Dynamic Rendering of Small Objects on a Tablet Depending on the Position of the User

Master Thesis

Vision, Graphics and Interactive Systems

Aalborg University, 2015

AALBORG UNIVERSITY

STUDENT REPORT

School of Information and Communication Technology
Aalborg University
Niels Jernes Vej 12a
9220 Aalborg Øst
http://www.es.aau.dk

Title:
Dynamic Rendering of Small Objects on a Tablet Depending on the Positionof the User

Theme:
Computer Graphics

Project Period:
Spring Semester 2015

Project Group:
VGIS

Participant(s):
Iskren Galentinov Vlaykov

Supervisor(s):
Martin Kraus

Copies: 2

Number of pages: 64

Date of Completion:
August 6, 2015

Abstract:

With the rapid pace with which the technologies are developing, it is hard for the classical museums to compete with modern media. They are in need of modernized technologies in order to keep the interest of their visitors and to provide them with enjoyable experience.
One way to achieve this goal is by using realistic looking virtual objects as a replacement of the real ones. Thus it will not only serve as a substitution when the real exhibits are unavailable due to maintenance, their frangibility, or if they are currently not present in the museum, but also will provide a possibility for more freely interaction.
The goal of this project is to create an application for a mobile device with high quality display, that will present to the users realistic looking objects depending on the position they are observing the device from. The method, which the head-detection of the observers is based on, is background subtraction between images captured with fish eye-lens camera. When the position of the head of the viewer is established, the camera representing the view-point of the user is translated accordingly in order to provide realistic feeling of observing a real object.

Iskren Galentinov Vlaykov

Abbreviation List

AMOLED	Active-Matrix Organic Light-Emitting Diode
CAVE	Cave Automatic Virtual Environment
DEM	Digital Elevation Models
FFD	Free Form Derformation
HSV	Hue-Saturation-Value
LCD	Liquid Crystal Displays
OLED	Organic Light-Emitting Diode
RGB	Red-Green-Blue
SDK	Software Development Kit
SfM	Structure from Motion

CD content

A CD is applied along with the report. The items that it contains are:

- Electronic copy of the report
- The main project application
- Auxiliary application used for calculating latency
- Application used for various visualizations during the implementation of the methods
- Matlab function for visualizing 3D point space
- Matlab script for visualizing exponential functions
- Result images

Contents

Chapter 1

Introduction

In order to keep up and compete with modern media, museums are trying to find and implement new pleasing ways of presenting their exhibits thus attracting and entertaining their visitors.

One direction in which can be worked to achieve this goal and which evoked the idea of this project, is implementing graphical application that creates the illusion of the presence of a virtual object by using view-dependent, photo-realistic rendering. This can be very helpful for enhancing the positive experience for visitors, and also in situations when given exhibit cannot be displayed due to cleaning, maintenance, currently unavailability or if it is fragile to exhibit.

A concrete example is an exhibition in Regional Historical Museum of Stara Zagora [1]. During a certain period of time some of the exhibits are lent to another museum and they are not available for display for the visitors. To compensate this absence the museum owners are presenting photographs of the missing objects (Figure 1.1), but they are not visually appealing and that's where 3D graphical objects can come in handy.

The use of compact mobile devices, having high-resolution screens with a relatively large dynamic range, is suitable for the purpose of detailed photo-realistic display of small objects. However, it is preferable the screen of the device to be large enough, in order visualized object to be displayed without many spatial constrains. This leaded to choosing of a tablet instead of a phone, although the created application can work on both types. Another requirement for the mobile device is to have front facing camera, in order to capture the user's position, so it can display the object according to it, creating the aforementioned view-dependent rendering, thus improving the realistic feeling of the virtual scene for the visitors.

It should be noted that although the project focus at application that supports museum exhibitions, the software can be used also in other areas like entertainment, education, etc.

(a) **(b)**

Figure 1.1: Examples of missing exhibits substituted with photographs.

Chapter 2

Problem Statement

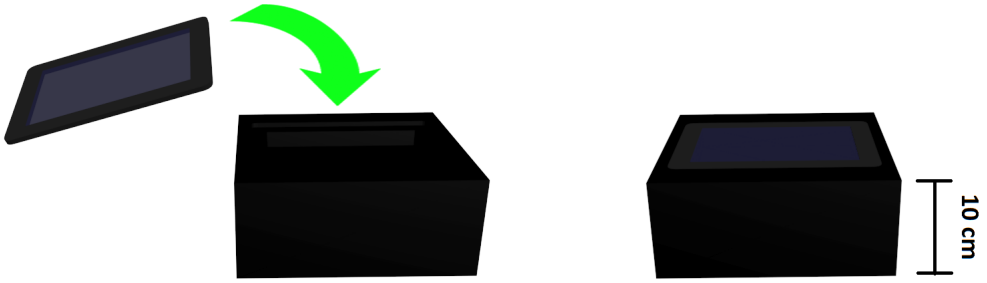
The aim of the project is to create sequence of methods and implement them together in a mobile application that renders realistic objects depending on the position of the viewer. It should be able to display external objects received from photo-realistic reconstruction software, correctly detect the head-position of the user and calculate the 3D position such that it corresponds to the current viewpoint of the user.

The application should respond fast to changes in the viewer position and provide good frames per second ration in order to keep the realism of the displayed scene. Also the device on which the application will run should be positioned in such manner that the viewers could believe that there is really an object behind its screen. How exactly this is done in the project will be discussed in details in the next Section 2.1

2.1 Placement Set-up

As already mentioned, in order to create the illusion of a presence of a real object behind the screen of the tablet, the tablet itself is put on top of a box with approximate height of 7-10 cm (Figure 2.1). This is high enough, so the viewer can assume that the box really contains a small object inside, thus increasing the realism of the application. Images representing steps of the creation of the box can be seen in Appendix A.

Having in mind that the application will exhibit objects in a museum, it is assumed that the box containing the device will be placed on table or a stand above the ground, with its screen pointing up, and the visitors will approach it from different directions (Figure 2.2 a). Therefore the application detects that direction, by estimating the position of the viewer's head, in order to display the object in accordance to it. In addition, it should be noted that it is assumed that all of the user are at constant height above the device - 50 cm (Figure 2.2 b).



(a) (b)

Figure 2.1: How the tablet is placed on top of a box with a height of 10 cm;

(a) (b)

Figure 2.2: (a)The application detects the head of the approaching person; (b) All the viewers are assumed to look at the tablet screen at constant height of 50 cm.

Chapter 3

Related work

This chapter presents ideas, techniques and projects which the current project is based on, or have similar scope. The chapter itself is divided into three sections depending on their focus. The first one (Section 3.1) shows projects similar to the current one in the sphere of photo-realistic display of virtual objects. It mainly covers the idea of virtual exhibitions and museums. Section 3.2 covers the techniques on which the software for photo-reconstruction of objects is based. Finally Section 3.3 explains the method of off-axis projection which is used for displaying 3D objects depending on the position of the user or the viewer.

3.1 Photo-realistic Display of Virtual Objects. Virtual Museums

The aim of Virtual Museums is to explore the use of virtual reality technologies and realistic object rendering to help both public visitors and researchers in viewing exhibits, visualizing artifacts and learning their use and characteristics. In the paper by Lepouras and Vassilakis [2], which proposes the use of 3D game technologies for developing affordable and easy to use virtual scenes (Figure 3.1), the authors are explaining the benefits of using 3D graphics and virtual reality in such projects.

Exhibits in museums that cannot be present because of reasons like being fragile, lack of space or if being missing due to cleaning or maintaining, can be displayed to audience using virtual reality applications. They can also help in cases when there is a need for visualizing and simulating environments, scenes, species, constructions or objects that no longer exist, have been extinct, are only partially preserved or cannot be easily visited (Figure 3.2).

On other hand graphically generated exhibits can be observed from various viewpoints or even tested or manipulated. Virtual environment can be build up for visitors where they are able to learn by exploring, get familiar with the way of use of the objects or even assemble and disassemble them to pieces (Figure 3.3). Technologies such as haptic feedback may enable visitors to touch and feel valuable objects, which originals are not accessible. In addition people with vision problems can take advantage from the technologies in order to sense and understand the exhibits in alternative ways.

In her paper Rizvic [4] introduces the readers to a example of virtual exhibits (Figure 3.4)

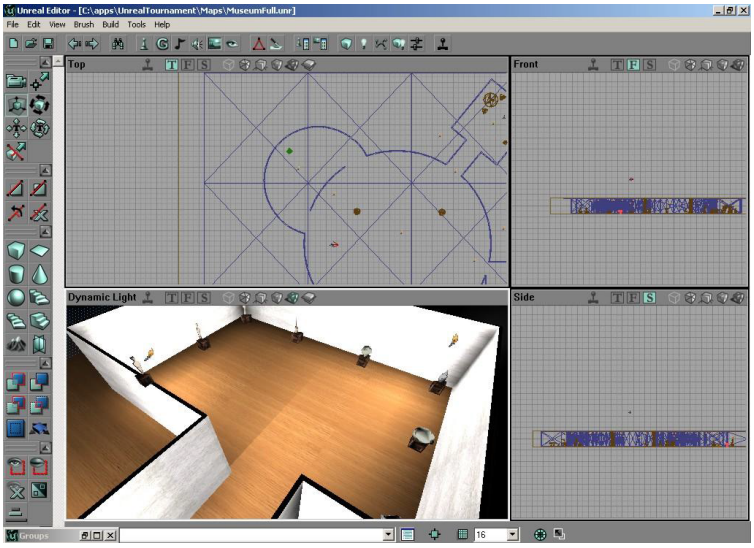


Figure 3.1: Example of game engine editor used for creating museum environment *Source: Lepouras and Vassilakis [2]*

(a) **(b)**

Figure 3.2: Virtual museum exhibits: (a) Exhibit set of ancient vases; (b) The workshop of Pheidias, in Olympia; *Source: Sideris and Roussou [3]*

where the authentic experience of the visitors is not only created by realistic looking objects but also is enhanced by using digital storytelling. They claim that interactive applications naturally request the storytelling to become interactive as well.

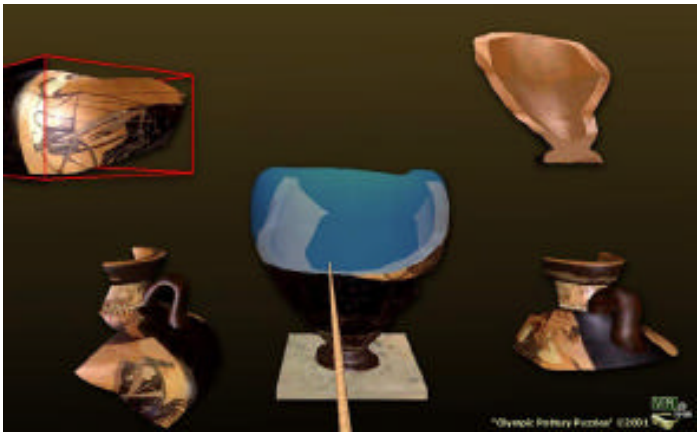


Figure 3.3: Museum visitors are able to interact with the virtual scene by assembling the 3D pieces of an ancient vase; *Source: Sideris and Roussou [3]*

Figure 3.4: Virtual Museum of Bosnia and Herzegovina Traditional Objects, presentation of Ibrik and Ledjen. Left to right: text, photos (top); movie, interactive 3D models (bottom). *Source: Rizvic [4]*

Examples of some well-known virtual museums or exhibits are presented in the following list:

1. Mummy: The Inside Story in the British Museum [5]
2. Leonardo: the Ideal City [6]
3. The Virtual Museum of Arts El Pais [7]
4. 3D Encounters: Where Science meets Heritage Exhibition [8]

Similarly to the concept of the current project, Wang et al. [9] are using augmented reality in order to visualize graphical objects or scenes combined with real objects. The idea implemented behind the project system is to retrieve related information from a server database about given painting and displays it as virtual content overlaid on top of the actual painting image (Figure 3.5). The results that they have acquired experimentally on real-world exhibitions have demonstrated the effectiveness of the proposed approach, leading to suggestion that such method can be used as a replacement of real exhibit.

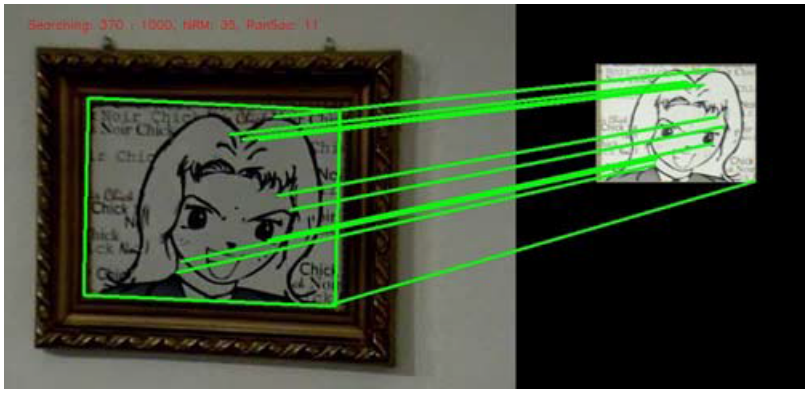


Figure 3.5: Real museum test, with image correspondences and recovered pose displayed; *Source: Wang et al. [9]*

3.2 Techniques for 3D Photo-reconstruction

In their article Álvaro Gómez-Gutiérrez et al. [10] are making comparison between two photo-reconstruction methods, while creating dense point clouds and high resolution DEMs (digital elevation models) of the Corral del Veleta rock glacier in Sierra Nevada (Spain). One of the methods is the fully automatic 3D photo-reconstruction (FA-3D-PR) method which is based and used in the mentioned in Section 4.2.1 software 123D Catch for constructing virtual objects from photographs. This method uses the Structure from Motion (SfM) and Multi-View Stereo algorithms and only needs oblique images of the desired object as an input.

In general, the fully automatic photo-reconstruction procedures have been continuously researched in the recent years and have been developed in several software packages similar to 123D Catch [10]:

1. ARC3D
2. Bundler and PMVS2
3. CMP SfM
4. Photosynth
5. VisualSFM

As already mentioned the way of work of the 123D Catch application is based on the simultaneous use of SfM and Multi-View Stereo techniques. These two types of reconstruction methods are discussed in the following subsections.

3.2.1 Multi-view Stereo Algorithms

The goal of multi-view stereo is to reconstruct a complete 3D model from set of images, that are taken from specified camera viewpoints. These algorithms have been improving rapidly over the last few years, leading to the development number of high-quality techniques.

