D model needs to be rendered in a scene. Furthermore, we need to add lights and cameras to the world space (see section 3.5).

To create our algorithm, we have been inspired by similar approaches from medical image processing. One of the most important methods in medical image is the reconstruction of volumetric data from MRI (Magnetic Resonance Imaging) which generates sequences of 2D slices of the human body. In this case the input data is a set of 2D images and the output is a 3D reconstruction of the body.

In our case our input data will be a set of 2D frames of a moving scene and we will reconstruct the scene, although the main difference is our result will be 2D+time.
3.2 Adapting frames

The input of our algorithm can be given in many different formats such as PGM, PNG, TIFF, JPEG, among others. Our algorithm must be compatible with a lot of formats but, on the other hand, must be simple. For this reason, we are going to convert every input frame to PNG. Afterwards, in case the segmented frames are RGB, they are going to be converted to Grayscale. Finally, a change in the background should be done if the background colour is different from black (0 value), and the reason for this is explained in section 3.4.

3.2.1 Convert to PNG

Portable Network Graphics (PNG) is an image format that supports lossless data compression. For this reason, PNG is the format that we have chosen to convert our frames. We needed an image format compatible with image libraries but without losing important information.

In examples below we can check lossless conversion:

As seen there is no quality lost in the PNG conversion.
3.2.2 Convert to Grayscale

As stated previously, when the segmented frames are coloured images they need to be converted to Grayscale.

There are three main methods for converting RGB images into Grayscale [13]:

- The **lightness** method averages the most and least prominent colour channels:

  \[ Grayscale = \frac{\max(R, G, B) + \min(R, G, B)}{2} \]

- The **average** method simply averages the RGB values:

  \[ Grayscale = \frac{R + G + B}{3} \]

- The **luminosity** method averages the RGB values but weights them taking into account the human perception. Because of our sensitivity to green colour, this will be weighted heavier than others:

  \[ Grayscale = 0.21R + 0.72G + 0.07B \]

Looking at these three methods, we decided to use luminosity method in our algorithm as it is the most accepted one. Some examples of the conversion can be seen below:

- Red (255,0,0) \(\rightarrow\)  \[ Grayscale = 0.21 \times 255 + 0.72 \times 0 + 0.07 \times 0 = 53.55 \approx 54 \]
- Green (0,255,0) \(\rightarrow\)  \[ Grayscale = 0.21 \times 0 + 0.72 \times 255 + 0.07 \times 0 = 183.6 \approx 184 \]

*Figure 3.5*: Example of an RGB to Grayscale image conversion with the luminosity method.
However, this is a temporary solution because in some cases two colours can result in the same greyscale value:

$$Grayscale = 0.21 \times 255 + 0.72 \times 255 + 0.07 \times 0 = 237.15 \approx 237$$

$$Grayscale = 0.21 \times 254 + 0.72 \times 255 + 0.07 \times 0 = 236.94 \approx 237$$

In future work we are going to change this conversion and label each colour in a new picture (see section 7.1).

### 3.2.3 Changing the background colour

The last adaptation to video frames is changing the background colour. Due to characteristics of the isosurface algorithm the background item must be the darkest one, see section 3.4.1. Thus, we are going to change the current background colour to black colour, i.e. 0 level. In case there is a black object in the segmented frames, we are going to change its colour to the previous background colour.

The example below shows the main idea of this part of the algorithm, the input frame and the resulting one:

![Figure 3.6: Input frame.](image1)

![Figure 3.7: Adapted frame.](image2)

The conversion is going to be made using a LUT (Lookup Table), specifically a CLUT (Colour Lookup Table). A CLUT is a technique used to transform a range of input colour levels into another range of colours. A palette of 256 colours is defined at the beginning of the conversion algorithm. It is initialized as a 1D array where every element will match each own index. Taking the background colour, we are going to change its CLUT element to black colour, as explained before. The same procedure is made in case there is some black object. The example below shows the CLUT creation:
3.3 Object detection

Our algorithm will analyse separately each object in the video, consequently an automatic algorithm to detect objects is needed. Our frames will correspond to segmentation results where each colour will be a different object. For this reason, we are going to use histogram techniques to know the number of items that we have in our video. Once we know that information, a 3D image is going to be created for each of them using a thresholding technique.

3.3.1 Image histogram

The histogram of an image \( f(x,y) \) plots the distribution of image pixels in terms of their gray levels \( f \). It is represented as a 1D function \( H(f) \), where the independent variable is the gray level \( f \) and the dependent variable is the number of pixels \( H \) with that level [14].

Taking a segmented frame as an example, we plot its histogram:

![Segmented frame](image-url)
As seen in the example above, each \( f \) value which has \( H(f) > 0 \), which are marked with an ellipse, corresponds to an item. Assuming that the number of items is \( N \), the number of moving objects will be \( N - I \) and the other item is the background. This means in our example we have 5 items, which correspond to 4 objects and a background.

It is necessary to recall that our algorithm works with 3D images so the previous example is just a simple case. The algorithm produces as many histograms as frames and calculates the maximum number of objects comparing all image histograms.

In our algorithm, usually the object with higher \( H(f) \) value represents the background but this may not be always true, as seen in Figure 3.12 where the ball has zoomed in and at that moment the moving object is bigger than the background.

**Figure 3.9:** In an image histogram, the \( x \) axis shows the gray level intensity and the \( y \) axis shows the frequency of this intensity.

**Figure 3.10:** Frame 1, the moving object is smaller than the background.
Figure 3.11: As seen the $H(f)$ for background (0 level) is higher than the one for the moving object (255 level).

Figure 3.12: Frame 23, the ball has zoomed in and now the moving object is bigger than the background.

Figure 3.13: Now the moving object has a higher $H(f)$ value than the background.
As explained before was the number of moving objects, from now on the number of moving objects will be $M$.

Taking this into account, we are going to ask the user of the application which is the exact background colour.

### 3.3.2 Image thresholding

Thresholding methods replace each pixel from a specific range into a desired value. We define two thresholds $T$, a lower ($T_{\text{low}}$) and an upper ($T_{\text{up}}$) one and replace each pixel intensity $I_{i,j}$ placed in the range for a new one $I_{\text{out}}$.

As stated previously, our algorithm needs a 3D image for each moving object in the video. Taking the original 3D image, we are going to threshold this image $N-2$ times$^1$. We want to obtain this amount of 3D images in order to segment each object and create an isosurface for each of them. This way, the user will be able to visualize and analyse each of them separately. Consequently, we are going to cover the objects that are brighter than the one we want to segment. So, in our case, the new image intensity $I_{\text{out}}$ will be 0 value, black colour, as the background. And our threshold will be:

$$T = (T_{\text{low}}, T_{\text{up}}) = (I_{\text{obj}} + 1, 255)$$

This means that we are taking all the objects brighter than our object ($I_{\text{obj}}$), from level $I_{\text{obj}}+1$ to 255, white colour. Once we have these objects selected we are going to convert them into black colour.

![Figure 3.14](image)

Figure 3.14: the graphic shows the transformation of the output image as explained before.

If we look at our previous example we are going to obtain 3 thresholded images because we had 4 objects. The following figure shows how many objects will appear in each thresholded image. As seen, all the intensities brighter than the object intensity are erased. Furthermore, the original 3D image is the segmented image for object 4 as is the brightest one. This is the reason why we only threshold $M-I$ images.

---

$^1$ As explained before $N-I$ was the number of moving objects, from now on the number of moving objects will be $M$. 
The obtained images are the following:

**Figure 3.15:** The figure shows the segmented objects in each image.

**Figure 3.16:** Resultant image from object 1.

**Figure 3.17:** Resultant image from object 2.

**Figure 3.18:** Resultant image from object 3.

**Figure 3.19:** Original image which corresponds to segmented image for image 4.
3.4 Isosurface generation

Once we have a 3D image for each moving object, an isosurface is going to be created for each one of them. After generating the isosurfaces, they need to be mapped in order to create the geometry and finally converted to actors. All these concepts will be explained below.

3.4.1 Isosurface

An isosurface computes and generates a surface within a volumetric data field. It extracts the surface (contour or boundary in 2D, that is, it can be locally approximated by a 2D plane) of a 3D scalar field, in our case will be a 3D image. The algorithm used to compute the isosurface is Marching cubes.

3.4.1.1 Marching cubes

Marching cubes algorithm was published in 1987 by Lorensen and Cline [12]. Its aim was the extraction of a polygon mesh from a scalar three-dimensional (voxel) data. Taking an isovalue as a threshold, they extracted the contouring in 3D. Its first approach was the reconstruction of medical images such as computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography (SPECT). Knowing this, it is obviously why we have used this concrete algorithm in our application (as said before, our aim is very similar to medical image reconstruction).

There are two main steps in this algorithm that are carefully explained in [12]. First, it locates the surface corresponding to a user-specified value and creates a triangulation of the surface. Then, to ensure a quality image of the surface, the normals to the surface are calculated at each vertex of each triangle. Marching cubes uses a divide-and-conquer approach to locate the surface in a logical cube created from eight pixels; from each two adjacent frames. Although this may seem an easy approach, there are more steps inside each one.

The algorithm steps are [15]:

1. Consider a cube.
2. Classify each vertex as inside or outside.
3. Build an index.
4. Get edge list from table [index].
5. Interpolate the edge location.
6. Go to next cell.

The description of every step in detail can be found in the Appendix A of this project. It is important to describe in detail this algorithm because is the most important part of our application, where the 3D volume is generated.
3.4.2 Mapping

To generate the object geometry, we need to map it after the isosurface is created. Mapping involves calculating the coordinates of a point known with respect to a coordinate system to a new coordinate system. Besides, more than one object may refer to the same mapper. In this stage of the algorithm, every object is converted from model coordinate system into world coordinate system (explained in section 3.5.4).

3.4.3 Actors

The last step is creating actors. Now that a mapper of each object is created we need to generate an actor, which is the representation of an object rendered in a scene. Thanks to the mapper we get the actor geometry. Then, we need to define the surface properties: colour and opacity. Both mapper and surface properties defines an actor and are in the world coordinate system.

Our isosurfaces do not have a specified colour, so we need to set a colour and an opacity in the actor properties. Since our main aim is visualizing and distinguishing between objects, we do not need to apply a texture into the objects. Therefore, we are just going to set different colours to every object.

3.5 World space projection

The last step consists in the actors’ projection in a 3D environment. In this section a very important part of the algorithm is taken part, the rendering. Once all object actors are created, they need to be rendered in the world space in order to be visible to users. This section explains what involves rendering a scene [16].

3.5.1 Overview of rendering

In computer graphics, rendering is the process of producing an image from a model description. We need to simulate the way lights, cameras and objects interact in the world around us, on a computer. First of all, we begin by looking at a real scene:
In the scene above there’s a light source (in this case the sun) which emits rays of light in all directions. Some of these rays strike the object (cube), its surface absorbs some of the light and reflects the rest of it. Some of these reflected rays will then enter the person’s eyes and the person will see the object. Likewise, some rays of light will be reflected into the ground and enter the eyes. This is the way human beings perceive the world and this is what we want to achieve with the rendering.

On a computer, instead of the sun we generate a light source, instead of objects we create actors and instead of a person we build a camera. To understand these concepts, we are going to briefly explain them in the next sections.

Moreover, the graphic technique used in our algorithm is object-order method. This process renders each object at one time. Rather than ray tracing that renders pixel by pixel, object-order technique renders one object completely and then the other ones.

The following example shows the scene we need to reproduce:

![Diagram](image.png)

**Figure 3.20:** Real scene showing light behaviour.

![Diagram](image.png)

**Figure 3.21:** Camera, light and object (actor) in the world space [27].
3.5.2 Lights

The lights are a really important part of a scene, if there is no light the resultant image will be black. This is why we need a source of light in the world space. When a ray of light strikes the actor it is reflected into the camera and displayed [16]. The Phong reflection model [17] breaks the light’s emitted intensity into three components: diffuse reflection, specular reflection and ambient light:

- **Diffuse reflection**: the incident light is reflected into many different directions and produces matte surfaces.
- **Specular reflection**: the incident light is reflected into a main direction and produces shiny surfaces.
- **Ambient light**: is the indirect light, all the rays reflected from other actors generates the ambient light, all the actors act as light sources for this light.

Both lights are modelled for directional light from a point light source.

In our scene, the light will be placed at the same position as the camera.

3.5.3 Cameras

Once we have lights and actors all we need to render the scene is a camera. To determine how a 3D scene is going to be projected onto a 2D plane (image), there are several factors to be taken into account: the position, orientation, and focal point of the camera, the method of camera projection, and the location of the camera clipping planes [16]. Coming next, all these factors are going to be explained.

3.4.1.2 Camera view

The location of the camera is determined by the position and the focal point. Both factors gives information about the situation of the camera in the world space and where it points. The vector defined from the camera position to the focal point is called the direction of projection, explained in detail in section 3.5.4. Orientation depends on the position, the focal point and a third vector called view-up.

![Camera view scheme](image)

**Figure 3.22**: Camera view scheme.
3.4.1.3 Projection and clipping

Once the camera is situated we need to decide what it can see. There are two factors to define, the first one is the method of camera projection. This method states how actors look with distance, how they are mapped to the image plane. The method used in the algorithm will be perspective projection that is produced when all the rays coming from the light source pass through the same point (centre of projection). To implement this projection we must determine a camera view angle.

Using perspective projection closest things seems bigger than further ones and parallel lines intersect at the vanishing point. In the example below we can observe the features explained previously:

![Perspective Projection](image)

**Figure 3.23:** Example of perspective projection.

On the other hand, there is the clipping planes which contain all the content of the scene. These planes intersect the direction of projection and they are usually perpendicular to it. Only actors or fractions of actors within the clipping planes are visible. The front clipping plane is the minimum image plane and the back clipping plane is the maximum.

All the parameters mentioned before define a rectangular pyramid. The tip of this pyramid is placed at camera’s position and it is truncated at the front and back clipping planes. The resulting area is called frustum and it is all the space visible to the camera.

![Camera Space Representation](image)

**Figure 3.24:** Camera space representation [16].
Once the camera is created, we can see how the 2D image is generated. The source of light emits, some of its emitted rays will pass through the camera lens. The camera properties determine which rays get captured and projected. The resulting image will be generated by the rays that intersect the camera’s position and are inside the viewing frustum.

Summarising, the light travels from its source to the actor’s surface, where is reflected. The reflected rays, which are captured by the camera, produce the 2D image.

### 3.5.4 Coordinate Systems

There are four coordinate systems used in computer graphics: model, world, view and display. It is important to know which coordinate system is being used in every moment of the algorithm to locate everything properly. To pass from one coordinate system to another we need transformation matrices. In the following chart we can see the transformations from one system to another one:

![Coordinate systems chart](image)

**Figure 3.25:** Coordinate systems chart.

#### 3.5.4.1 Model coordinate system

The model coordinate system identifies the shape of the actor, its geometry. It is typically a local Cartesian coordinate system, but this depends on the decisions of who has created it.
3.5.4.2 World coordinate system

The world coordinate (or world space) is the 3D space in which the actors are positioned. This coordinate system is very important since it defines all the locations of the actors, the camera and the source light. As explained in Figure 3.38, we need to convert from model coordinate system to world space with a model transformation matrix. Every actor must scale, rotate and translate its model into the world coordinate system.

![World coordinate system](image)

Figure 3.27: World coordinate representation.

3.5.4.3 View coordinate system

First of all, we need to create the coordinate system for the camera, as the view system will depend on the location of the camera in the world space. Camera view features have to be defined. As explained, we have three important features to define:

- Position \( P = (p_x, p_y, p_z) \)
- Focal point \( F = (f_x, f_y, f_z) \)
- View-up vector \( U = (u_x, u_y, u_z) \)

Creating a vector using the position \( P \) and the focal point \( F \) and taking the view-up vector \( U \), a new plane \( Z \) can be defined. We can construct an orthogonal vector \( O \) to this plane that will give us the third vector of the coordinate camera system [18].

The view coordinate system represents what is visible to the camera. When we create a volume in 3D, we want to give the impression that we are viewing the world from a certain position (as if with the camera) and that means we have to transform every location in the world space. Using the camera coordinate system we know what is inside the frustum and therefore visible to it. In this coordinate system, the \( x \) and \( y \) axis represents the location of the objects in the 2D image plane and the \( z \) axis the distance from the camera. In this system the perspective projection that we have talked before takes place.
3.5.4.4 Display coordinate system

Finally, the display coordinate system is similar to view coordinate but the $x$ and $y$ axis are pixel locations. Depending on the window’s size, the system will determine the transformation from view system to the display one. Likewise, the $z$ value in the display coordinate system also represents depth into the window.