An approximate separation can be made, classifying the multi-view stereo algorithms into four classes. The first class consists of algorithms that first extract and match a set of feature points and then fit a surface to the reconstructed features. The steps involved in the method can be summarized as follows (Figure 3.6):

1. Extract features
2. Get a sparse set of initial matches
3. Iteratively expand matches to nearby locations
4. Use visibility constraints to filter out false matches
5. Perform surface reconstruction

In the second one a cost function on a 3D volume is initially computed and then a surface is extracted from the computed volume. Voxel coloring algorithm is an example for this class. It works by fragmenting the 3D scene into small volume elements, called voxels, and reprojecting each voxel back onto the image set. Then the variance between the voxels in the different images is compared. If it is low the voxels are colored to the according colors. In other case they are set to black.

Figure 3.6: From left to right: a sample input image; detected features; reconstructed patches after the initial matching; final patches after expansion and filtering; polygonal surface extracted from reconstructed patches. *Source: Furukawa, Ponce [11]*

Space carving technique can be given as an example for the third class of algorithms. They work by iteratively evolving a large initial surface to decrease or minimize a cost function. The algorithm can be presented by the following steps (Figure 3.7):

1. Initialize to a volume V containing the true scene
2. Choose a voxel on the current surface
3. Project to visible input images
4. Carve if not photo-consistent
5. Repeat until convergence

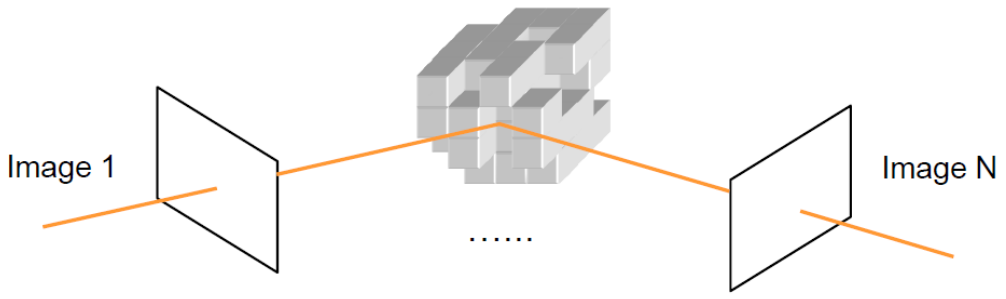


Figure 3.7: Representation of the space carving method. *Source: Fergus [12]*

The last class is image-space methods that compute a number of depth maps. In order tho ensure a single consistent 3D interpretation of the scene, common consistency constrains are enforced between the calculated depth maps, or as an alternative these maps are merged into a 3D scene as a post process.

3.2.2 Structure from Motion

Structure from Motion is a range imaging technique that estimates three-dimensional structures from two-dimensional image sequences that can be corresponding to each other because of presence of local motion.

Humans perceive a lot of information about the three-dimensional objects present in their environment by moving through it. When the observer moves and the objects around him move, information is obtained from images perceived over time. The Structure from Motion method is based on the same presumption. It tries to find correspondence between sequential images, by determining features such as corner points and track them from one image to the next.

The methodology of the SfM technique is explained by Haming and Peters [13], by presenting functional pipeline of the reconstruction method (Figure 3.8).

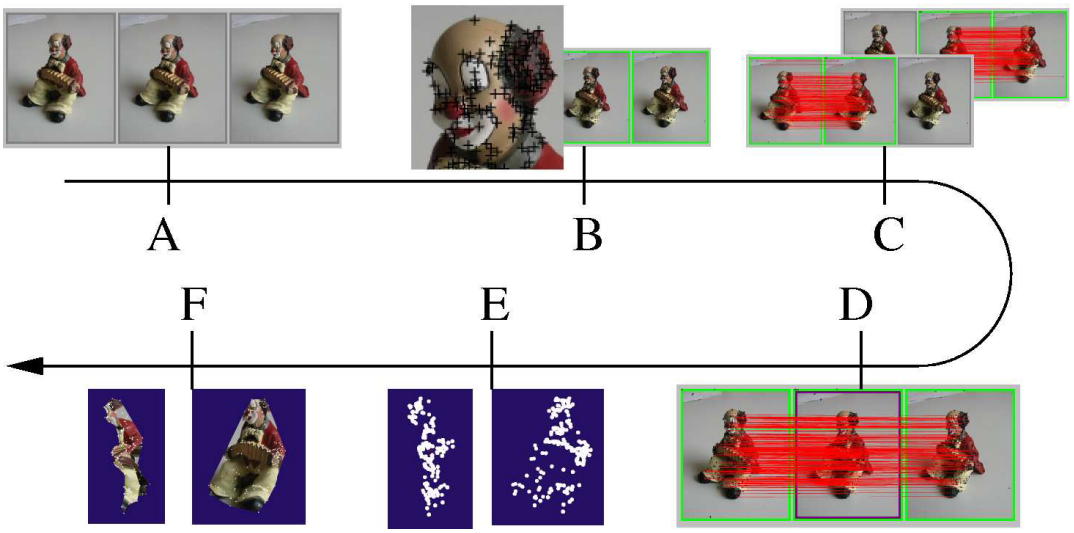


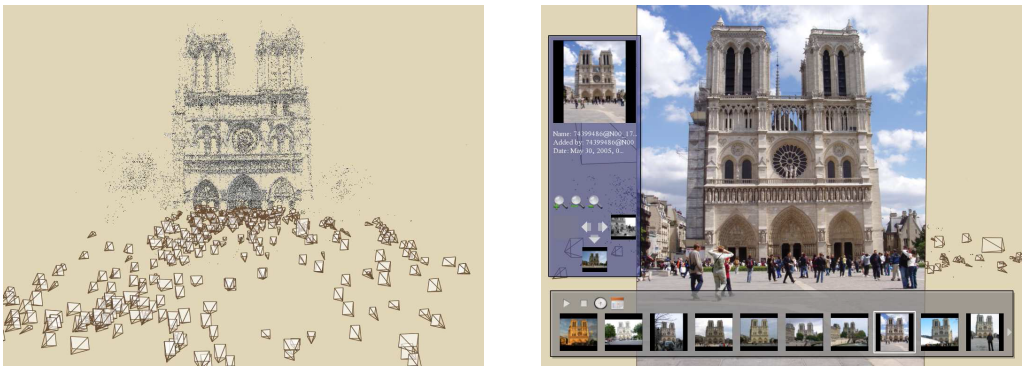
Figure 3.8: Methodology pipeline of SfM reconstruction method. A) Select input images; B) Feature detection; C) Feature matching D) Filtering (optional); E) Metric reconstruction, visualized as point cloud; F) Final object reconstruction, visualized as textured model. *Source: Haming and Peters [13]*

As always the reconstruction processes starts by capturing a number of photographs of the same object. From this set of images a single features present in more than one image are extracted. Next, in order to reconstruct 3D points of the desired object directly, the cameras positions and the position of the detected features in space should be determined. This can be achieved by intersecting the rays from the camera centers through the feature points of one particular correspondence. This triangulation is done by using only the images, because the parameters of the cameras are unknown. Having the information about the correspondence between the image feature points, it is sufficient to reconstruct point cloud representing the object's surface. This is due to the fact that the estimated correspondences should follow certain geometrical constraints. It should be noted that the authors of the paper have included one extra step - filtering of the matches, which provide higher robustness, but

can be omitted in the general SfM technique.

3.2.3 Example of a Photo-reconstruction Project

An actual illustration of simultaneous usage of the two types of photo-reconstruction techniques described in the Section 3.2 is the Photo Tourism project [14]. Its idea is to present a system for interactively browsing and exploring large unstructured collections of photographs, and in the same time to allow the user to experience full 3D navigation and explore the all the images in the collected set, and their world geometry (Figure 3.9). The system makes the development of photo tours of different historic locations easy, and in the same time provides additional information such as overhead maps (Figure 3.10).



(a) **(b)**

Figure 3.9: (a)Reconstructed 3D points and viewpoints; (b) Photo explorer for browsing the images; *Source: Snavely et al. [14]*

The goal behind the usage of photo-realistic rendering in the aforementioned system is to evoke a natural sense of presence in the 3D scene, that will allow evolving of the virtual tourism of the world's interesting and important sites.

Figure 3.10: Example of an overhead map *Source: Snavely et al. [14]*

3.3 Off-axis Perspective Projection

Perspective projection is general and well-understood part of 3D graphics and usually does not require too much attention from the graphical programmers. In most of the cases the only things needed, for creating a perspective projection with OpenGL library, are specifying near and far clipping plane and selecting field of view, and then calling the function **gluPerspective** or **gluFrustum**. But when using these functions few assumptions are made. For example **gluPerspective** assumes that the user is positioned perpendicularly to the screen and looking to its center, while **glFrustum** function assumes a perspective rooted at the origin of the user-view and a screen lying in the XY plane.

Because of these assumptions the functions can rely on using the method of on-axis projection that refers to camera positions which lies on the axis through the center of the view plane and orthogonal to it. In general this method is sufficient and suitable for majority of the cases. In other hand, in virtual reality applications in which user-tracking, stereoscopic viewing or multiple screens are used for creating the graphical scenes this method is not applicable.

Examples of such virtual reality set-ups or applications are the CAVE (CAVE Automatic Virtual Environment)[15] and The Varrier Autostereoscopic Virtual Reality Display [16].

The CAVE actually is a theater made up of three rear-projection screens for walls and a down-projection screen for the floor, as shown (Figure 3.11). Actually the projection plane positions correspond to the locations of these walls. The goal of the system is creating reality or scientific visualization with high definition.

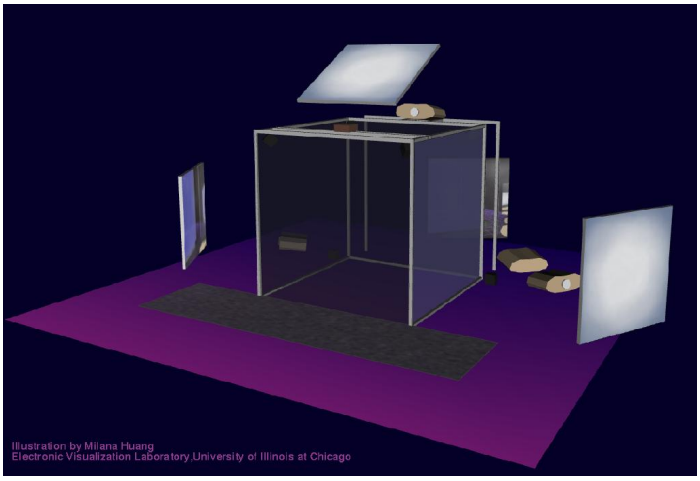


Figure 3.11: CAVE Illustration by Milana Huang, University of Illinois at Chicago. *Source: https://www.evl.uic.edu/pape/papers/idesk.cg.may97*

Another virtual technology shown on Figure 3.12 is the Varrier. Its display contains of a 12 x 5 array of LCD screens arranged in a 180 degree arc, ten feet in diameter. Each LCD displays has a resolution of 1600 x 1200 pixels, with a parallax barrier affixed to the front, giving auto-stereoscopic viewing, meaning that user can experience 3D viewing without specialized 3D glasses. Both of the mentioned virtual reality set-ups use multiple screens and

the user is able to move freely in the space, meaning that the origin of the perspective view is impossible to remain in the center of the projection screen.

Figure 3.12: The Varrier Autostereoscopic Virtual Reality Display *Source: Kooima [16]*

In his paper Generalized Perspective Projection [16], Kooima explains why the already mentioned commonly used functions fails when used in such virtual reality applications, and draws a comparison between on-axis and off-axis projection.

As already mentioned, because of the fact that user is free to move in the space, when using virtual reality graphical applications, the view position does not remain centered upon any of the screens - therefore the **gluPerspective** function fails. In other hand, because the display wraps around the user, most screens do not lie in the XY plane and the **glFrustum** also function fails. So for such set-ups a more generalized perspective projection should be formulate - the method of **off-axis** perspective projection.

By using on-axis projection, the defined position of the user's eye relative to the screen is in its center (Figure 3.13). The line drawn perpendicular to the screen strikes it directly in the middle. This point can be referred as the screen-space origin. In that case the pyramid-shaped volume is perfectly symmetric and has the screen as its base and the eye position as its apex. This is exactly the type of perspective projection received by using **gluPerspective**.

However, if the position of the eye is moved away from the center of the screen a new situation is at presence. In that way, the view-frustum is no longer symmetric, and the line from the eye does not strike the screen in the middle (Figure 3.14). Similarly to the function **gluFrustum**, distant parameters can be estimated, showing the left, right, bottom, and top frustum extents. They may be considered as the distances from the screen-space origin to the edges of the screen.

But the functionality that actually makes the difference from the **glFrustum** is the usage of transformation (rotation) matrix in order to rotate the projection screen depending on the position of the viewer's eye. Thus it does not lie in the XY plane and shows different images in correspondence to the viewer position. If that is not implemented all the screens of the

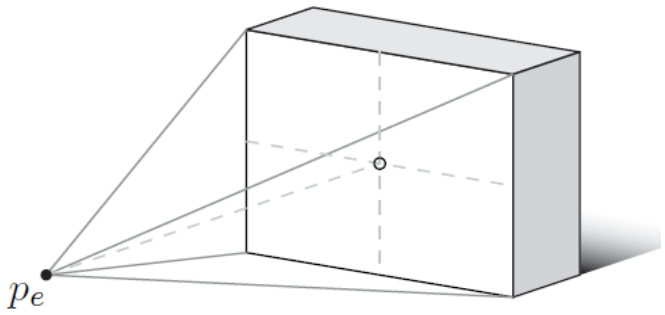


Figure 3.13: An on-axis perspective projection, with point **pe** denoting the position of the viewer's eye *Source: Kooima [16]*

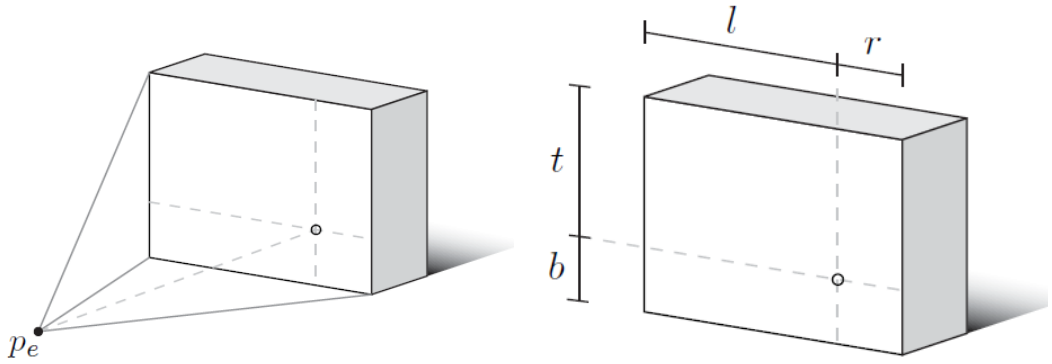


Figure 3.14: (a)An off-axis perspective projection, with the eye position **pe** and screen-space origin moved away from center; (b) The left, right, bottom, and top extents of the perspective projection, defined as distances from the screen-space origin; *Source: Kooima [16]*

Varier would show nearly the same limited view of the virtual scene.