3.5.4.5 Coordinate systems summary

In Figure 3.41 we can observe the relationships and transformations mentioned previously between all the coordinate systems.
Figure 3.30: Summary of coordinate systems [16].
Chapter 4

PROPOSED APPLICATION

4.1 App definition

4.1.1 Main idea

Video provides a wide range of information. The most exploited features are motion and appearance, but there is other data much less explored, temporal interactions among objects. We aim to analyse this interactions creating an application which will allow to view a video in a 3D environment. Taking segmented videos as our input, we are going to reconstruct the frames in 2D (frame) plus time.

We propose a powerful, but user-friendly, application where researchers will be able to visualize their video segmentations and analyse objects interaction. Moreover, we want this application to be a tool that will help the evaluation of their segmentations and test their accuracy.

Furthermore, we want to implement some editable features in order to ease the visualization, such as changing the colour and opacity in real time, choosing which objects are visible or visualizing the original frame superimposing the segmentation frame.
4.1.2 Used technologies

In order to create our application we have used a wide range of technologies. The operating system used is Linux (Ubuntu). This choice has been made due to the request of the Image Processing Group since this is the environment they always use in their projects.

Moreover, the app has been programmed in Python. From all the technologies mentioned in section 2.3, we have finally chosen Visualization Toolkit (VTK) which can be programmed in Python. As explained, VTK is a set of libraries consisting of 3D computer graphics, image processing and visualization. From all their techniques, there are some as useful as 3D image reading, isosurface generating, image thresholding, rendering, and so on.

Apart from VTK library, thanks to program in Python, we have been able to use other image and mathematics libraries such as PIL (Python Image Library), Numpy, Math, among others.

Finally, we need a GUI (Graphical User Interface) where print our volume and the user can interact. For developing our application, we have preferred Tkinter, a GUI package used in Python. It has permitted us to create the user interface from the beginning and personalize all kinds of gadgets.

4.2 Design and features

The application will be focused in one main window. Inspired by visualization applications already available in the market, it will create a central window where the 3D environment is going to be rendered. Additionally, this main window will have a dropdown menu with all the app options such as adding original images, editable features or making a screenshot.

Researchers, the users in our application, will be able to modify the 3D resulting volume in order to ease the visualization and analysis of the spatiotemporal tubes. We have implemented some edit features that are also going to be described in this section.

4.2.1 Input data

To be able to run the application, the user must enter some input parameters:

- **Frames folder location:** the application need to know where it can find the frames that have to be reconstructed, i.e. /home/user/folder
- **Frames root:** the algorithm will find automatically the frames but it needs to know the frames root. If the set of frames are: image1.pgm, image2.pgm, image3.pgm, the frames root will be ‘image’.
• **Background colour:** as explained in chapter 3, the background colour must be black. If that is not the case, the user must specify which colour the background has in the segmented frames.

• **Original frames:** if the researcher wants to test his/her segmentation, he/she must add the original frames in the application system.

The user may specify these parameters using two different ways: the Linux console and an input window launched by the application. The first option will be the console, when the user runs the app the parameters must be entered next to the application command. Then, the application checks if these parameters have been entered. If so, it starts generating the 3D volume in the main window. Otherwise, if the system detects that the parameters are missing, it creates an input window which asks the input information.

### 4.2.2 Main window

The main process of this application consists in the algorithm described in chapter 3. Once the system has the information needed, it starts generating the 3D reconstruction of frames. The volume is rendered in a main window where it is visible the world space taking into account the camera position. As explained in section 3.5, the visible volume depends on the camera position. However, the user will be able to interact with the volume moving the camera orientation and position using the computer mouse.

Furthermore, the main window will have a dropdown menu containing all the options in our application. This menu will be structured in three main sections and subsections:

- **File**
  - Upload original frames

- **Options**
  - Surface Properties
  - Discretization
  - Object visualization
  - Frame visualization

- **Export**
  - Take screenshot

In the following sections we are going to explain the main idea of all these options menu.

### 4.2.3 Upload original frames

If the user has not selected the original images before, he/she must select this option and add the original frames before their visualization in the 3D scene. The advantage of this option is that the user can select these frames at any time, not just in the input window.

When the option is selected it appears an open window and the user must select all the original frames at once. These frames are going to be saved by the system and visualized using the ‘Frame visualization’ option.
4.2.4 Surface properties

Aiming to ease the visualization, the user will be able to modify the colour and opacity of every rendered object. Motivated by this goal, we have created a window to change the surface properties.

In the top of this window will appear a list of the objects. This list will be generated automatically depending on the video. The user will select the object that wants to modify from the list. Afterwards, the user might adjust the colour or the opacity changing the values of the sliders. The window will have four sliders, three concerning RGB colour (red, green and blue) and the fourth concerning the opacity.

4.2.5 Video discretization

In the same way as surface properties, the user might modify the distance between frames (\(\Delta t\)). The distance corresponds to the separation between two consecutive slices in the \(z\) axis in the world coordinate system. This window will be really simple. It will have a single slider from 1 to 30 and the user will be able to change its value in real time.

4.2.6 Object visualization

One of the researchers’ requests was to visualize only one object at a time. To accomplish this goal, we have created an object visualization window. Like surface properties window, a list of rendered objects is created. The main difference is that in this list more than one object might be selected. The selected objects will be visible and the unselected invisible.

In order to ease the selection, we have added two more options to the window: ‘View all’ and ‘Outline’. ‘View all’ will select and make visible all the objects of the list. In the 3D reconstruction we will render a bounding box with the image size. This bounding box is called ‘Outline’ and its visibility is turned on and off with this last button.

4.2.7 Frame visualization

Another important feature is the amount of frames to visualize. An important issue for the researchers is choosing the range of visualization. We have created a sophisticated window with a lot of options to meet all the researchers’ requirements. The window is divided in two main sections: ‘Frame range’ and ‘Individual frame’. The user will be able to visualize a range of frames, chosen by them, or an individual frame from all the set of images. Furthermore, in both modes will be possible the original frame visualization.

The frame range visualization mode will consist in two sliders. The first one concerning the first frame range and the second one concerning the last frame range. Moreover, a check button will be added after the sliders to enable the original frames slider. If the user moves this last slider, the original frame corresponding to slider value will be printed in world space.
On the other hand, the individual frame visualization mode will have one slider to select the visualized frame. Once this mode is enabled, the 3D volume will be converted into a 2D image. Since the rendered image will be just as the input frame, we have taken advantage of this simple visualization and have added a ‘print original image’ option. If this option is turned on, the original image of the current selected frame will be rendered. The user might use the individual visualization mode to check the segmentation accuracy.

### 4.2.8 Screenshot

In order to export images of the resulting 3D volume, we have implemented a button that will screenshot the main window and will create a saving window. The user will be able to choose the saving folder and the image name.

### 4.3 Requirements

The catalogue of requirements is the specification of the expected behaviour of any software application. We need to predefine a number of requirements which are considered necessary for the app. After talking with the researchers from Image Processing Group, we have listed and described all their requirements in the following section.

#### 4.3.1 Functional

Functional requirements describe all the interactions that users will have with the software.

**Input data**

1. The app must permit the user to enter input data, both by console and by window. The users must enter:

2. The frames folder location and the root name of the frames.
3. The background colour in case is different from black.
4. The same amount of original images as frames in the video.

5. The users may enter the original images location folder, although they can enter it later in a main window option.
6. The segmented frames and the original images must not have the same root name.
7. The system displays an error message if any of the data is incorrect or does not meet the conditions specified before.
8. If the data has been correctly entered, the system will save the inputs and use them to create the frames 3D reconstruction.
9. If the user wants to leave this window, another window will be displayed asking if the user really wants to leave or wants to continue.
Main window

10. The main window will consist in a dropdown menu and a space where the volume is displayed.
11. The users can interact in this main window and move inside the world space freely.
12. Every change given by the edit windows will be displayed in real time in the main window.

The system must:

13. Recognise all types of image formats.
14. Work with different image sizes.
15. Select the segmented frames automatically from the given folder.
16. Print every object in a different colour and independently.

Dropdown menu

17. The main window menu has three main sections:
   a. **File**: where the original images can be uploaded if they have not been added before.
   b. **Options**: where the editable features can be found. Every change in the 3D volume is saved by the system and updated in real time.
      i. **Surface properties**
      ii. **Discretization**
      iii. **Object visualization**
      iv. **Frame visualization**
   c. **Export**: different options of exporting the 3D resulting volume.

Upload original frames

18. Visualize the original frame with the 3D volume.
19. If the user wants to visualize the original frames without being uploaded first, the system will show an error.

Surface properties

20. The colour of the rendered objects must be modifiable independently.
21. The opacity (transparency) of the rendered objects must be also modifiable independently, two objects might have two different opacities.
22. The user needs to know which object is being modified at any time.
23. The window must be intuitive no matter how many objects the video contain.
24. If the ‘surface properties’ window is closed, the system must save all the changes.