Chapter 4

Materials

This chapter presents the device chosen for the purpose of the project in Section 4.1, along with the fish-eye lens used as an addition to the camera of the mobile device. In Section 4.2 the software products that have been used for the application have been briefly discussed and their main functionalities, important for the purpose of the project, explained.

4.1 Selected Devices

As already mentioned in the Introduction (Chapter 1) for better visualization of the desired objects, a tablet with high resolution and relatively large dynamic range is in need. Actually the dynamic range describes the ratio between the maximum and minimum measurable light and color intensities, meaning that such displays provide more vivid and realistic images.

One type of displays having such characteristics are the so called OLED displays. OLED stands for organic light-emitting diode and the way they work is without using a back-light [17]. Thus the OLED displays can provide images with deep black level, enabling greater artificial contrast ratio (both dynamic range and static, measured in purely dark conditions). The OLED colors appear correct and unchanged, even the viewing range approaches 90 degrees, providing wider observation angle in comparison with liquid crystal displays (LCDs). In addition the OLED display itself can be thinner and lighter and in a dark room condition can achieve higher contrast ration than the LCD.

As a device of choice for this project is selected the Toshiba Excite 7.7 tablet (Figure 4.1). It has 7.7 inch AMOLED (active-matrix organic light-emitting diode) display with resolution of 1200x800 pixels. Its operation system is Android 4.0.3 Ice Cream Sandwich and it has 5 Megapixels rear camera and 2 Megapixels front-facing - the one used for the project.

Other devices that have similar characteristics and have been considered as suitable option or substitution for the Toshiba tablet are shown in the list below:

- Samsung Galaxy Tab S 10.5
- Samsung Galaxy Tab S 8.4
- Fujitsu Arrows Tab F-03G

Figure 4.1: Toshiba Excite 7.7 Tablet Device *Source: https://technicalbigbang.wordpress.com /tag/new-tablet*

- Dell Venue 8 7000
- Samsung Galaxy Tab

In order the project application to cover wider area a mountable fish-eye lens is attached to the front-facing camera of the tablet. It is easy to acquire and not expensive, and in general it can be attached to the cameras of all the devices independent of their type. The lens used in the project is XCSOURCE 180 Degree Fish Eye Lens and is shown on Figure 4.2.

4.2 Used Software

4.2.1 123D Catch

The photo-reconstruction software used for this project is **123D Catch** developed by **Autodesk**. It is a free application (both desktop and mobile) that allows users to recreate realistic looking 3D models by uploading set of images of the desired object. The methods that the software is using is described in Section 3.2.

It is important that the object must remain static and not to be moved out of their initial position in all of the used images. Another restriction is that 123D Catch cannot handle transparent and/or reflective objects. In addition the pictures should not be taken while using flashlight that illuminates the objects, and in general it is good that the lightning condition to be consistent [18]. All these restrictions are needed due to the fact that the software searches for matching features in the photos which, if the aforementioned conditions are present, are distorted, or unique only for single photo, thus becoming unusable.

Figure 4.2: The fish-eye lens used for the project *Source: http://www.xcsource.com/p_detail.php?id=1185*

123D Catch software operates based on the cloud computing methodology. Users upload the images on the servers provided by Autodesk and execute all the needed operations and calculations for the object reconstruction there. Thus the users' computers or mobile devices are not overloaded, the computation is faster and the users even can turn off the application and use their devices for another purpose, while waiting for the process to be completed.

Once the application is done with constructing the object, the user is provided with the resulting model and various functionalities for processing it. One of them is the possibility the users to see the object from exactly the same viewpoints as the uploaded pictures present. These viewpoints put the images as a background behind the object and use them as a reference (Figure 4.3). Furthermore the model can be rendered transparent to a percentage of choice, thus allowing the users to make comparison between the created object and its image on the particular photo.

However, sometimes not all the pictures can be initially used for the reconstruction of the object due to inability of the software to create or match unique features between some of the images leading to gaps or inconsistencies in the reconstructed models (Figure 4.4). In that case the object is created from smaller set of only the usable photos, but the users have the possibility to stitch the rejected images manually.

This is done by selecting unique features (edges, corners unique texture parts) between the usable and the rejected images (Figure 4.5). The minimum number of required reference points is 4 but users can select more in order to receive better results. Once they are done and satisfied with choosing of the reference points the images are once more uploaded to the cloud server and the object is newly reconstructed.

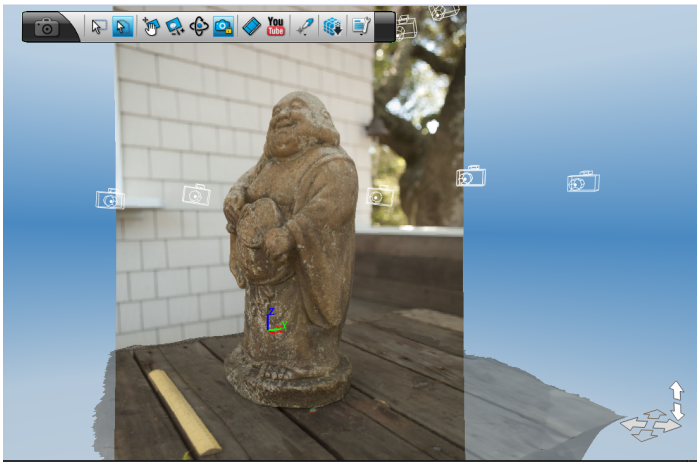


Figure 4.3: Half-transparent model of an object seen from an image viewpoint. The image itself is put in the background as a reference

Figure 4.4: Inconsistent reconstructed method due to rejected not usable images. They are shown with yellow exclamation mark in the image gallery

123D Catch also provides tools for changing the number of the object polygons, in order to receive models with higher or lower quality, or removing the unwanted or unnecessary ones. When the users are satisfied with the look of the reconstructed object, they can export it for external use or manipulation as OBJ, FBX or DWG type of files.

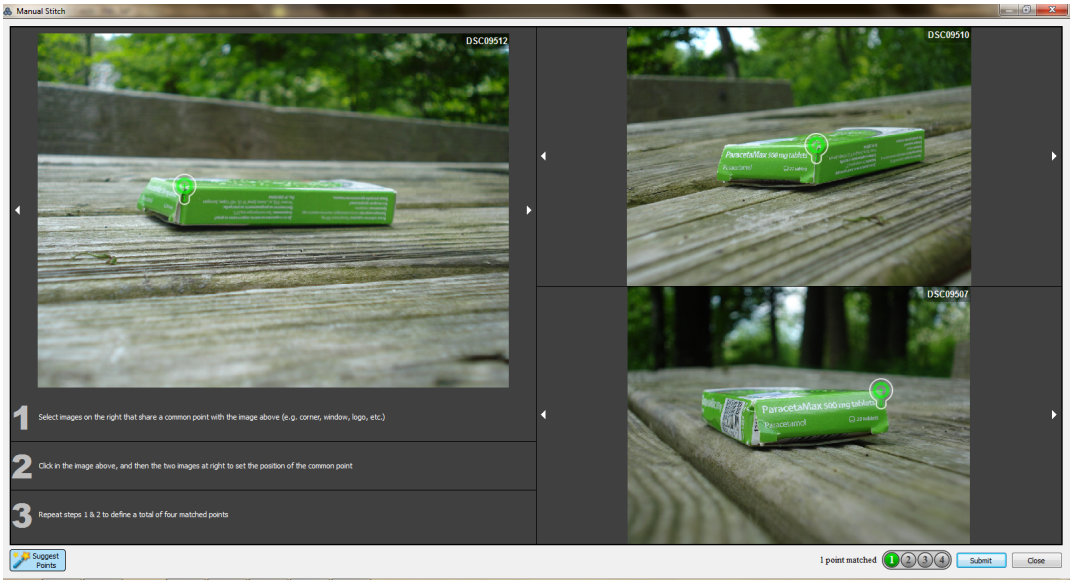


Figure 4.5: The Manual Stitch window. On the left side is the rejected image and on the right usable images that can be used for selecting features. When the user select features that are recognizable in the corresponding images they are colored green

4.2.2 Unity

The project application is programed using one of the commonly chosen software for the purpose of creating graphical applications - textbfUnity developed by Unity Technologies. Mainly its usage is aimed in the sphere of entertainment - creating games and special effects, but there are also many applications which serve purpose of object visualization, 3D modeling and architectural software.

Programming of graphical mobile applications with Unity is very similar to the developing of desktop ones, so the computer graphics programmers are not required to have additional knowledge in the programming of mobile applications. It uses scripts written in Java or C# to control the parameters and/or behavior of 3D objects (common Unity objects such as cube, sphere, etc. or more sophisticated imported objects), cameras or lights.

The only thing needed in order to create an Android mobile application with unity is to install the latest Android Studio and SDK - the current version used in the project is 1.1.0 Build 135.1740770. Next thing in order to create mobile application from existing Unity project is just to select the platform to be Android, connect the mobile device with the computer, and click Build and Run from the File drop-down menu. Thus Unity will aromatically crate the needed application .apk file, will upload it to the device and will run it when the upload is completed.

In addition some of the settings of the mobile application can be set up or changed by using Unity menu called Player Settings found in the Build Settings tab (Figure 4.6). Some of the common settings that can be adjusted are the name and the developer of the application, its icon, the orientation which it will use when run on a mobile device, the Android version,

which Graphic library to be used and many others.

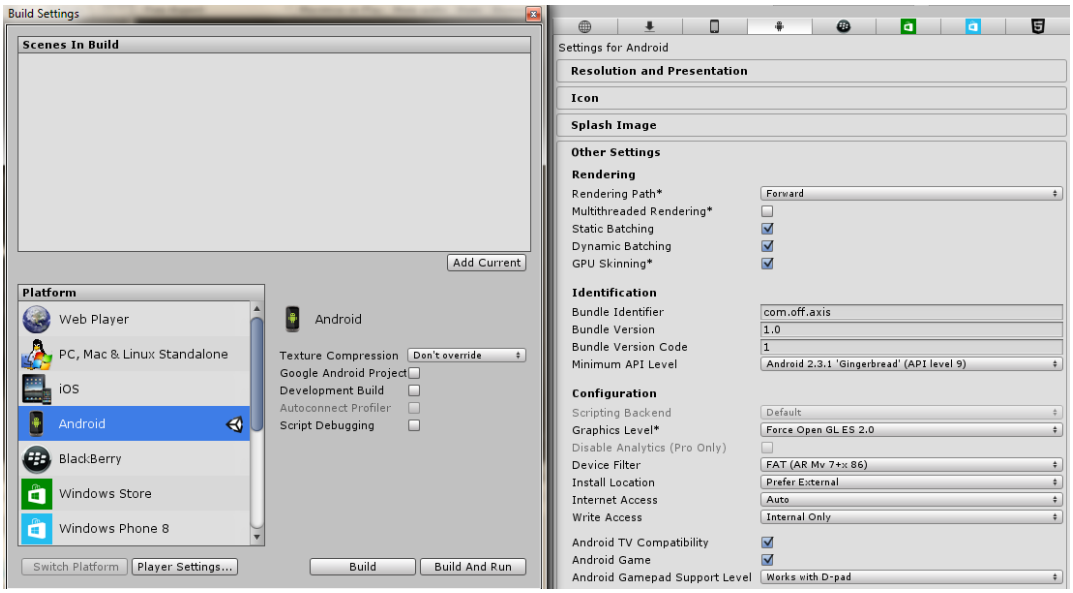


Figure 4.6: The Build Settings and Player Settings tabs in Unity containing the application parameters

Chapter 5

Methodology and Project Design

This chapter will thoroughly present and explain the steps included in implementing the project application. The whole process is visualized on Figure 5.1. It can be said that it is separated in two branches. The first one (the left one on the figure) is executed only once and it aims at creating and arranging the 3D scene. Initially the 123D Catch software is used in order to create realistic looking models of the desired objects. Next these objects, along with the projection plane representing the device display, are positioned in the scene using their real measurements. The final step in that branch is to create a camera that is a representation of the view-point of the user and apply the off-axis script to it.

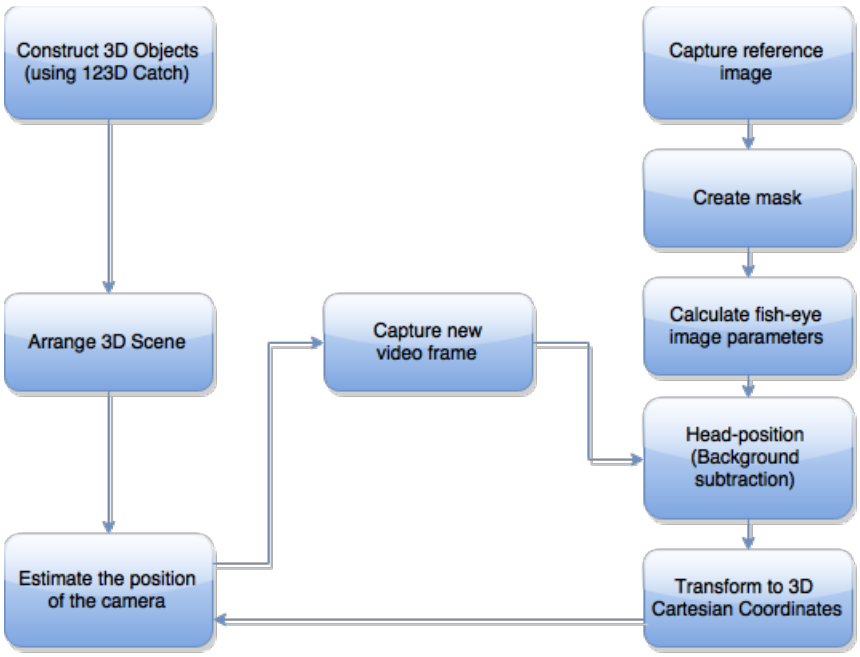


Figure 5.1: Overview of the methodology of the project application

The aim of the other branch of methods is to estimate the correct viewer position and move the camera accordingly. Initially an image, where no users are present in it, is captured and used as further reference. Next a mask representing the fish-eye image, is calculated from it and used to remove unwanted and unneeded areas from the future captured frames. In addition it is used for calculating the fish-eye image parameters - radius length and center coordinates, which are needed for the further calculations.