Video discretization

25. The distance between two consecutive frames must be modifiable.
26. If the ‘video discretization’ window is closed, the system must save all the changes.
Object visualization

27. The user may turn on and off the objects visualization.
28. Each object must have its own checking button.
29. The window must be intuitive no matter how many objects the video contain.
30. Add an option to visualize all the objects at once.
31. The user might turn off the 3D image bounding box.

Frame visualization

32. Be able to visualize a frame range of the video or an individual frame.
33. The initial frame range must not be greater than the last one.
34. The last frame range must not be lower than the first one.
35. The user may visualize the original frame in both modes, ‘frame range’ and ‘individual frame’.

4.3.2 Non-functional

Non-functional requirements or quality attributes specifies judging criteria system operation, in other words, describe how the system works, its behaviour. They are usually grouped in non-functional sections.

Usability

1. The application should consist of a simple, intuitive and attractive interface.
2. Data entry should be structured and intuitive to the user.
3. The visualization result should be clear and understandable.

Performance and portability

4. The application must be supported for most possible Ubuntu versions.
5. It is not necessary to have an Internet connection, it works offline.
6. The application will be suitable for Python 2.7 and 3.4.
7. It is expected to run the application with time not exceeding 5 seconds.
8. Inside the app the responsive time must be lower than 1 second.

Supportability

9. The application supports up to 20 objects in order to be effective.
10. The maximum recommended image size is 1280x960.

Open source software

11. The application must be created using open source software.
4.4 Mock-up

Mock-up is a model of design used for demonstrating how the software interface will look like after the source code is written. Before starting to code, we should design and draw every window to avoid having to go back and make changes to the code. In this section we are going to show the mock-up of our application, the relationship between each window and a description of their functionalities.

4.4.1 Interface design

The app interface has been designed to be intuitive and simple but, at the same time, powerful and fulfilling all the requirements described before in section 4.3.

4.4.1.1 Input window

If the user does not introduce the input data using the console, the first window that they will see will be the input window (the image above). As mentioned before, we could have up to 4 input parameters depending on the video. The first two input parameters are required to launch the application. We have added an asterisk at the end of both parameters, like in similar existing forms, to emphasize these mandatory information. If the user tries to continue before entering the required information, the system will respond by emphasizing the missing parameter.
The background colour parameter is only required if the background is different from black. So it is not mandatory as long as this requirement is met. In case the background colour is different from black and the user wants to continue, having left the parameter as empty, the application will show an error message in the console.

The last parameter, which is selecting the original images of the video is optional. The user can add these images in this window or later.

### 4.4.1.2 Main window

The main window will consist in a big space to render the 3D volume. The user will be able to move and rotate the view using the mouse. As seen, the menu is always visible at the top of the window in order to ease the edition of the volume.

This will be the main window and all the rest of the windows will be created from this one. If the user clicks the exit button, the system will release all used memory and close all the opened windows including itself.
4.4.1.3 Upload original images

When the user selects the option of adding the original images, an uploading window will appear permitting to choose them. The user must find the container folder and select all the images at once. Finally, click on the ‘open’ button and the system will save all these images.

4.4.1.4 Surface properties

As mentioned in the requirements, it is necessary to know which object is being modified. The system will create automatically a radio button for each object in the video. A radio button is used to implement one-of-many selections, only one object can be selected at once. As seen in the mock-up, after the radio buttons, there is a colour pattern which shows the colour of the selected object. Each time the user changes the slider’s value, this colour is updated. When the selected object is changed the colour pattern is updated by the current object colour.
The first three sliders Red, Green and Blue will have a range of values between 0 and 255, all the possible values that a colour can set. On the other hand, the last slider, corresponding to the opacity value, will have values between 0 and 100. This value will coincide with the percentage of transparency in the object.

4.4.1.5 Video discretization

![Video discretization window mock-up.](image)

This is a simple window. The only widget of this window is a slider controlling the frames distance. As explained before, the user will be able to change the $z$ axis of the video. The slider will have values between 1 and 30. It is important to mention that the 30 value has been chosen experimentally. We have proved that this is the maximum efficient value in terms of visualization.

4.4.1.6 Object visualization

![Object visualization window mock-up.](image)

In order to ease the visualization, we have created an edit window to choose the visible objects. This window starts with a list of all the objects in the video. Every object has a check button in order to permit the user to select or unselect the visualization. Moreover, the window includes two more check buttons. As mentioned in previous sections, these buttons permit to make all the objects visible at once and turn on and off the 3D image bounding box, the outline.

The relationship between ‘view all’ button and all the objects buttons must be intuitive. That means that, if the ‘view all’ button is pressed, all the objects check buttons will be enabled. On the contrary, if some object is stated as invisible, the ‘view all’ button will be disabled.
4.4.1.7 Frame visualization

The last edit window is the most sophisticated and completed one. The purpose of the frame visualization window is to allow the user to choose the range of visualization. This window is divided in two stages: the selection of the mode and the selection of the frames.

As explained before, there are two visualization modes (first stage). We have added two radio buttons to select the mode, since only one mode can be selected at once. For this reason, the selection of frames (second stage) is disabled until the user choose the selection of the mode. The Figure 4.9 shows the first stage, the window that will appear at the beginning of the edition.

**Figure 4.9:** Object visualization window mock-up, first stage.

**Figure 4.10:** Frame range mode chosen in the second stage.

**Figure 4.11:** Individual mode chosen in the second stage.
Once the user has chosen, the selected mode will be enabled. The frame range mode has two sliders to select the frame range. The first one gives the first frame value and the second one gives the last frame value. Both slider lengths will be the number of frames. Moreover, one of our requirements was the first frame could not be greater than the last one and vice versa. For that reason, we have blocked the sliders when the user tries to neglect that requirement.

Under the sliders there is a check button that enables the original frame visualization when pressed. The original frame visualization will be only enabled if the user has uploaded the images previously. Otherwise, the system will show a warning message reminding the user to upload the images first. When the original frame visualization is enabled, the user might move the slider and choose the original frame. The frame will be rendered in real time, synchronized with the slider movement.

On the other hand, the individual frame mode has only one slider which is also as long as the number of frames. When this mode is enabled, the 3D volume is converted into a 2D image. Besides, there is also an original frame check button that works in the same way than in the first mode. It shows the original frame behind the segmented frame.

### 4.4.1.8 Screenshot

When the user wants to save a screenshot of the main window, he/she selects the screenshot option and the window from Figure 4.12 appears. It is a really intuitive window where the user should write the screenshot name and select the save option.

#### 4.4.2 Layout scheme

Once we have designed the application windows we should decide the interactions between them. We have created buttons with intuitive names in order to guide the user to all the options and features of the application.

The main window has a dropdown menu and all the options will be available in there. The user will know that whatever they are looking for is going to be in that menu.

The following charts will show the layout scheme of our application.

**Figure 4.12: Saving image window mock-up.**
Figure 4.13: Input layout scheme.
Figure 4.14: Application layout scheme.
4.4.3 Final design

The first task of application implementation is coding the windows described previously. In this section we are going to show the final appearance of all the application windows. We are going to compare the initial mock-up design with its final implementation.

![Input data window comparison.](image1)

**Figure 4.15:** Input data window comparison.

![Quit window comparison.](image2)

**Figure 4.16:** Quit window comparison.

![Video discretization window comparison.](image3)

**Figure 4.17:** Video discretization window comparison.
Figure 4.18: Main window mock-up.

Figure 4.19: Main window final design.

Figure 4.20: Surface properties window comparison.
Figure 4.21: Object visualization window comparison.

Figure 4.22: Frame visualization window comparison with disabled modes.

Figure 4.23: Frame visualization window comparison with first mode enabled.
4.5 Class diagram

A class diagram is a type of static structure diagram describing the relationship between classes in a particular programming code. It describes the structure of the project by showing the system’s classes, their attributes and their methods. This diagram is used to describe briefly the organization of the programming code.

Our class diagram is really sophisticated and, for this reason, we have implemented two diagrams, a simple one showing just the classes names and the real one [19] [20].
Figure 4.27: Class diagram of our application.
Chapter 5

IMPLEMENTATION

In chapter 3 we have defined the steps of our algorithm in a theoretical way. Now in this chapter, we are going to describe the pipeline of our application from a technical point of view and detailing the actual implementation. Our pipeline is divided in four main steps: frame processing, object identification, volume generation and volume displaying.

Section 5.2 is devoted to describe the implementation of each of the four steps while next Section 5.1 focus on describing VTK process and several graphical objects used through the application development.

5.1 VTK

As explained in chapter 4, we have finally decided to use Visualization Toolkit (VTK) to program our application. Create graphic applications with VTK is a process consisting in two basic steps:

1. Build a data pipeline to process the information.
2. Create graphical objects to interpret these data.

Build a pipeline means to connect sources (to create data), filters (to process data) and mappers (to transform data into graphs).

On the other hand, there are a lot of VTK graphical objects. Some of the most used are described in this section (see also [16]). In the process of rendering a scene VTK uses seven basic objects:

- **vtkMapper**: the geometric representation for an actor. More than one actor may refer to the same mapper.
- **vtkProperty**: defines the surface properties of an actor such as colour, opacity, and lightning properties (i.e. specular and diffuse).
• **vtkActor**: is the representation of an object rendered in a scene. An actor must include a vtkMapper and a vtkProperty which are both in the world coordinate system.

• **vtkLight**: a source of light that illuminates the scene. There can be more than one light illuminating the scene.

• **vtkCamera**: defines the visible zone of the scene. It will depend on the position, the focal point, the perspective projection and the clipping planes of the camera.

• **vtkRenderer**: coordinates the rendering process which involves all the actors, lights and cameras present in a scene.

• **vtkRenderWindow**: creates a window where all the rendering process is displayed. More than one renderer may display in the same window.

Another interesting object that was not mentioned in the list is the **vtkRenderWindowInteractor**. This object permits the user to interact with the data without rendering again the scene. Using the computer mouse the user might move the camera position so that they can view the scene from a different point of view.

Finally, before all the rendering process starts, we need to read the image. We are going to create an instance of **vtkImageData** once the frame is read. This object is really useful in our application because it can represent 1D line samples, 2D images and 3D volumes.

![Illustrative diagram of graphic objects](image.png)

**Figure 5.2**: Illustrative diagram of graphic objects [16].

In Figure 5.2 we can see the graphical objects described previously. There are three actors which are defined by the vtkMapper and the vtkProperty. These actors can be shown thanks to the vtkRenderer and the vtkRenderWindow. We illustrate using two windows the fact that there are two vtkRenderWindow instances.
5.2 Application pipeline

This section describes in detail each step taken by our application. Following the pipeline previously mentioned, we are going to pass through all their parts and describe the pseudocode and the activity diagram for each of them.

5.2.1 Frame processing

Assuming that the process of entering the input data has been done correctly, the system is going to start processing the frames. The required information to start this procedure is:

- path: frames folder location.
- root: common data in frames names.
- bg: background colour of segmented frames.
- Nframes: number of frames in the video.
- imageRange: range of frames to be rendered

At this point, all these parameters are going to be used as input data in frame processing method. The following activity diagram shows the process of adapting the frames in order to isolate the in different objects lately.

![Frame processing activity diagram](image)

Figure 5.3: Frame processing activity diagram.

As a consequence of adapting the frames, they are going to be edited. Since the user does not want their frames to be changed, the system is going to create a temporal folder where it will work with from now on. Then, the system will check the format of the images and convert them to PNG (in case they were not). Afterwards, the images are going to be...
copied and doubled (the temporal folder will have twice each frame). The reason why we have to duplicate each frame has to do with the volume generation and it is going to be explained in that section (5.2.3).

Then, the frames will be converted into grayscale, no matter the initial number of channels. This decision has been taken after experimentally verifying that, checking the nature of the frames consumes more execution time than converting them to grayscale directly. The following graphs show the difference between both procedures.

![Conversion RGB to Grey Scale](image)

**Figure 5.4:** Conversion RGB to Grey Scale graphic comparison.

In the experiment for obtaining the graphic above, we had taken a set of frames with sizes from 640x480 to 5120x3840. All the sequences are made of 6 frames and all the frames have 3 objects. In order to take the worst case in which the user can be found, all the frames where RGB so they all need a conversion to Greyscale. As seen in the graphic, the execution time when the system analyses the channels of the set of frames is much higher than without analysing them. For this reason we have decided to make always the conversion, without analysing the channels of the frames.

As explained in chapter 3, the frames need to have black background colour. Thus, if this condition is not fulfilled, the images need to modify the background colour. The original background colour must be a number between 0 and 255. Since we do not know the nomenclature the user will use, we have created a dictionary that will translate the colour whether the user gives the colour name (i.e. white), the colour array (i.e. [255,255,255]) or the hexadecimal version (i.e. #ffffff).

Afterwards, the system must change the frame names. The object used to read the video frames is vtkPNGReader, this class reads 3D images as long as the images that compose the 3D image are in a particular format name. The reader will interpret all the images started with a specified root and an index as an element of this 3D image, i.e. image.png.1, image.png.2, image.png.3. For this reason we are going to change the frames names into this format, taking the stem of the video frames name and adding a point and an index at the end of the image name.
Finally, the system will create an extra black image at the beginning and at the end of the frame range. This last method will be done in order to create a ‘cover’ in the volume. As seen in the following figures, when the system add these two black images the volume appears closed.

![Generated volume without adding the black images.](image1)

![Generated volume after adding the black images.](image2)

**5.2.2 Object identification and isolation**

Once the frames are adapted we need to create a 3D image for each object. The steps used to identify the objects are described in the following activity diagram:

![Object segmentation activity diagram.](image3)
The isovalues, or object values, are going to be found automatically. Using an image histogram method (computed using PIL library), the system is going to obtain all the values appearing in the image. The isovalues are going to be found as described in the following pseudocode:

```
N = Nframes
for i = 0 to N do:
    number, colour = unique(histograma[i])
    for j = 0 to NUMBER OF HISTOGRAM VALUES:
        if number[j] == 0: # Empty value
            no_value = j
        elif colour[j] == 0: #Background colour
            bg = j
    for k = 0 to NUMBER OF HISTOGRAM VALUES:
        if k != no_value and k != bg:
            ISOVal = ISOVal.add(colour[k])
ISOValues = unique(ISOVal)
```

The system generates a histogram for each frame and taking the unique values found in all the histograms (deleting the empty value and the background colour), we get the isovalues.

Afterwards, the system builds a dictionary where the objects will store their information parameters: object ID, colour, opacity and isovalue. Using a colour matrix which contains six different colours, we initialize each object colour with this matrix. If there are more than six objects in the scene, the counter starts again and two objects will have the same colour at the beginning. Every time the colour or the opacity of any object experience a change, the dictionary need to be uploaded.

Once the 3D image is read, if the video has only one object the segmentation object has ended. Otherwise, the system will threshold the 3D image as explained in chapter 3 and thus create a 3D image for each object.

### 5.2.3 Volume generation

The most important part of our algorithm, is volume generation. Nevertheless, it turns out to be easy to program thanks to VTK objects. The steps described in diagram activity are going to be done once for each object:

![Volume generation activity diagram](image)

**Figure 5.8:** Volume generation activity diagram.
The isosurface is generated by vtkMarchingCubes object. As described in chapter 3, we use Marching Cubes algorithm to generate the isosurface of the 3D image and it can be created in only three lines:

```python
objectISO = vtk.vtkMarchingCubes()
objectISO.SetInput(image.GetOutput())
objectISO.SetValue(0, IS0Value)
```

We just need to specify the isovalue and the image which extract the isosurface.

The following step is mapping the isosurface. Using the vtkMapper object, the mapping process is also easy to code:

```python
objectMapper = vtk.vtkPolyDataMapper()
objectMapper.SetInput(objectISO.GetOutput())
objectMapper.ScalarVisibilityOff()
```

We declare the scalar visibility off in order to turn off the scalar data because the objects will be coloured with the actor properties.

Finally, we create the actors with vtkLODActor object. We use vtkLODActor object instead of vtkActor because the first one stores multiple levels of detail (LOD) and can automatically switch between them. It selects which level of detail to use based on how much time it has been allocated to render.

```python
actorObj = vtk.vtkLODActor()
actorObj.SetMapper(mapObj)
actorObj.GetProperty().SetColor(arrayColor)
actorObj.GetProperty().SetOpacity(opacity)
```

Then, we set the mapper and the surface properties, which can be found in the dictionary, to the actor.

### 5.2.4 Volume displaying

The rendering process is developed with a wide range of VTK objects described previously. In this section we need to simulate the position of the camera and the light (vtkCamera and vtkLight, which will be created automatically on the camera), the renderer (vtkRenderer), the window to render the scene (vtkWindowRender) and the interactor to permit the user to move inside the scene (vtkWindowInteractor).