When the reference image is established and all the aforementioned calculation done, the next thing is to capture new frame and perform a background subtraction in order to detect the position of the head of the viewer in it. Having these 2D positions few transformations are used in order to translate them into 3D Cartesian Coordinates and use them to change to position of the camera in the scene. This is the end of the method cycle so a new frame is captured and background subtraction is again performed and new camera positions calculated. This process is repeated for each newly captured video frame, thus keeping the view-point updated according the users position.

The implemented methods are further explained as follows: Initially in Section 5.1 the process of creating the photo-realistic objects and their placement in the 3D Scene will be presented, followed by explaining of the use of Unity WebcamTexture for receiving and manipulating the image sequence from the device camera (Section 5.2). Next Section 5.3 will show how the fish-eye lens parameters are calculated. The methods for head-detection are discussed in Section 5.4 followed by the methods for calculating the exact 3D position of the camera in the scene in Section 5.5.

5.1 Object Creation and Scene Set-up

The idea of the project is to present small in size objects because it should be believable that they can fit in the box as the set-up describes in Section 2.1. In addition objects with small height are looking better than high objects when using view-dependent rendering.

5.1.1 Photo-reconstruction of the Objects

For the application have been reconstructed models of, as already mentioned, small not transparent and not reflective objects such as rock, candy, pen and a small cube (Figure 5.2). A try have been made to reconstruct a coin, but although not very reflective it haven't presented good results.

For each separate object set of images counting between 15 and 30 were made from various directions capturing all the sides of the objects (except the bottom side which was used as base). The majority of photos were accepted by the 123D Catch software defining the unique features of the objects, and only on very few images there was need of manual stitching.

Normally, when the reconstructed models are received they contain unwanted parts - for example the table where the objects lie or parts of the background scene. Therefore these parts are removed manually by selecting and deleting the extra polygons, leaving only the object of interest (Figure 5.3).

(a) **(b)**

Figure 5.2: Some of the reconstructed objects: (a) Rock; (b)Candy

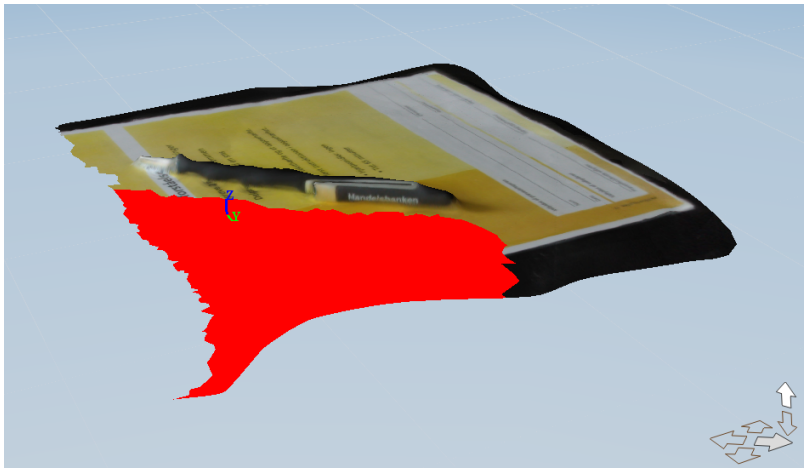


Figure 5.3: The process of selecting (red area) and deleting the unnecessary object polygons

When this process is done the final step is to export the models in order to be used in Unity. Both types .OBJ and .FBX are suitable for use but the .FBX was chosen for the purpose of the project because when imported in Unity as whole object, while the .OBJ file is separated in smaller parts connected in one. Also it should be noted that when exporting the model in both ways a material .MAT file is also created along, keeping the object texture information.

5.1.2 Objects Scaling and Arrangement of the Scene

When creating the 3D scene in Unity an effort is made to represent the real world set-up (Section 2.1) as correct as possible. In Unity it is not so hard because 1 unit corresponds to 1 meter in real world.

So the scene is arranged having the camera, which represents the user viewpoint, to be 50 cm directly above the projection plane, and the object that is rendered around 2-3 cm bellow it (Figure 5.4). The dimensions of the tablet display are measured and the projection screen is shaped according to them - 16,5 cm in width and 10 cm in height. Thus the viewpoint is representing the real view as if the user is looking through a window in the box.

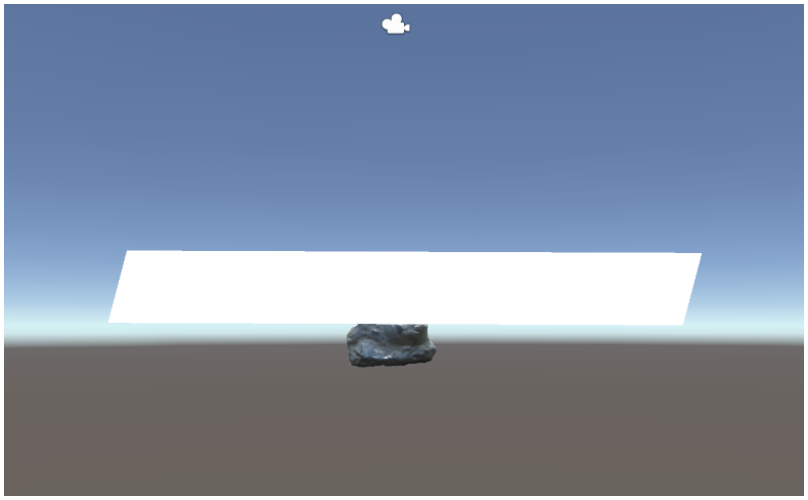


Figure 5.4: Arranging the 3D scene. The camera is set 50cm above the project plane, while the rendered object (the rock model) is approximately 3 cm below it

Having in mind the purpose of all the items in the scene to represent the real measurements, the imported photo-reconstructed objects should also be scaled to their real sizes. There are two ways of doing this. The easier but not so accurate one is by importing the object in Unity and manually scaling it to the desirable size using cube object as a reference. This reference is needed because when imported the object does not have real measurements and the scaling is done by percentage.

The second method is more accurate but in the same time it is more time consuming and requires additional modeling software. The one used in the project is the Autodesk 3D Studio Max. The scaling method however is similar - again uses cube with defined measurements as a reference - but this software provides very accurate functions needed for the scaling.

Initially the reconstructed object is imported and a box with the same size as the real object is also included in the scene (Figure 5.5). Next FFD (Free Form Deformation), having 2x2x2 control points, is applied to the imported object. These points are selected in pears and aligned with the sides of the reference cube in all the three directions, using the Align Tool (Figure 5.6). When all the control points are aligned with the all the cube sides the

object is scaled to the desired size and can be exported again as .FBX file and used in Unity(Figure5.7).

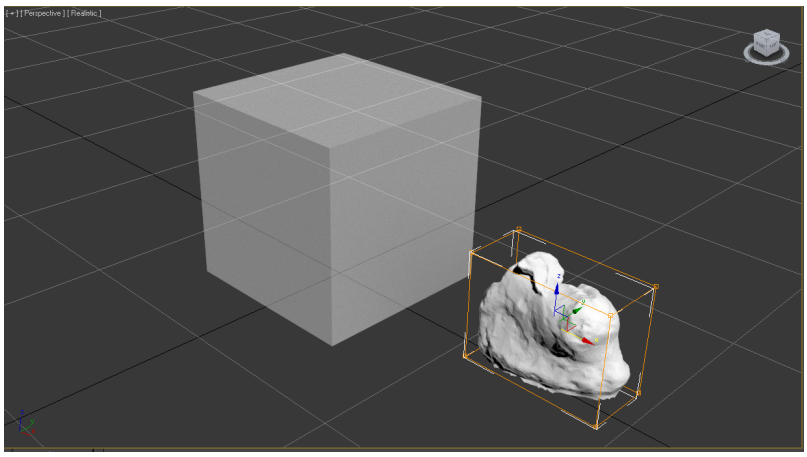


Figure 5.5: The imported rock object surrounded by FFD control points and the cube with the desired measurements

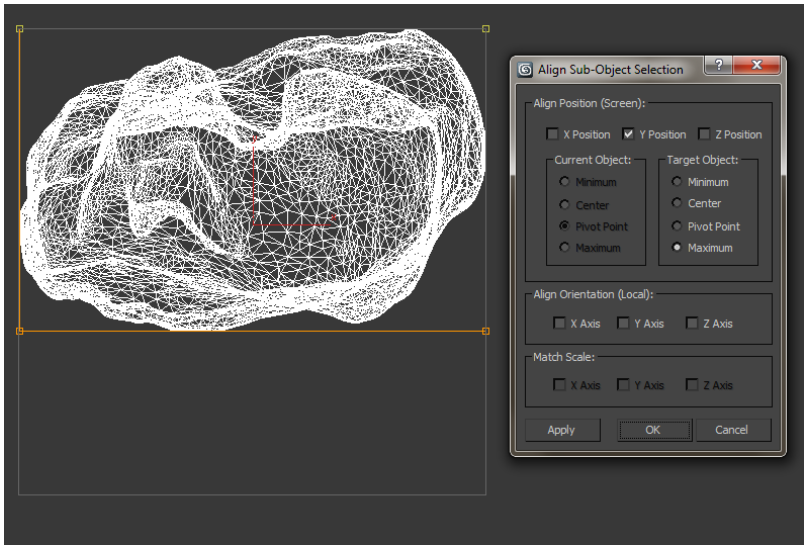


Figure 5.6: The process of aligning pairs of control points with the cube sides in the three directions

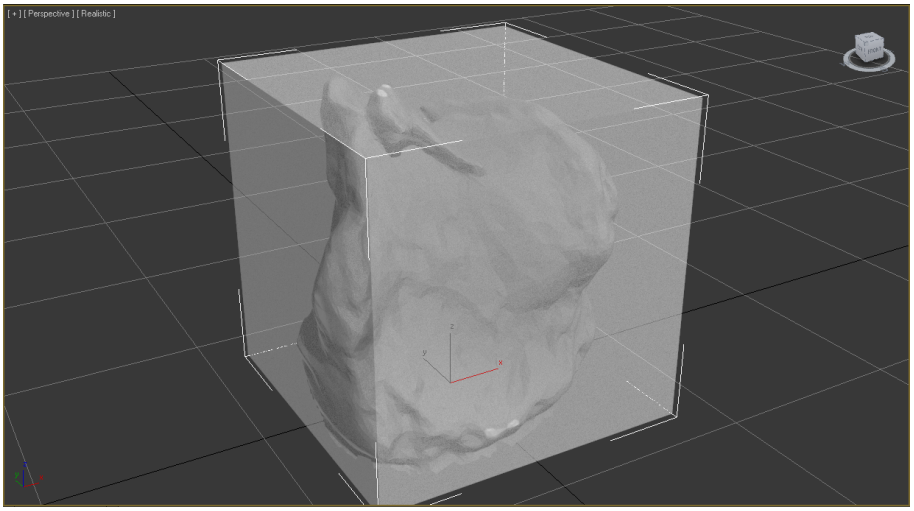


Figure 5.7: The final result after the scale process is completed

5.2 Using WebcamTexture

Unity provides a special texture class called textbfWebCamTexture. The members of this class are textures onto which the live video input is rendered. WebCamTexture is used in the project application for capturing the images of an approaching user and transform them into array of colors, thus allowing various operations to be applied (e.g. Background Subtraction - Section 5.4.2).

WebCamTexture allows the programmer to select the capturing device if more than one by using the variable textbfdevices - for example for using the front facing camera of the tablet this variable has to be set to *webcamTexture.deviceName = devices[1].name;* In addition size of the images can be chosen manually. High resolution means more heavy computations, so it should be consistent with the device it will be executed on. For the project the best performance is achieved by using 176x144, which doesn't overload the tablet and in the same time is sufficient enough for the subsequent computations. WebCamTexture inherits the base class **Texture** so it also can be applied to various objects, thus visualizing in real-time what exactly the device camera is capturing (Figure 5.8)

Figure 5.8: WebCamTexture is applied to a cube displaying what images the camera is capturing

5.3 Fish-eye Image Parameters

By using a fish-eye lens the application is able to capture wider area of the room in which the device is placed. But it requires additional computations because the received image is visually distorted. Because the fish-eye lens is not mounted to the device, but it is attached to it before running the application, these computations are needed to be done each time it is started.

5.3.1 Using an Image Mask

As already said the result image when using fish-eye lens is a circular image thus the corners of the regular rectangular image are left unused. Therefore it is good idea to create a mask that covers these areas and leaves only the active part of the image. Thus further computations in these areas are skipped, lightening the load on the device. In addition, because the lens is not every time tightly adjoined to the device, light marks can appear in the corners of the image adding false visual information. The mask helps in those cases too, by removing that type of noise.

The method of creating the mask is by simple intensity thresholding. All pixels that have values above 0.9 are considered part of the image that presents interest and marked white - the others are marked black. After the mask is created it is applied to the captured image leaving only the pixels in the circular area of interest. In the application the function for estimating the mask is activated by a button on the application screen (Figure 5.9). It can be activated at any time the user wants, but it should be used for example when the fish-eye lens have been displaced from its initial position.

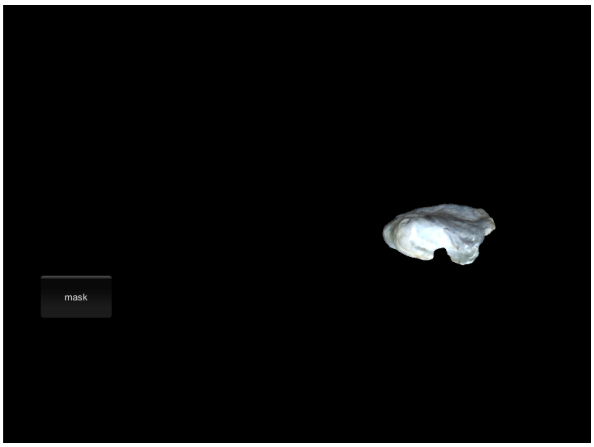


Figure 5.9: The button on the application screen which activates creation and applying of the mask

It should be noted that sometimes when creating the mask the image can contain black areas because of presence of dark objects in the scene (Figure 5.10 a). So it is more convenient when creating the mask the lens to be covered with white paper or handkerchief for better estimation (Figure 5.10 b). This is also the reason why such high value of the threshold is chosen - all the pixels of interests are close to white.

(a) **(b)**

Figure 5.10: Results of creation of the mask method: (a)Sometimes the image can contain black parts that are also falsely marked; (b)Better results when covering the lens with white cover - paper or handkerchief while capturing the mask

5.3.2 Estimation of Fish-eye Image Center

The received mask image is also used for calculating the parameters of the fish-eye image needed in the later computational methods.

One of them is the center of the circular image. Here it should be noted that when the lens is attached to the device camera, the later cannot capture the whole area of the lens, thus resulting in not fitting the whole circle in the result image (see Figure 5.10).

Thus on the first and last rows and columns of the mask there are presence of white pixels. Exactly these pixels are used for calculating the center. The ones on the top and bottom rows estimate the x-coordinate, and the ones on the leftmost and rightmost column - the y-coordinate. This is done by finding the mean of the coordinates of these pixels - see the pseudo code below:

Algorithm 1 Finding image center

```
1: sum.x_coord = 0
2: count = 0
3: for all pixels do
4:     if pixel.y_coord == bottom_row then
5:         if pixel.color == white then
6:             sum.x_coord = sum.x_coord + pixel.x_coord
7:             count = count + 1
8: circle.x_coord = sum.x_coord/count
```

In the example above all the x-coordinate of the white pixels on the bottom row are summed and then their mean is found. The same is done with all the pixels on the top row. This two values are meaned again in order a more precise value of the x-coordinate to be estimated. This process is repeated also for the y-coordinate - thus leading to the calculating of center of the circular image (Figure 5.11).

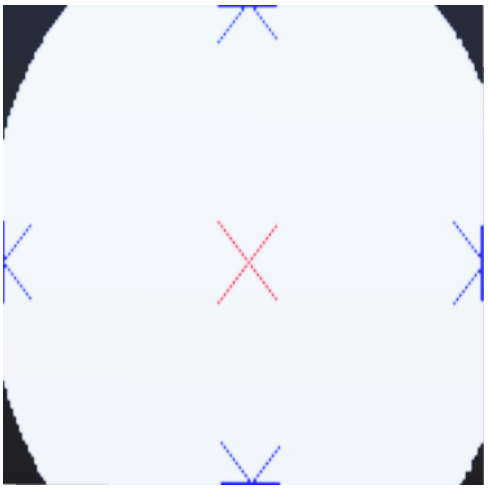


Figure 5.11: Visualization of the method of calculating the fish-eye image center (red cross). The x-coordinate is the mean value of the mean of all top-white pixels x-coordinates (top-blue mark), and mean of all bottom-white pixels x-coordinates (bottom-blue mark). Similarly the y-coordinate is the mean value of the mean of y-coordinates of all the white pixels on the leftmost column (left blue mark), and the mean of the y-coordinate of the all rightmost white pixels (right blue mark)

5.3.3 Radius Calculation

When the center of the circular image is already found it can be used for calculating also the radius. Observing the black and white mask it is visible that all the points that lie exactly between the black and the white areas belong to the edge of the circle. So by using the coordinates of some the points lying on it and calculating the euclidean distance between them and the already found center (see Equation 5.1 below), the radius can be easily estimated.

$$R = \sqrt{(c_x - p_x)^2 + (c_y - p_y)^2} \tag{5.1}$$

where

R	is the radius of the circualar image
c	is the center of the circle
p	is the selected point from the edge of the circle

For the current algorithm the 8 points chosen for the calculation are the pixels that belongs to the top and bottom rows, and left-most and right-most columns (Figure 5.12).

When a radius is calculated from using each of these points the final value is found by calculating the mean of the resulting 8 values.

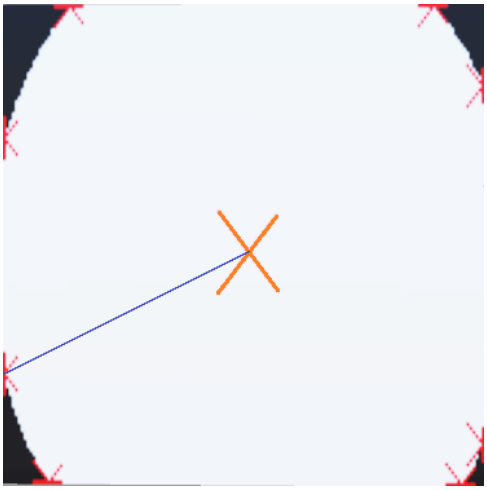


Figure 5.12: The red marks represents points lying on the edge of the circle on the top and bottom row, leftmost and rightmost column. The radius of the fish-eye image is the euclidean distance (blue line) calculated between each one of them and the center of the image (big orange cross)

5.4 Head Detection

A head detection method is needed in order to estimate the position of an approaching viewer to the device, and render a corresponding projection of the 3D scene according to this position. There were tried out a few different approaches for achieving this before the final one was chosen.

5.4.1 Initial Attempts

Enox Software provides an Asset Plugin for using OpenCV from within Unity [19]. It allows variety of functions for processing of images in real-time using the WebCamTexture (see Section 5.2). The capabilities of Unity.OpenCV for Unity runs on mobile devices as well as on desktop application and support both Android, Mac OS, iOS and Windows applications. The plugin is easy to use and many tutorial videos and examples can be found in Internet (Figure 5.13). Unfortunately the reason for not using it in the current project is that is not free Asset to use and requires payment in order to use its functionalities.

Alternative external software was tested in order to implement a head-tracking algorithm. The non-commercial software **HeadTrackingDemo_NC** developed by Seeing Machines [20], is based on the **faceAPI** and allows the user to apply head-detection on real-time video captured by a device camera . It provides data not only about 3D head position (distance from the center in meters) but also about the rotation of the head (Figure 5.14). This is so, because the tracking algorithm is based on finding facial features such as eyes, lips and nose. It also allows the users to select which camera exactly they wish to use.

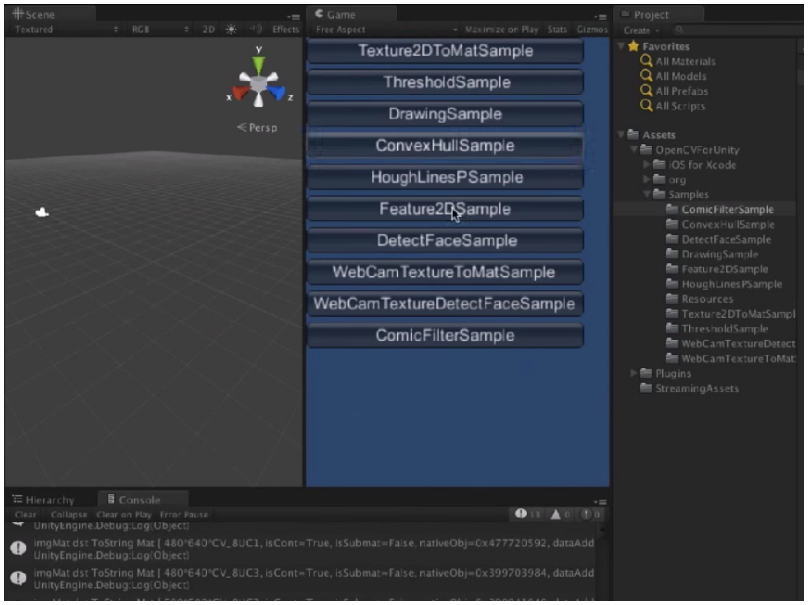


Figure 5.13: Video example showing the capabilities of OpenCV Asset for Unity *Source: Unity Asset Store [19]*

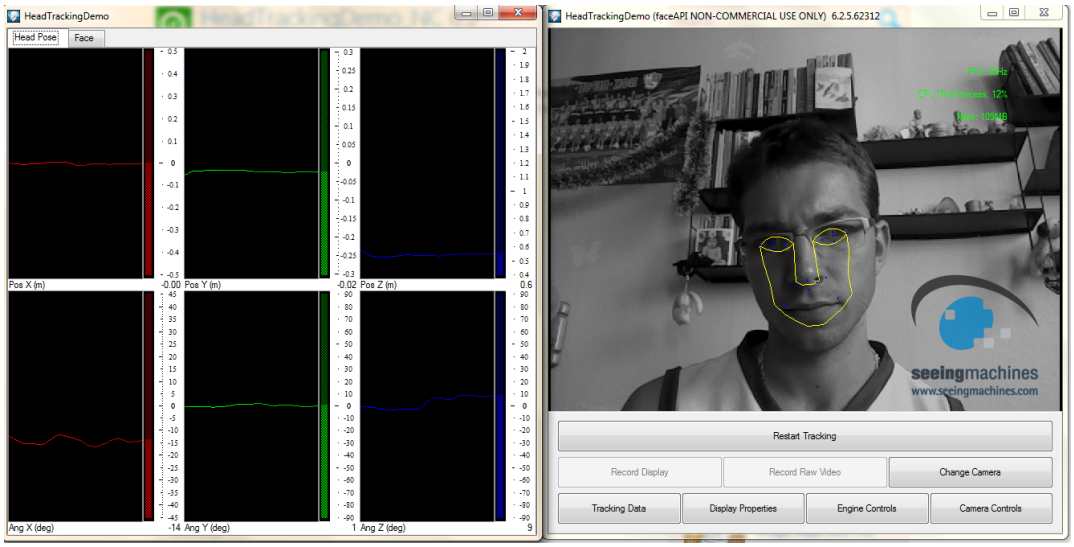


Figure 5.14: The interface of the HeadTrackingDemo_NC (right) and the visualized results from the head tracking procedure (left)

In order to use this information in Unity, however, a plug-in called **faceApiStreamer** is needed (Figure 5.15). It transfers the collected data, using specified port, from the HeadTrackingDemo application to Unity where a script collects it and it can be used for instance for moving a camera or applying transformation to a game character. Very good example of how all the aforementioned software is communicating and working together is shown and

explained by Andy Saia [21].

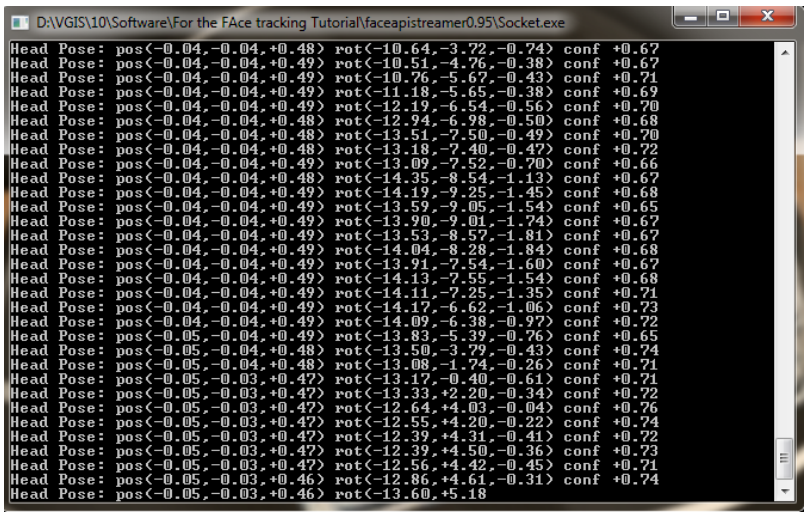


Figure 5.15: FaceApiStreamer is getting the head position information and transferring it via port to Unity

However the main reasons for not choosing this approach for the current project are first because the FaceApiStreamer doesn't work on a mobile devices and a desktop PC should be used as mediator in the communication, and second because the plug-in does not allow manual selection of the device camera. Thus, only the main camera of the tablet could be used which is not applicable for the current project.

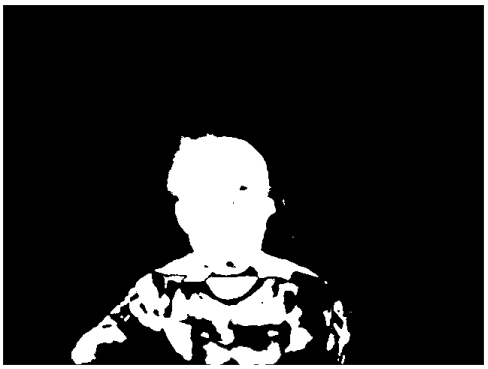
5.4.2 Background Subtraction

The method that is used in the project for estimating the users' head-position is by applying background subtraction. It captures an image when there are no persons visible in the room and it is used as reference image. Having this, a subtraction between this reference image (or background image) and every next frame captured by the camera is performed (Figure 5.16 a,b). The subtraction is done between the corresponding pixels of the both images, therefore if there are no difference in the scenery the result of the subtraction is 0. In other case, in the areas where the images are different (meaning that there are movement in the scene - e.g. a viewer have appeared in the room) the subtraction result values are different than 0, indicating in which part of the image is the difference. The resulting values in these areas are different so a threshold is applied in order to transform the resulting image into black and white (FIgure 5.16 c) for easier understanding and more important for easier further manipulation.

In their paper Cheung and Kamath [22] present the main challenges in developing a good background subtraction algorithm, reveal the main steps involved in the process and in the same time compare few different approaches for doing a background subtraction.

The most problematic issue that the algorithms have to deal with is the one with the changing illumination. When the video sequence is captured over long period of time, the

(a) (b)



(c)

Figure 5.16: The process of performing background subtraction (a) Background image used as reference; (b) Image from a video sequence in which a movement is present in the scene; (c) The resulting thresholded black and white image indicating the difference between the current frame and the background image

illumination of the scene is also changing and therefore a static background image fails. Another issue with the lighting is present if the illumination source is blocked or cover by an object causing changing of the intensity values of the whole image, thus causing false detection. Other issues can appear when shooting outdoor scenes. The methods should avoid detecting non-stationary background objects such as swinging leaves, meteorological conditions (e.g. rain, snow) and shadow cast by moving objects. Finally, quick changes in background such as starting and stopping of vehicles should also be considered.

Although there are many background subtraction techniques they follow the same methodology flow, described by the diagram in Figure 5.17. There are four major steps, although not all of them are mandatory in the process - preprocessing, background modeling, foreground detection, and data validation. **Preprocessing** stage is used for transforming the input image sequences into specific, easy to process by the following steps, format. **Background modeling** uses the new video frame to calculate and update a background model, which is used as a reference to future frames. During the **foreground detection** phase groups of pixels from current video frame that does not correspond to the background model are de-

tected and output as a binary candidate foreground mask. The final step, **data validation** examines the output mask in order to eliminate and filter false positive pixels that don't correspond to moving objects giving the final foreground result.