![Figure 5.9: Volume displaying activity diagram.](image-url)
The first step is creating the bounding box (outline) of the scene. We create an actor for the outline, just like it is done for others objects in the scene. Then, a vtkRenderer is created, we set the background scene colour and add all the actors in the renderer.

Afterwards, a camera needs to be set up. As explained in chapter 3, we need to declare three vectors: position, view-up and focal point. The combination of these three vectors states the initial view of the scene. We have chosen a frontal view of the scene because it gives a general interpretation of the objects movement in a first look. The following code lines show the camera setting up.

```plaintext
camera = vtk.vtkCamera()
camera.SetViewUp(0, 1, 0)
camera.SetPosition(0, 0, 1)
camera.SetFocalPoint(0, 0, 0)
camera.ComputeViewPlaneNormal()
```

Moreover, the system creates a vtkRenderWindow which is going to be our main window where the scene is going to be displayed. We have set the window size to 600x800 since is an appropriate area to display the scene in a wide range of screen sizes. The renderer has been added to the render window in order to render all their actors in there. Finally, the system generates a vtkWindowInteractor. The last step is render the scene. At this moment, the system will display the created window in the screen with the scene inside it.

All the previous sections has been created in VTKRender class specified in chapter 4.

### 5.2.5 Editable visualization features

Apart from the basic pipeline, the user can edit some visualization features which can adapt the scene volume in real time. Each time the volume is edited, it needs to be rendered. Although it is computationally expensive, these visualization features will ease the analysis and visualization of the scene.

#### 5.2.5.1 Surface properties

In the DialogRGBO class we have created a dialog (window) to edit the surface properties. The user can edit the colour and the opacity of each object. This will imply the volume will change and therefore will be rendered again. The required information to edit and create this window is:

- dictionary: where the information about the surface of every object is stored.
- nObjects: number of objects in the scene.

The system will create the window depending on the number of objects and will create a button for each object where the user will be able to select the object to be edited. The system will response depending on two events: the selection of a new object and the variation (motion) of any slice. Knowing this, the activity diagram for this method will be:
Selection of a new object

When the user selects a new object the system will obtain their object index from the window. This index coincides with the object ID and, using the dictionary created in the app execution, it will be able to get the current object colour. Then, the system will update the sliders value and the colour label with its object colour.

Variation of a slice

In the event where a slice is moved, the system must update the colour label combining RGB channels to get the resultant colour. Moreover, the dictionary also needs to be updated with the new surface property of the edited object. However, the most important factor to update is the actor surface in the scene. Since this update implies changing actors’ properties the system will change the execution thread from DialogRGBO class to VTKRender class. The specific actor will edit its vtkProperty for the new one. Finally, as the actor has changed, we need to render to perceive the changes in the scene.

5.2.5.2 Video discretization

The video discretization window will be created in the DialogZ class. This is a really simple dialog with a unique event, the variation of the distance slice. Consequently, its activity diagram will be also simple:

![Activity Diagram](image-url)
The event is going to be identified in DialogZ class since the distance slice is placed in the video discretization window, nevertheless, the distance update will take place in the VTKRender class. The system will store the new Z axis value in a variable and will send this information to VTKRender class where the reader will be updated with the new distance value. Afterwards, the scene will be rendered.

```python
self.reader.SetDataSpacing(1,1,self.zValue)
```

The above code line shows the reader update. The first two variables correspond to x and y axis. These variables are set to 1 in order to keep the image resolution size of the spatial grid. The last variable is the zValue where the new distance value will be stored.

5.2.5.3 Object visualization

The DialogVisualization class creates a dialog to permit the user to choose which objects are visible and which are not. In the same way as DialogRGBO, the system creates a button for each object automatically and, consequently, it needs to know the number of objects in the scene. So, this is going to be the only required parameter.

The event that will activate the object visualization is the change of state of some button in the window. The check button has two states 0 and 1. We have programmed the system to show the object when the button is activated (1) and hide it when the button is disabled (0). In case the ‘View all’ button is activated, all the objects that were hidden are going to be shown.
As seen in the diagram above, the first step is to obtain the object index which has changed, then get the changed value in order to adapt the visibility to the contrary state. As in all other cases, the alteration of any feature regarding the actors needs to be done in vtkRender class. The change to the object visibility will be done editing the vtkProperty of the object, its opacity. Afterwards, the dictionary needs to be updated as the opacity of some object has been changed. Finally, the scene will be rendered.

**5.2.5.4 Frame visualization**

Finally, the user might edit the range of visualization and show the original image on the volume. The DialogFrame class is responsible for creating the frame visualization window.

This window is the most sophisticated one and, consequently, will have three possible events: the mode’s choice, the motion of some slider and the activation of original image rendering.
Mode’s choice

When the user chooses the mode of visualization all the widgets of that mode need to be enabled and the other mode will be disabled to avoid the choice of both modes at the same time.

Motion of some slider

If the moved slider is on the ‘Frame range’ mode, the system must assure it fulfils the following condition:

- Initial slider value < final slider value
- Final slider value > initial slider value

If the user tries to move the initial slider to a value upper than the final slider value, the condition will not fulfil and consequently the slider will be set to the current final slider value.

The next step is creating a new temporal folder inside the other one called ‘Range images’. We need a new folder to tell the system which images are going to be rendered in each case. Since we are going to edit the frame range, we need to create a new 3D image and, therefore, the system will repeat the object segmentation, volume generation and volume displaying methods before rendering the scene again.

The first step is deleting the renderer and the actors to free some memory. Taking the values from the sliders, we are going to get the new range to visualize. The system will
copy the new range frames into the new temporal folder. Since we need the frame index to start with 1 in order to create the 3D image correctly, we are going to rename the frames.

Afterwards, the system will obtain the new isovalues (if there is any) and the number of objects. The volume generation will be the same as the basic case. Finally, in the volume displaying, we will create a new outline and a new renderer. The new actors are going to be added to the new renderer and it is going to be added to the window render. Then, the camera will reset and the scene will be rendered again showing the new range frames.

**Activation of original image rendering**

As mentioned our application includes the possibility of adding the original textured information of the original sequence (not segmented) into the volume reconstruction rendering. To do so, they need to be uploaded by the user. This will be the first requirement of the window framework. In case the user wants to access this feature without uploading the original images, an error will be displayed in the console of the computer asking the user to upload them.

Once the user has uploaded the original images, the system will create a new folder called ‘Original images’. As we need to create a new renderer each time the system generates a new actor we are going to delete the actors from the current renderer and delete the rendered itself. Afterwards, the system will convert the original image to PNG in case is not already in that format and will copy the image in the new folder.

We are going to use the method proposed in [21] to render the 2D image into the scene. The display of a 2D image in VTK is not an intuitive task. We have used vtkImageSlice to render this image. First of all, the system will read the image. Afterwards, the system will calculate the centre of the image domain (taking its size and origin). We need the centre of the image to know the point where the image needs to be positioned in the world space.

We need a projective transformation matrix to transform the position of the image. In projective geometry 3D points are represented using homogeneous coordinates, thus with four coordinates. There are some components in a projective transformation matrix to scale and translate a vector. A projective transformation matrix can be defined as:

\[
P_t = \begin{pmatrix}
s_x & 0 & 0 & t_x \\
0 & s_y & 0 & t_y \\
0 & 0 & s_z & t_z \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Where the \( s = (s_x, s_y, s_z) \) vector describes how the object is going to be scaled and the \( t = (t_x, t_y, t_z) \) vector describes its translation.
Since we do not want to scale the image, we define the $s$ vector as 1, meaning there will not be any scaling. The translation vector $t$ will be the image’s centre vector $(c_x, c_y, c_z)$ calculated previously. Thus, our projective transformation matrix will be the following:

$$P_t = \begin{pmatrix}
1 & 0 & 0 & c_x \\
0 & 1 & 0 & c_y \\
0 & 0 & 1 & c_z \\
0 & 0 & 0 & 1
\end{pmatrix}$$

Once the projective transformation matrix is defined we are going to multiply each point in the image by this matrix in order to translate the image to the desired position.

$$\text{Position} = \begin{pmatrix}
1 & 0 & 0 & c_x \\
0 & 1 & 0 & c_y \\
0 & 0 & 1 & c_z \\
0 & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
p_x \\
p_y \\
p_z \\
1
\end{pmatrix} = \begin{pmatrix}
p_x + c_x \\
p_y + c_y \\
p_z + c_z \\
1
\end{pmatrix}$$

As seen, the resultant vector will be the same point plus the center coordinates. A vtkImageReslice will be created during this process. Finally, we are going to convert this object into a vtkImageActor which is a specific actor for images.