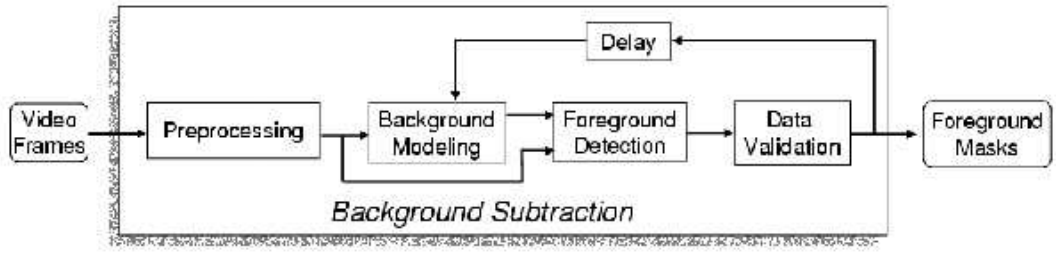


Figure 5.17: Flow diagram of a generic background subtraction algorithm. *Source: Cheung and Kamath [22]*

In many computer vision systems in general, smoothing of the initial input images, by using mean or median filters is used in early stages of the processing in order to remove or reduce camera noise. When used in outdoor background subtraction systems the smoothing is useful also for removing environmental noise such as rain and snow. However useful in general, for the current project it is not very effective first because the application is designed for indoor use, and in addition it increase computational load of the whole system, not improving significantly the result image.

Another part of the preprocessing phase is to transform the input images in suitable data format. Most of the background subtraction algorithms use the luminance intensity of the image pixels. However, using color space of images, either RGB (Red-Green-Blue) or HSV (Hue-Saturation- Value), is becoming more popular data format in the background subtraction. This is so because color is considered better than luminance at identifying objects in low-contrast areas and in suppressing shadow cast by moving objects [23, 24]. In the current project application the green channel showed slightly better visual results over the intensity value of the pixels, and it was chosen for the purpose of the application, although the red, blue and intensity also can be used as a substitution.

In general the HSV color space is considered robust against changes in light, which are very common in the outdoor conditions Moeslund [25]. It features a cylindrical geometry, as shown on Figure 5.18. The angle around the center is called the hue, the distance from the center is called saturation and the height is called the value, which corresponds to brightness. A hue value of 0 is the primary red color, going around 120° is the green primary color and the primary blue is at 240°, and then rotating back to red at 360°. Exactly the hue parameter in the HSV color space was also tested for the purpose of the project but it didn't resulted in acceptable image mask (Figure 5.19), leaving the usage of the green color channel as the best choice for the application.

The second step in the background subtraction methodology is skipped in the current project, because it requires much computational capacities, and the application is tired to be kept as simple as possible, in order not to be too heavy for running on mobile device. But in general much effort has been devoted into researching that step in order to develop background model that is robust against environmental changes in the scene, but still sensitive

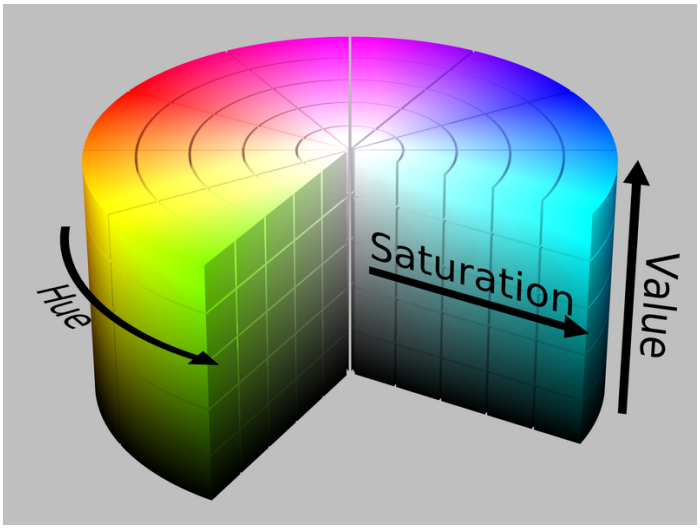


Figure 5.18: Representation of HSV color space; *Source: http://en.wikipedia.org/wiki/File:HSV_color_solid_cylinder_alpha_lowgamma.png*

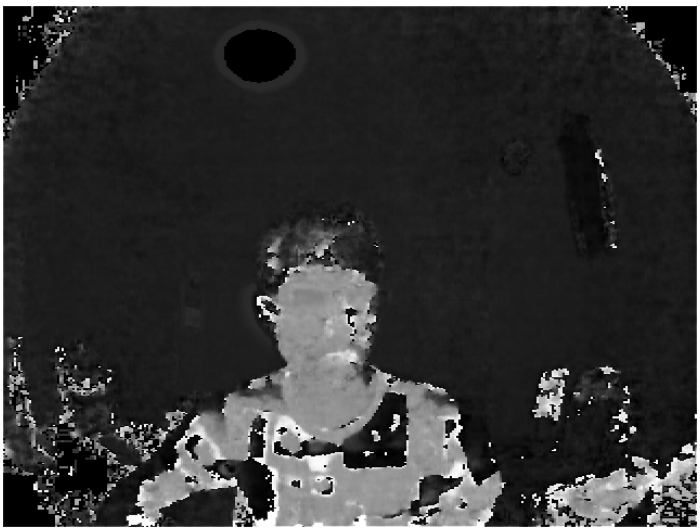


Figure 5.19: Result image showing the representation of an input frame using the Hue parameter of the pixels

enough to identify all moving objects of interest.

The background modeling techniques are classified into two main categories - non-recursive and recursive. The non-recursive techniques use a sliding-window method by storing a buffer of the previous N video frames, and calculates the background image based on the variations of the pixels from the images within the buffer. Contrary the recursive techniques do not maintain such buffer for background estimation. Instead, they recursively update single background model based on each input image, thus allowing input frames from distant

past to have an effect on the current background model.

The actual background subtraction is performed in the foreground detection phase. As already mentioned it compares the input video frame with the background model, and identifies candidate foreground pixels from the current input image. The most common approach for foreground detection that is also applied in the project is to use a threshold in order to check whether the input pixels value are significantly different from the corresponding pixels from the estimated background. The formula that this method is based on is shown on Equation 5.2

$$|I_t(x,y) - B_t(x,y)| > T \tag{5.2}$$

where

T	is the foreground threshold
$I_t(x,y)$	is the pixel value at spacial location (x,y) and time t
$B_t(x,y)$	is the background estimated pixel value at spacial location (x,y) and time t

The final step is the data validation. It is the process of improving the candidate foreground mask based on information obtained from outside the background model. In the current project this phase includes simple filtering of the noise pixels and it is further explained in the next Section 5.4.3.

5.4.3 Adjustments

Sometimes when performing the background subtraction small false detected groups of pixels can appear in the foreground binary image, due to slight moving or shaking of the camera, light change or other cause. In order to remove that kind of noise and to detect only large enough objects that correspond to a person appearing in the image a threshold on the count of the pixels is performed. All the detected pixels, colored in white are counted in the foreground mask, and if their number does not exceed the set threshold they are ignored. Otherwise it is considered that there is a person captured in the scene, so the algorithm continues with performing the head detection based on the detected group of pixels.

In order to find the position of the head of the viewer the center of the mass of all the white pixels is calculated. The method is very simple and easy to implement but unfortunately it does not give correct results. This is due to the fact that usually when a person is approaching his/hers whole body is detected by the background subtraction algorithm, and for the project only the position of the head is needed. Therefore adjustments of the calculated coordinates is required.

The idea implemented in the project is based on the image refraction that is done using the fish-eye lens, and that the camera is positioned horizontally according to the ground. In that case whatever direction the users are approaching the tablet from, their heads are always closer to the center of the image in comparison to their body (Figure 5.20). Based on this observation a weight function is created for each of the detected pixels, depending of their distance from the center of the image. It uses an exponential function (Equation 5.3),

represented by the graph on Figure 5.21.

$$W(p) = exp\left(\frac{-R^2}{\sigma^2}\right) \tag{5.3}$$

where

$W(p)$	is the weight of the pixel
p	is the current pixel
R	is the calculated radius of the fish-eye image
σ	is the distance from the pixel **x** to the center of the fish-eye image

(a) **(b)**

Figure 5.20: Viewers approaching from different directions to the fish-eye camera, captured by the background subtraction method

Thus pixels that are closer to the center of the image, and are assumed that are part of the head of the viewer, have much greater weigh in the calculation of the center of the mass compared to those that are situated near the edges of the image. A result from the calculated head detection, after applying the aforementioned adjustments is shown on Figure 5.22.

5.5 Calculation of 3D Position

Having the already estimated coordinates of the viewer head in the captured image, next step is to calculate the 3D world coordinates and translate the camera to them. To do this, first the 2D fish-eye image coordinates should be translated into 3D spherical coordinates. Having them it is easier to transform them to 3D Cartesian coordinates (Figure 5.23) and use them in Unity for moving the camera in the scene, corresponding to the movement of the viewer.

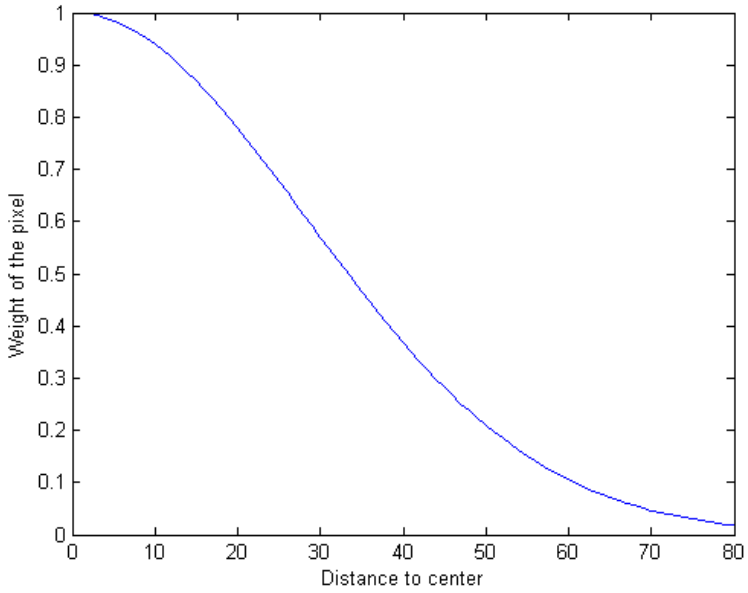


Figure 5.21: The graphical representation of the exponential function, using a sample radius value of 80 pixels. As smaller the distance from the given pixel to the center is - the higher weight it has in calculation of the head-position

(a) **(b)**

Figure 5.22: Examples of the final head-detection method. The red cross represents the calculated position of the head

5.5.1 From Fish-eye to 3D Spherical Coordinates

The way in which spherical coordinates describe a point position is using angles instead of signed distances (used in Cartesian coordinate system). Angle ϕ or the azimuthal angle in the XY-plane (horizontal plane) is defined in the range between 0° and 360°. The other angle θ is

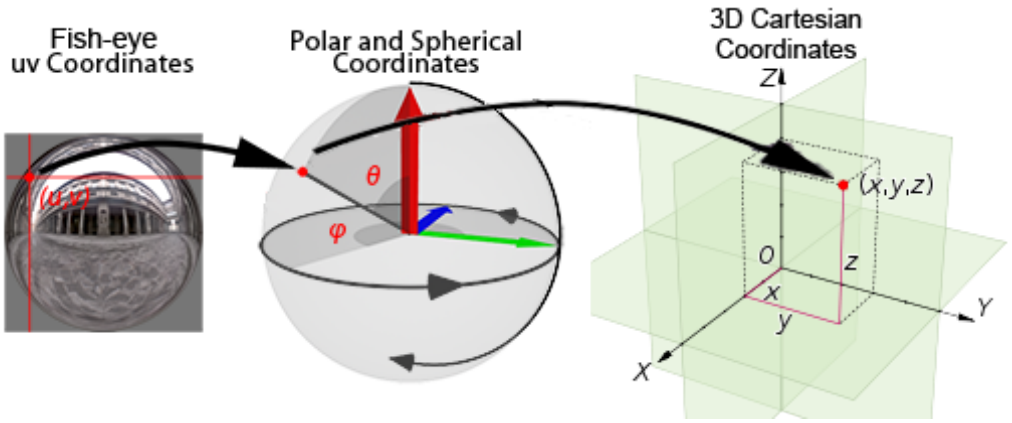


Figure 5.23: Illustration of the transition between fish-eye, spherical polar and Cartesian coordinates

the polar angle (also known as the zenith angle) and ranges between 0° and 90° (Figure 5.24).

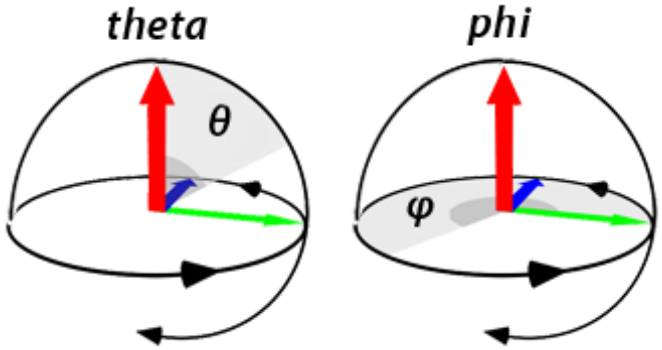


Figure 5.24: The two angles ϕ and θ describing a point in spherical polar coordinates *Source: Scratchapixel [26]*

The ϕ angle is computed simply by applying the arctangent function in the triangle defined between the center of the image, the point defining the detected head position and the x-axis of the image (Figure 5.25). For the arctangent equation the length of the opposite side of the triangle is represented by the y-coordinate of the detected point, and the adjacent side is its x-coordinate. It should be noted that these coordinates are considered as part of a 2D coordinate system with origin the center of the fish-eye image.

The zenith angle θ is calculated by finding the distance between the detected head-position point and the center of the image in pixel units. Afterwards this distance is multiplied by a coefficient **k** describing the angle value for each pixel of the image (see Equation 5.4).

$$\theta = k\sqrt{(c_x - p_x)^2 + (c_y - p_y)^2} \tag{5.4}$$

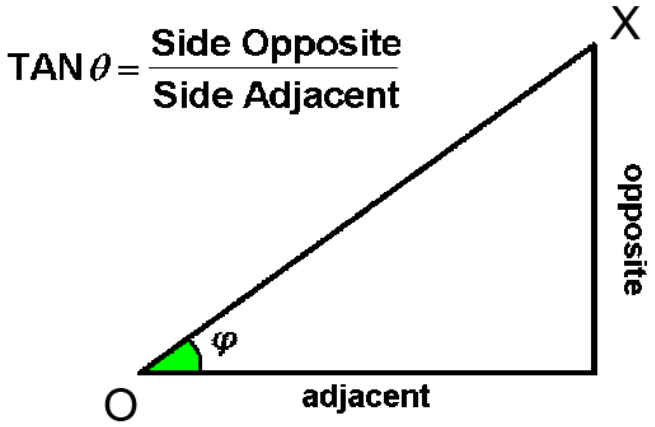


Figure 5.25: The tangent of the angle ϕ is calculate using the right triangle described by the center of the fish-eye image **O** and the point representing the detected head position **X**

where

k	is the coefficient describing the angle value for each pixel of the image
c	is the center of the circle
p	is the selected point from the edge of the circle

Initially **k** was simply calculated by dividing 90° by the length of the radius. Thus pixels that are lying in the center of the image have values of 0° for the θ and these in the edge of the image have values close to 90°.

However this assumption is not correct. This is due to the refraction of the image by the fish-eye lens (Figure 5.26). Observing the image it is clearly visible that the areas around the edge of the circle contain more spacial information in fewer numbers of pixels than the center area. Thus the same real-world distances don't correspond to equal distances in pixel units in the image. Therefore a new approach is required for calculating the θ angle.

After proving that linear correspondence between the pixel distance and the angle value is not correct the idea of using polynomial function for describing this correspondence is implemented. This function is found by calculating number of angle values for specified points in the image, and then fitting a curve to the collected data set.

In order to calculate the needed angle values a measuring tool with marked areas on each 10 cm distance is captured with the fish-eye lens (Figure 5.27). The measuring tool is positioned on predefined height above the camera - 48 cm - corresponding as close as possible to the value of the height of the camera in the 3D scene (Section 2.1). Therefore for each marking there are pairs of distances that can be used for calculating the tangent of the angle corresponding to these points - and consequently the θ itself (Equation 5.5).

$$\theta_p = \arctan\left(\frac{d}{h}\right) \tag{5.5}$$

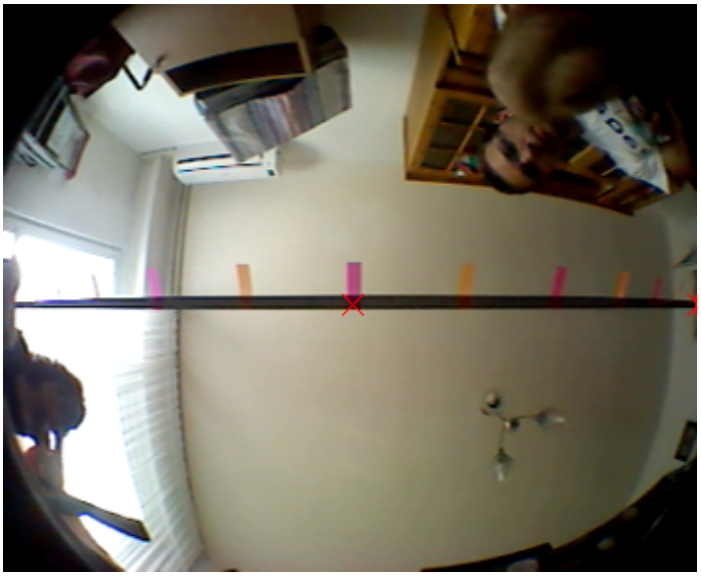


Figure 5.26: Image showing that the same real-world distances are not equally scatter along the fish-eye image due to the refraction by the lens. A measuring tool is captured with marked points on each 10cm length. In the center of the image the distance between two markings contains more pixels than these in the edge of the image.

Figure 5.27: The set-up used for calculating the θ angle. Measuring tool with marked distances on every 10 cm is captured by the camera with fish-eye lens

where

p is the current point in which the θ value is calculated

h is the height on which the measuring tool is positioned above the camera (48 cm)

d is the distance between the mark above the center of the image and the marking in point $\mathbf{p}$

After calculating the angle values in each of the marked points a set of paired data is collected. The first parameter is the distance of the point to the center and the other is the calculated angle value. Afterward this data is inserted into the Curve Fitting Toolbox of Matlab in search for a polynomial function describing the data set. The estimated equation (Equation 5.6) contains of 4 parameters and is visualized on the Figure 5.28. This function receives as an input the distance between the head position and the center of the image and returns the searched angle θ.

$$\theta = p1 * x^3 + p2 * x^2 + p3 * x + p4 \tag{5.6}$$

where

x	is the distance between the head-position point and the image center
$p1$	= 0.00006241
$p2$	= -0.005225
$p3$	= 0.6658
$p4$	= -0.2607

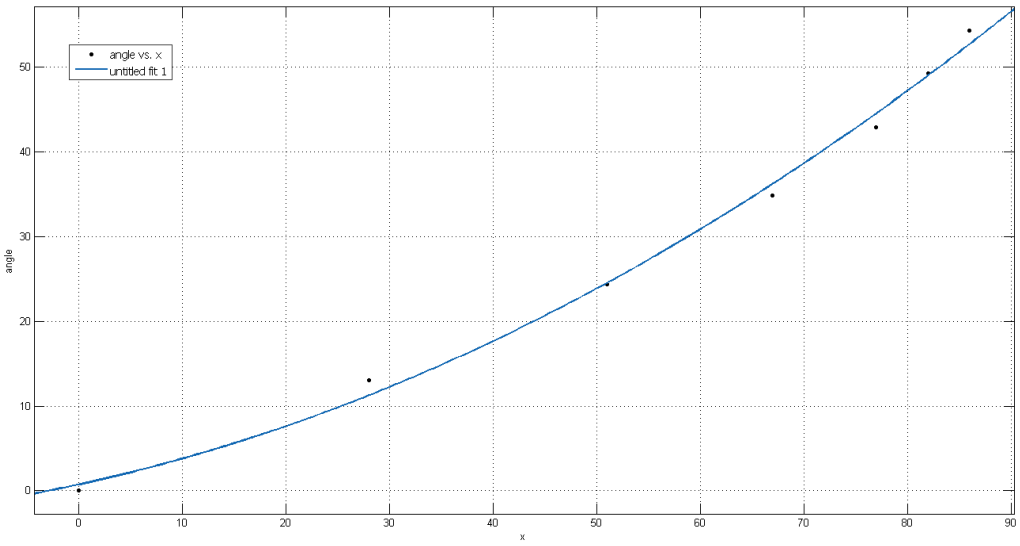


Figure 5.28: Visualization of the curve corresponding to the estimated polynomial function and the input data points

5.5.2 Estimating 3D Camera Position

Using the functions described in the previous section, for each input point from the fish-eye image that denotes position of the viewer head the corresponding spherical coordinates are calculated. Next step is to transform them into 3D Cartesian coordinates so the camera can

be translated to point in the 3D scene representing the position of the viewer.

Initially this was done by using the formulas suggested by Scratchapixel [26] for computing world coordinates from polar/spherical coordinates (Equation 5.7):

$$\begin{aligned} X &= \sin(\theta)\cos(\phi) \\ Y &= \sin(\theta)\sin(\phi) \\ Z &= \cos(\theta) \end{aligned} \tag{5.7}$$

However this leads to creating of a hemisphere with radius 1 unit (Figure 5.29), which limits the user position in the three directions (X, Y and Z) exactly to 1 unit. Therefore this formulation is not sufficient enough for the purpose of the project because sometimes the viewer can be positioned outside the space of the defined hemisphere.

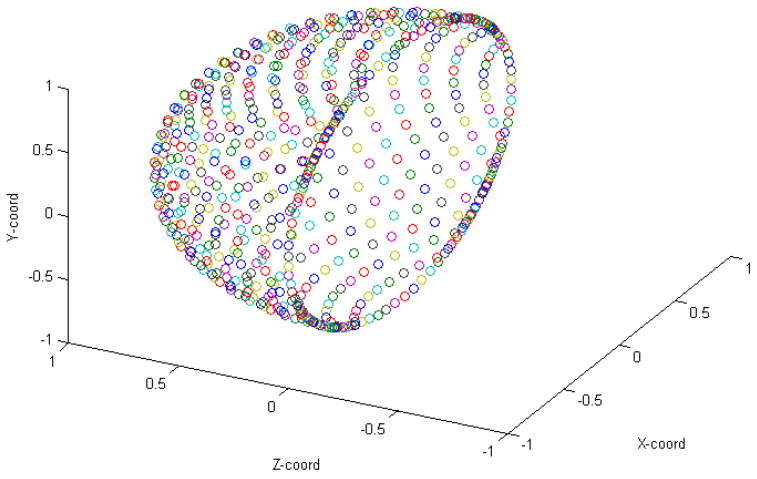


Figure 5.29: Hemispherical space created by set of input head-position points using Equation 5.7

An alternative method is to calculate the radius of this sphere by using predefined height (Z-coordinate) of 0.5 units. Then this radius value is used in the calculation of the other coordinates according the Equation 5.8. In that way, the X and Y coordinates can extend outside of the limited hemisphere space, resulting in resemblance of an Gaussian shaped 3D Space (Figure 5.30).

$$\begin{aligned} R &= \tan(\theta)0.5 \\ X &= \cos(\phi)R \\ Y &= \sin(\phi)R \\ Z &= \cos(\theta) \end{aligned} \tag{5.8}$$

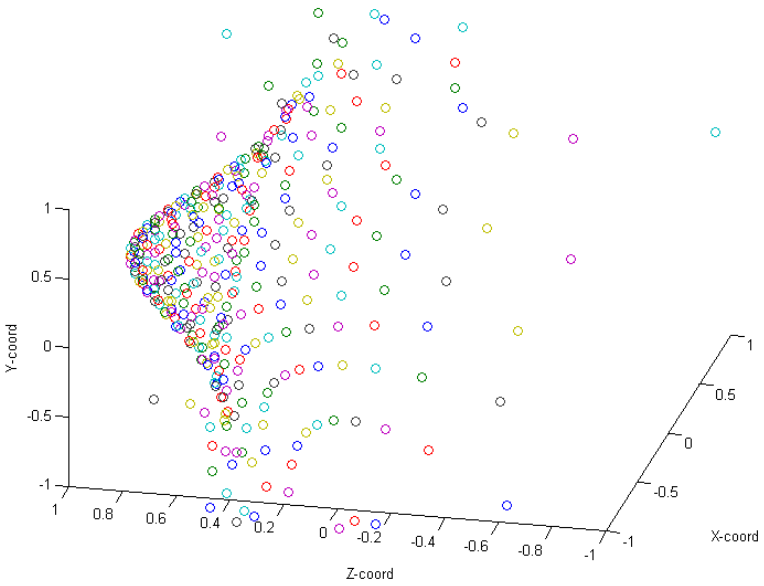


Figure 5.30: 3D space created by set of input head-position points using Equation 5.8

It should be noted that because the project is using only one camera, and the 3D coordinates of the viewer position are calculated from single image, the Z-coordinate (or the height at which the viewer is above the camera) cannot be correctly calculated. Therefore for the purpose of the project an assumption is made that all the users are observing the tablet from constant height of 50 cm. Thus the input head-position points don't correspond to positions in 3D space, but rather lie on a plane hovering 50 cm above the mobile device (Figure 5.31).

5.5.3 Smoothing of the Position Coordinates

Every single frame results in different head-positions detected by the background subtraction method. These positions are passed to the method for calculating of 3D camera coordinates, therefore they are also different for each frame. Because of the fact that the head position is calculated using the center of mass, sometimes the transition between the frames leads to big difference in the estimated 3D positions causing shaking or very fast moving of the camera, which is not visually appealing for the users.

In order to overcome this problem a method for smoothing of the 3D position transition between frames is implemented. Three different approaches for smoothing was tested. The first (and chosen for final) one is calculating the mean position between sequence of 5 frames.

Another method tested is also using 5 consecutive frames. The positions calculated in these frames are gathered in groups of 3 and median 3D position is calculated for each single group. The final smoothed position is the mean of these median positions. These method is assumed to be more accurate, but visually it does not provide better results, that's why the simpler one is chosen.

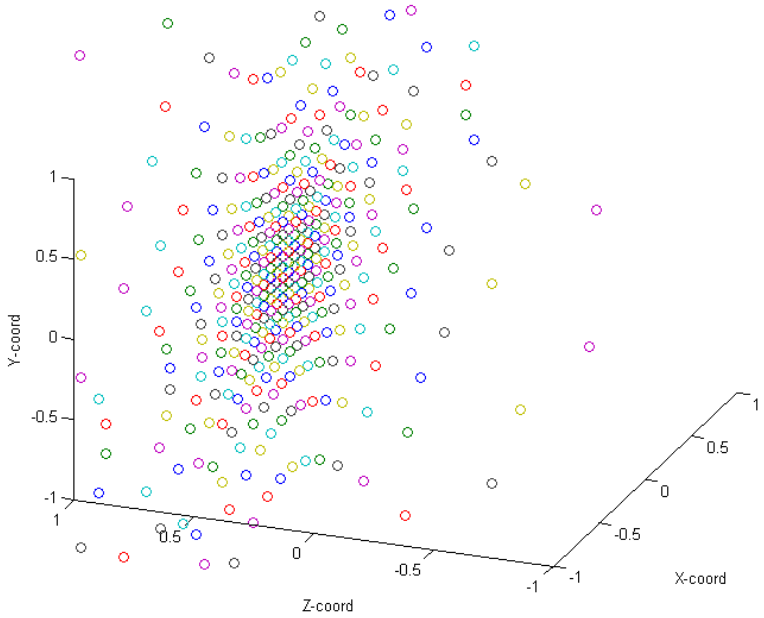


Figure 5.31: 3D plane space calculated by the same input points and using the equation like Figure 5.30 but with constant Z-coordinate - 0.5

Alternatively the approach using only the mean value is tested by using 10 and 20 instead of 5 frames, but that results in higher latency time. This is especially noticeable using the mean between 20 video frames showing very high response time between the head-position detection and moving the camera leading to very inconvenient and even confusing behavior.

5.6 Rendering

The rendering of the objects is done using the technique of off-axis perspective projection. A script based on the methods described in Section 3.3 is created and applied to the scene camera. It needs a projection plane representing the screen of the device, so one is created having the measurements of the display of the tablet. It should have its Mesh Renderer setting deactivated in order to be rendered invisible and the users to see the objects true it. Another thing that should be noted, is that the Unity uses left-handed coordinate system while the methods explained by Kooima [16] are defined for right-handed system, therefore when defining the normal of the projection plane it should be signed with minus.

Chapter 6

Results

In this chapter the mobile device hardware characteristics will be briefly discussed in Section 6.1 followed by performance response parameters (Section 6.2). Finally in Section 6.3 images showing the resulting objects are displayed along with real photographs of the same objects.