At this point we have the original image prepared to be rendered. The last step will be the same as the previous case. The system will create a new renderer and will add all the actors, including the new image actor, to the renderer. Then, the camera is going to be reseted and the renderer will be added to the render window. Finally, the system will render the scene.
Chapter 6

EVALUATION AND RESULTS

The evaluation of the results is a crucial part of any project. It is decisive to analyse if the initial goals have been met. In this chapter we present the more relevant results of the application. Taking different data bases segmentations, we expose their visualization results. Moreover, we have evaluated some important factors of the application. Finally, we list some of the limitations that the application has currently.

6.1 Visualization of the results

The input of our application is a set of segmented frames. We have taken some segmentation video frames from known data bases and we are going to use them to test our application. These data bases are:

- Freiburg – Berkeley Motion Segmentation Dataset (FBMS - 59) [22][23][24]
- MPI – Sintel [25][26]

Furthermore, we have used some simple segmented frames from the Image Processing and Computer Vision Group from the UPF.

Our first goal was to generate a volume from the segmented frames, we tried with a simple set of frames and the final result is the following:

In the first example we can appreciate what we have been explaining in chapter 5. The resultant reconstruction consist of a volume and their outline. This outline is useful to situate the object in the frame. The frontal frame reconstruction corresponds to the last frame, thus, this video shows a round object (could be a ball) falling.

Figure 6.1: Example of input and output of our application.
Our second objective was creating an application enabling to interact with the 3D reconstruction. The following images taken from ‘alley_1’ [25] [26] show some of the views the user can get moving through the scene.

Figure 6.2: Frontal and initial view.

Figure 6.3: Semi-lateral view.

Figure 6.4: Up view.
The set used for this second example had 8 segmented frames. As we can see in Figure 6.4, we can visualize the transition from one frame to another. If we count these transitions, we can conclude that there are 8 frames as expected.

The interaction of two objects is also an important issue. Using a set of frames from ‘ambush_7’ [25][26] we can visualize and analyze the interaction between a hand and a torch trough time.

Moreover, the following example shows up to five objects in the same scene. We can observe that the visualization is still very clear.

The set of frames from this previous example was extracted from ‘cars_2’ [22][23][24]. The scene is recorded in a crowded road and recreates the interactions between some automobiles.
The editable visualization features where also some of the requirements coming from the Image Processing and Computer Vision Group. In the following example can be seen a change in the opacity’s hand from Figure 6.7.

The change of the opacity might lead in a better visualization. The user will be able to better interpret the interaction between objects through time.

Since the system initialize the objects colour through a colour matrix which only contains six different colours, when there are a lot of objects in the scene the user may need to change the objects colour to differentiate them.

The following example shows the initial generated scene and the final scene after changes in the objects colour and their opacity made by the user.

As seen in the example, thanks to the changes made by the user the interaction between the objects can be better appreciated. The user might differentiate each object with a colour and the interaction when two objects intersect is easily understandable over time.
The second edit window was video discretization. This window permits the user to choose the distance between frames. In the following example we can see the difference between a distance of 5, which is the default one, and a distance of 18.

![Video discretization](image1.png)

**Figure 6.10:** Visualization with a distance of 5 between frames.

![Video discretization](image2.png)

**Figure 6.11:** Visualization with a distance of 18 between frames.

The edition of this value might help the user to better visualize the trajectories of the different objects.

When there are a lot of moving objects in the scene is also important to visualize an object one by one. This can be achieved with the object visualization window which permits to choose the visualized objects. In the following scene there are a family of ducks taken from ‘ducks’ [22][23][24]. The user wants to analyse the behaviour of one of them, therefore they need to hide the other ones.
In Figure 6.14 we can see the selection of one of the ducks in the displaying window and the object visualization window. In figure 6.13 there are all the objects selected except for one in the object visualization window, therefore this is transferred to the scene and the pink duck is not shown anymore.
Finally, we need to visualize the results of frame visualization window. As mentioned this window has two modes of trimming. The first one selects a range of frames from the reconstructed ones. Taking the set of frames from Figure 6.6, we are going to select a new range of frames to visualize.

The new frame range is \([2, 4]\), meaning from the second frame to fourth one. In this new visualization of the scene we can see that there is an object that has disappeared. This is because this object only appeared in the first frame.
If the user continues moving the initial frame slider to the right we are going to visualize a new frame range \([3,4]\). In the following figure this new visualization can be appreciated:

![Figure 6.16: Visualization of the scene with a new frame range: \([3,4]\).](image)

The outline is also updated although it does not appreciate in the images.

The second mode of the window is the ‘individual frame’ which allows the user to go through every frame one by one. The utility of this mode might be analyzing the accuracy segmentation of the video comparing the segmented frame and the original one.

Following the previous examples, we are going to show the results of this mode of visualization.

![Figure 6.17: Visualization of the first segmented frame.](image)

This type of visualization will have no volume, so it is just taking the segmented frame and rendering it in the 3D environment.
Another important requirement from the GPI was the visualization of the original frame at the same time as the 3D reconstruction. This visualization will be available in both modes.

We can see how the original image fits exactly the outline. Moreover, the user can understand better the segmentations visualizing both images at the same time. In Figure 6.18, we can see why the green car was occluded, there are other cars covering it.

If we zoom in the previous example, the user can perceive the accuracy of their segmentations looking at the boundaries of the object segmentations. Since the blue car is occluded by a tree, its segmentation is more difficult to extract. Consequently, there are some parts of this car that should have been set as a part of this segmented object and they are not.
Taking the set of frames from ‘bamboo_1’ [25][26], we can show the results of the original image visualization in the frame range mode.

In this case, the original image will be placed at the same position as its segmented frame. For this reason, the original frame will move along the 3D reconstruction volume. In Figure 6.21, we can see the image printed at the middle of the volume. However, the image will be always rendered behind its segmented frame.
6.2 Evaluation

After testing our application with a wide range of frames sets, we have realised that there are some factors that will affect the proper functioning of the application. These are:

- Size of the frames.
- Amount of objects appearing in the video.
- Number of frames.

We have created some synthetic frames sets in order to test these three factors.

Examples of testing frames can be seen below:

![Figure 6.22: 1 object testing frame.](image1)
![Figure 6.23: 3 objects testing frame.](image2)
![Figure 6.24: 6 objects testing frame.](image3)

We have evaluated the execution time of our application, in other words, the time the app needs to generate the 3D reconstruction of frames. After testing the execution time in different frames sets, we have obtained the following results:

<table>
<thead>
<tr>
<th>Size</th>
<th>1 obj / 3 frames</th>
<th>3 obj / 3 frames</th>
<th>6 obj / 3 frames</th>
<th>3 obj / 10 frames</th>
<th>3 obj / 15 frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>640x480</td>
<td>1.40 sec</td>
<td>1.43 sec</td>
<td>1.71 sec</td>
<td>1.72 sec</td>
<td>1.93 sec</td>
</tr>
<tr>
<td>960x720</td>
<td>1.89 sec</td>
<td>2.41 sec</td>
<td>3.07 sec</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1280x960</td>
<td>2.69 sec</td>
<td>4.36 sec</td>
<td>5.42 sec</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1920x1440</td>
<td>5.53 sec</td>
<td>10.16 sec</td>
<td>13.59 sec</td>
<td>11.15 sec</td>
<td>13.03 sec</td>
</tr>
<tr>
<td>2560x1920</td>
<td>10.66 sec</td>
<td>20.26 sec</td>
<td>27.55 sec</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3840x2880</td>
<td>29.20 sec</td>
<td>58.27 sec</td>
<td>79.47 sec</td>
<td>55.43 sec</td>
<td>59.26 sec</td>
</tr>
<tr>
<td>5120x3840</td>
<td>63.91 sec</td>
<td>128.28 sec</td>
<td>175.51 sec</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.1 --: Execution time results.

From the table above we have generated some summarized graphics to extract some conclusions about the three factors we wanted to analyse.
From the first graphic, which compares the execution time depending on the frame image size, we can conclude that it is a factor to take into account. We can observe that the execution time is exponential over the image size. With a usual image size like 640x480 or 960x720 the execution time is about 1.5-2.5 seconds. In this case it is not going to be a problem for the user.

We have established a threshold (red line) in order to ensure the proper functioning of the application. We do not assure an appropriate app functioning over 2000 pixels.

As seen in the graph above, we can conclude that the number of objects also affects the execution time but when the image has a high resolution. The difference of seconds between frames in the usual image sizes is up to 3 seconds and 8 seconds as maximum in the image 1920x1440. For this reason, it is an important factor, but it should not cause problems in the functioning.
In this last graphic we have just taken three different sizes but it is enough to get some conclusions. The difference in the execution time between these three cases is so small (about 1-2 seconds) that it does not really matter the number of frames that compose the video. In the case the user takes a video of 200 frames could appear a relevant difference, but, in practice, the researchers are not going to take so many frames in the application usage.