6.1 Hardware Parameters

The mobile device used for the purpose of the project is Toshiba Excite 7.7 (Section 4.1). The operation system of the tablet is Android 4.0.3 Ice Cream Sandwich and its CPU is Quad-core 1.5 GHz. The chipset of the device is Nvidia Tegra 3 and the GPU is ULP GeForce 2. The RAM that is available for use is with capacity of 1 GB.

6.2 Latency and Frames per Second

In order to calculate the latency of the main project application, an auxiliary one was created having similar functionality. It contains of the same methods and calculations but instead of showing some of the recreated objects a plain cube is displayed. However to this cube different color texture is applied depending of the position in which the camera is moving. So in order to estimate the latency time of the application a pendulum is swing in front of the tablet. On each of its swings it is changing its direction thus making the camera in the 3D scene to move accordingly. This whole action is captured by another camera and the time period between the actual swing of the pendulum and the response of the application is noted thus estimating the latency time (Figure 6.1). As already mentioned the cube in the scene changes its color depending of the movement direction - red for left and green for right. This color codes are used for easier detection of the response.

After calculating the latency and the frames per seconds of the application they have been estimated to have average values of **217 ms** and **57 fps**. In addition the auxiliary application was also used in order to establish the delay time between moving of the viewer and the corresponding movement of the camera (to the correct position) using the four different configurations for the smoothing algorithm of the 3D camera positions (Section 5.5.3). The results are shown in Table 6.1, proving that the 5-frames smoothing technique provide best results.

(a) (b)

Figure 6.1: The auxiliary application used for estimating the latency time. When the pendulum is moving to left (a) the cube in displayed in red color and when it is moving to the right (b) - it is colored in green

Smoothing method	Average delay measured
None	226 ms
Mean of 5 Frames	288 ms
Median of Mean of 5 frames	292 ms
Mean of 10 frames	495 ms
Mean of 20 frames	1056 mm

Table 6.1: Delay time measured when using different configuration of the smoothing algorithm (Section 5.5.3)

6.3 Result Images

During the project implementation several objects were reconstructed using 123D Catch, and the best looking of them was chosen to be shown in the application. They have different spacial dimensions in order to check if the application can handle object with different sizes and how they will be displayed. In the table below (Table 6.2) are shown the used objects along with their measured dimensions and further down resulting images of their display in the application. More photographs can be seen in Appendix B.

Object	Length	Width	Height
Rock	4 cm	1,8 cm	3,5 cm
Cube	3 cm	3 cm	3 cm
Candy	7,5 cm	2 cm	1 cm
Pen	16 cm	1 cm	1 cm

Table 6.2: Measured dimensions of the objects used in the projects

Figure 6.2: Image of the application displaying the model of a rock object, along with the real object next to the box

Figure 6.3: Image of the application displaying the model of a Rubik's cube object, along with the real object next to the box

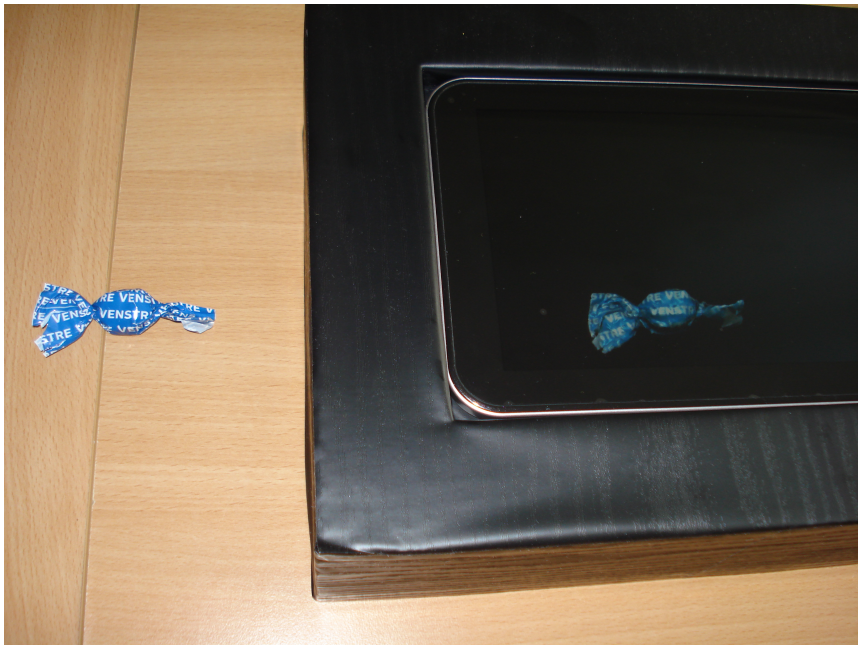


Figure 6.4: Image of the application displaying the model of a candy object, along with the real object next to the box

Figure 6.5: Image of the application displaying the model of a rock object, along with the real object next to the box

Chapter 7

Conclusion and Analysis

The aim of this project is to implement a mobile application that renders realistically looking models of small objects depending of the position of the viewer that observes it. As a conclusion it can be said that the methodology of the project provides not so complicated approaches for detecting the users position, that not overload the device memory, but in the same time the application provides sufficient realistically looking objects and projects them depending of the user position in space.

7.1 Problems and Limitations

The main problematic issues that the application experiences are connected with the illumination and the performance of the mobile device. As already mentioned in Section 5.4.2 the changes of the lighting conditions and obstruction of the direct illumination can lead to errors in the method of background subtraction, thus leading to incorrect estimation of the position of the viewers.

More sophisticated and precise background subtraction method can be implemented in search for better results but that leads to the other aforementioned problem - the performance of the mobile device. When programing software for not so powerful devices it should be always considered the ratio between performance and functionality. The applications should work without long processing time and high workload but in the same time they should fulfill the tasks they are assigned for.

Another limitation is set due to the use of only one standard camera for the project. Thus the application cannot take advantage of the functionality of stereo imaging or active stereo techniques like structured light or Time-of-Flight cameras. They can provide much better estimation of the object positions in 3D space and thus no need of assumption will be required (see Section 5.5.2). In addition assumption is made, that only one user is in front the tablet and looking at the screen. So if more viewers appear in the scene the head estimation method will fail and incorrect projection will be present for the user.

Chapter 8

Future Work

The current project just scratches the surface of the possibilities the application is capable of. There are many potential directions that the later can be extended or improved.

8.1 Interface Implementations

One idea that have broad base for work is implementing **Options** screen where the users will be able to set different parameters of the application. This will enhance the user experience and will improve the look of the interface, and in the same time will give the application much flexibility and new functionalities.

One of the parameters that can be set is the time delay between the assigning of the application to capture the background and the actual performing this action. Now this time offset is preset to 3 seconds and the user cannot changed. But sometimes it could not be enough for the user to disappear of the scene, for example if the room is too big. With this implementation this problem will be overcome and there won't be such difficulties for executing the background subtraction.

Following this line of thoughts, another parameter connected with capturing the background image can be implemented. In the option menu can be set parameter which defines a period of time in which a new background image is captured. In that way issues with changes in the illumination during long period of time can be overcome. But it should be thought about a method that do this only if the room is empty, otherwise a viewer can appear in the image that is used as reference for the background subtraction method leading to incorrect results.

Although there are a few objects reconstructed and used in the project, currently only one (or single set of objects) can be shown in the application and they are predefined - meaning that the users cannot choose what exactly they want to observe. A functionality that can be also implemented in the Options menu is to provide the users the opportunity to select what type of objects to be rendered by themselves, thus giving more diversity to the application.

8.2 Background Subtraction Improvements

The project can be developed in direction for improving the background subtraction technique. Now the method is simple and provide adequate results, but it cannot handle change in lightning conditions. More effort can be put into the background modeling part (see Section 5.4.2) of the subtraction method in order to make it more robust against changes in the illumination. There are many non-recursive techniques described by Cheung and Kamath [22] that use a frame buffer to store sequence of images and consider them for when execute background modeling. These techniques are considered highly adaptive as they do not depend on the history beyond those frames stored in the buffer. However sometimes the storage requirement of the buffer can be significantly large, so again the performance of the mobile devise should be considered. In the list below are given examples of some of the highly-adaptive techniques:

- Frame differencing
- Median filter
- Linear predictive filter
- Non-parametric model

As an alternative recursive methods can also be used. Compared with non-recursive techniques, they require less storage, but any error in the background model can apply its effect for a much longer period of time. Such techniques are:

- Approximated median filter
- Kalman filter
- Mixture of Gaussians (MoG)

8.3 Using External Software for Object Creation

Currently, by using 123D Catch software realistic looking objects can be created and used in the application. In order to extend the idea of the project other 3D modeling softwares like 3D Studio Max, Zbrush and Maya can be used for creating futuristic or unreal objects that cannot be captured in real life and used in the 123D Catch. In addition the restriction set by the photo-recreation software (see Section 4.2.1 will not apply and transparent, reflective or glossy objects can be used too.

Bibliography

[1] "Regional historical museum stara zagora." *http://www.tour.starazagora.bg/en/sightseeing/in-the-city/museums/208-regional-historical-museum.html* Last visited: 2.08.2015.

[2] G. Lepouras and C. Vassilakis, "Virtual museums for all: Employing game technology for edutainment," *Virtual Reality Vol.8*, 2004.

[3] A. Sideris and M. Roussou, "Making a new world out of an old one: in search of a common language for archaeological immersive vr representation," in *Creative Digital Culture*, 2002.

[4] S. Rizvic, "Story guided virtual cultural heritage applications," *Story Guided Virtual Cultural Heritage Applications*, 2014.

[5] "Mummy: The inside story." *https://www.britishmuseum.org/explore/online_tours/egypt/mummy_the_inside_story/mummy_the_inside_story.aspx* Last visited: 13.07.2015.

[6] "Leonardo: the ideal city." *http://www.da-vinci-inventions.com/ideal-city.aspx* Last visited: 13.07.2015.

[7] "The virtual museum of arts el pais." *http://muva.elpais.com.uy/flash/muva.htm?&lang=en* Last visited: 13.07.2015.

[8] "3d encounters: Where science meets heritage exhibition." *http://www.ucl.ac.uk/museums-static/science-of-3d/virtual-exhibitions/* Last visited: 13.07.2015.

[9] S. Y. Quan Wang, Jonathan Mooser and U. Neumann, "Augmented exhibitions using natural features," 2004.

[10] J. d. M.-B. Alvaro Gomez-Gutierrez, Jose Juan de Sanjose-Blasco and F. Berenguer-Sempere, "Comparing two photo-reconstruction methods to produce high density point clouds and dems in the corral del veleta rock glacier (sierra nevada, spain)," *Remote Sens*, 2014.

[11] J. P. Yasutaka Furukawa, "Accurate, dense, and robust multi-view stereopsis," *Pattern Analysis and Machine Intelligence Vo.32*, 2009.

[12] R. Fergus, "Lecture about multi-view stereo & structure from motion."

[13] G. P. Klaus Haming, "The structure-from-motion reconstruction pipeline," *Kybernetika, Vol. 46*, 2010),.

[14] R. S. Noah Snavely, Steven M. Seitz, "Photo tourism: Exploring photo collections in 3d," *ACM Transactions on Graphics (TOG)*, 2006.

[15] T. A. D. Carolina Cruz-Neira, Daniel J. Sandin, "Surround-screen projection-based virtual reality: The design and implementation of the cave," *SIGGRAPH '93*, 1993.

[16] R. Kooima, "Generalized perspective projection," August 2008.

[17] Wikipedia, "Oled." *https://en.wikipedia.org/wiki/OLED* Last visited: 20.07.2015.

[18] "123d catch video tutorials." *http://www.123dapp.com/howto/catch* Last visited: 22.07.2015.

[19] "Opencv for unity." *https://www.assetstore.unity3d.com/en/content/21088* Last visited: 24.07.2015.

[20] "Head tracking demo nc." *http://headtrackingdemo-nc.software.informer.com/* Last visited: 26.07.2015.

[21] A. Saia, "Unity 3d plus faceapi." *http://www.andysaia.com/radicalpropositions/unity-3d-faceapi-love/* Last visited: 26.07.2015.

[22] S.-C. S. Cheung and C. Kamath, "Robust techniques for background subtraction in urban traffic video," *Proceedings of the SPIE*, 2007.

[23] R. B. P. KaewTraKulPong, "An improved adaptive background mixture model for real-time tracking with shadow detection," *Video-Based Surveillance Systems vol.2*, 2002.

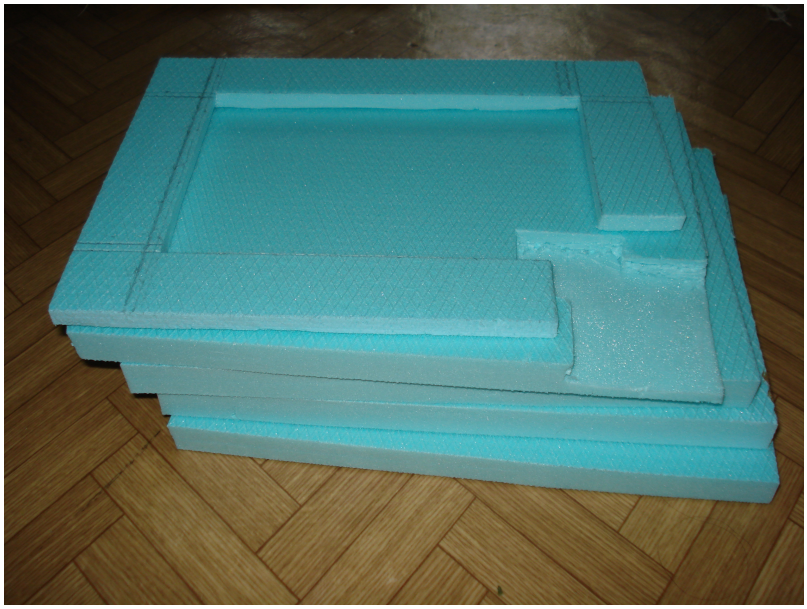
[24] R. Cutler and L. Davis, "View-based detection and analysis of periodic motion," *Pattern Recognition vol.1*, 1998.

[25] T. B. Moeslund, *Introduction to Video and Image Processing: Building Real Systems and Applications (Undergraduate Topics in Computer Science)*. Springer, 2012.

[26] Scratchapixel, "Converting latitute-longitude maps and mirror balls." *http://www.scratchapixel.com/old/lessons/3d-advanced-lessons/reflection-mapping/converting-latitute-longitude-maps-and-mirror-balls/* Last visited: 24.07.2015.

Appendix A

Creation of the Tablet Holder Box



Appendix B

More result images