In conclusion, the three factors that we were talking at the beginning are important to be taken into account, but they are not going to become a problem for the expected functioning of the application.

6.3 Limitations

As in all the applications, there are some limitations in our app that are worth to be described. It is important to remind that our application is aim to 3D reconstruct a set of segmented frames in order to visualize the interactions and the movement of the objects. For this reason, the segmented frames of the video need to be consecutive, i.e. frame $t$, frame $t+1$, frame $t+2$, and so on.

If we look at a set of segmented frames where the user has taken a frame every 10 frames we are going to get the following result.

Figure 6.27: Execution time graph depending on the frames.
The resultant visualization has no sense in terms of analysing trajectories and movements. We cannot understand what the rabbit is doing. Therefore, one limitation of our application is on the consecutiveness of the input segmented frames.

Although the user takes successive frames as their input, we can get an unwanted result if the movement of the object is really fast. Taking some frames from ‘temple_3’ [25][26] we have obtained the following visualization result:

We can observe the difference between the girl’s movement and the dragon’s movement. The girl is lowering the hand, and that can be perceived because of the velocity of her displacement. On the other hand, the dragon is flapping its wings so fast that there is no progression in the wings’ movement. We can understand that the dragon is approaching the girl but we have to suppose the movement of the wings between one frame and the other.
Chapter 7

CONCLUSION

Nowadays the use of technologies related to video manipulation is rising due to its important visual information such as motion and appearance, and even more for its crucial spatiotemporal interactions among objects.

Therefore, taking this concern into account, we have decided to develop a visualization tool for the analysis and automatic interpretation of a dynamic three-dimensional scene from a single video.

One of our main objective was to create an intuitive, interactive and powerful application to ease the visualization of a moving scene. From all the obtained results we can conclude that we have achieved it. Our application not only allows to visualize and analyse a 3D reconstruction of video frames but also to interact with the environment and modify visualization features in real time.

Moreover, it is important to bring out that this tool has been developed in an open source environment, using tools available in the market. Since the users of our application are computer vision researchers this was a key factor.

On the other hand, we are proud to announce that our application will be used in a research group in the university (GPI from Pompeu Fabra University) and has accomplish their specifications. Among all the requirements they asked, there is the possibility of editing the frame range visualization or the rendering of the original textured image into the volume reconstruction, like others competitive and licensed software available in the market do.

In conclusion, although there could be improvements in a near future, our application has become as powerful as we expected and with all the functionalities working properly.

7.1 Future work development

As we have mentioned along the project dissertation, there are some features that can be improved in the future. Some of these features are:

- The automatic labelling of objects values when the image is entered in RGB instead of the conversion into Grey Scale values.
- The automatic detection of background objects trough background subtraction algorithms usually used in computer vision.
- The possibility of entering 3D images, such as medical images, in order to visualize them like others visualization software already do.
Appendices
Appendix A

MARCHING CUBES

Step 1: Create a cube

As explained, we create a cube formed by pixels, 4 coming from the frame \( t \) and 4 coming from the frame \( t+1 \). Then, each of the 4 pixels defines a voxel. Each voxel has a value from 0 to 255 and, therefore, we have to choose an isovalue in that range.

![Figure A.1: Definition of the cube.](image1)

Step 2: Classify each voxel

Next step is to decide whether the pixel lies outside or inside the surface depending on the isovalue. We are going to classify the voxel according to the following rule:

- \( Value < isovalue \rightarrow \) outside (red colour in the examples)
- \( Value \geq isovalue \rightarrow \) inside (green colour in the examples)

Taking the following cube as an example we are going to show two resulting classifications depending on two different isovalues:

![Figure A.2: Cube example.](image2)
Cube vertices with values below the surface receive label zero and the ones above receive label one.

**Step 3: Build an index**

Since there are eight vertices in each cube and two states, inside and outside, there are $2^8 = 256$ ways a surface can intersect a cube. The algorithm creates a look up table enumerating these 256 cases. The table contains the intersected edges in each case.

However, Lorensen and Cline [12] derived three types of symmetries: rotational and reflective symmetries of the cube and sign changes of voxel. Thanks to them, the 256 cases were reduced to a set of 15 base cases, which can be seen in Figure A.4.

In the third step, matching cubes uses the binary labelling of each voxel to create an index and, thus, finds the case of the cube in the look up table.

The index is created as seen:

<table>
<thead>
<tr>
<th>v8</th>
<th>v7</th>
<th>v6</th>
<th>v5</th>
<th>v4</th>
<th>v3</th>
<th>v2</th>
<th>v1</th>
</tr>
</thead>
</table>

*Table A.1: Index building.*
Following the examples before the indexes for each cube will be:

**Figure A.5: Cube indexing.**

<table>
<thead>
<tr>
<th>Figure A.6: Index examples.</th>
</tr>
</thead>
<tbody>
<tr>
<td>11010000</td>
</tr>
<tr>
<td>11110011</td>
</tr>
</tbody>
</table>

**Step 4: Lookup edge list**

Once we have found the cube index, we search in the look up table the corresponding case and their triangles. Knowing that a cube has 8 vertices and 12 edges, Lorensen and Cline [12] created the following reference cube which is used to build the triangles formation.

**Figure A.7: Cube reference.**
Having found the index, we can know the triangulation by just looking at the lookup table:

**Index** → 10110001

- **Triangle 1**: e4, e7, e11
- **Triangle 2**: e1, e7, e4
- **Triangle 3**: e1, e6, e7
- **Triangle 4**: e1, e10, e6

**Step 5: Interpolate the edge location**

For each triangle edge, it finds the vertex location \( P \) along the edge using linear interpolation of the voxel values. Taking the consecutives vertex coordinates \( P1 \) and \( P2 \) from the cut edge and the scalar values at each vertex \( v_{p1} \) and \( v_{p2} \), the intersection point \( P \) is given by:

\[
P = P1 + \left( \frac{isovalue - v_{p1}}{v_{p2} - v_{p1}} \right)
\]

Finally, we need to compute the normals at each cube vertex using linear interpolation:

\[
N_x = v_{i+1,j,k} - v_{i-1,j,k} \\
N_y = v_{i,j+1,k} - v_{i,j-1,k} \\
N_z = v_{i,j,k+1} - v_{i,j,k-1}
\]

**Step 6: Go to next cell**

We have to repeat this procedure for each group of pixels in the image and each pair of frames in the video, i.e., for each voxel.

**Ambiguous cases**

The interpolation along each edge should always be in the same direction. If this does not happen, it can generate points that do not match exactly, due to the number rounded, and will not be merged correctly. This derives in some ambiguous cases because some of the possible combination of voxels may be countered in more than one way.

Ambiguous cases are: 3, 6, 7, 10, 12 and 13. We cannot choose one of ambiguous cases regardless of everyone else. Different techniques have been developed to solve this problem:

- **Marching tetrahedra**: subdivide voxels into tetrahedra with compatible neighbourhood. One of the disadvantages is that it generates isosurfaces formed by more triangles, so it is computationally slower.
- **Asymptotic decider:** consist in the evaluation of the asymptotic behaviour of the surface and then decide which cases join or break the contour.
- **Complementary cases:** this simple, but effective, technique adds complementary cases to the 15 original ones. This cases are designed to be compatible with close cases and prevent holes in the isosurface. It creates six complementary cases which correspond to the original cases: 3, 6, 7, 10, 12 and 13. The following figure shows how arbitrarily choosing marching cubes cases leads to holes in the isosurface and the complementary cases solution:

![Case 3 and Case 6](image)

*Figure A.9: Ambiguous cases which causes holes [16].*

![Case 3c, Case 6c, Case 7c, Case 10c, Case 12c, Case 13c](image)

*Figure A.10: Complementary cases [16].*

Marching cubes algorithm uses complementary cases as a solution for ambiguous cases. Isosurfaces represent only a single level within the data range. This means that only an isovalue can be chosen at once. As we have more than one object in our algorithm, this derives in two problems from Marching Cubes algorithm:

- It does not allow creating a threshold (isovalue) range.
- It only takes as inside voxels the values higher than the isovalue.

Knowing this, we have adapted the frames and created a 3D image for each object. We have solved the first problem executing Marching Cubes algorithm for each 3D image. Secondly, the next problem has been solved thresholding the images and erasing the objects brighter than the threshold object.
Bibliography


