TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SANTIAGO CORTES REINA

DEPTH ASSISTED COMPOSITION OF SYNTHETIC AND REAL 3D SCENES

Master of Science thesis

Examiner: prof. Atanas Gotchev
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on Apr 8, 2015

ABSTRACT

SANTIAGO CORTES REINA: DEPTH ASSISTED COMPOSITION OF SYNTHETIC AND REAL 3D SCENES
Tampere University of technology
Master of Science Thesis, 66 pages
October 2015
Master's Degree Programme in Information Technology
Major: Signal processing
Examiner: Atanas Gotchev

Keywords: Mixed reality, capture and display, motion capture, previsualization.

In media production, previsualization is an important step. It allows the director and the production crew to see an estimate of the final product during the filmmaking process. This work focuses on a previsualization system for composite shots which involve real and virtual content. It shows the camera operator a correct perspective view of how the real objects in front of him look placed in a virtual space. The aim is to simplify the workflow, reduce production time and allow more direct control of the end result.

The real scene is shot with a 3D scene capture device, which combines an RGB color camera with time-of-flight depth camera. The device's pose is tracked using a motion capture system. Depth-based segmentation is applied to remove the background and content outside the desired volume, the captured geometry is aligned with a stream from the RGB color camera and a dynamic point cloud of the remaining real scene contents is created. The virtual objects are then also transformed into the coordinate space of the tracked camera, and the resulting composite view is rendered accordingly. The prototype camera system is implemented as a self-contained unit with local processing.

A prototype system was constructed from a Microsoft Kinect v2, providing depth and color information of the real scene and a Microsoft Surface Pro 3 as a processing and display device. Both instruments were attached to a camera shoulder mount, with optical markers fixed to the body of the camera. The pose of the camera in 3D space is tracked with a Natural Point OptiTrack motion capture system, which streams the location information to the Surface device over a wireless 802.11n channel.

At its current state, the system is running at 15 frames per second with a resolution of 1024x768. Subjectively, the frame rate is already smooth enough for the operator to feel as if using a regular camera. Further improvements are targeted in the processing speed and the image quality provided by the system. The image suffers from some depth capture related artefacts which influence the depth segmentation, and therefore adaptive filtering methods based on edge-aware bilateral filtering have been investigated. The tested filtering has improved the quality significantly, while more effort has to be put in implementing the filtering in an efficient way.

PREFACE

The work done for this thesis was performed as part of the 3D media group in Tampere University of Technology. The work is part of the Centre for Immersive Visual Technology (CIVIT). The aim of the work is to research the potential of new technologies in industry, in particular in media production.

I am grateful for everyone in the group. In particular I wish to thank Atanas Gotchev and Olli Suominen for their help and support. I also want to mention Atanas Boev for aiding me in the early stages of the process.

I wish to mention Professor Alfredo Restrepo from the Universidad de los Andes for steering me in the signal processing direction at the early stages of my academic life.

I would not be in Finland were not for the support of my father and my brother. I am very grateful to Kristen Kurtzeborn for being with me during this whole process. Finally I wish to mention Hidetaka Miyazaki for providing a balance during the previous months.

Tampere, 21.10.2015

Santiago Cortes

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AR	Augmented reality
CCS	Camera coordinate space
CIVIT	Centre for Immersive Visual Technologies
HMD	Head mounted display
MLE	Maximum likelihood estimator
MR	Mixed reality
LS	Least squares
ToF	Time of flight
RCS	Real coordinate space
VCS	Virtual coordinate space

1. INTRODUCTION

The system proposed in this work is a blend of two relatively recent concepts and the technology that has been developed from them. These concepts are mixed reality (MR) and the virtual camera.

Mixed reality systems are a relatively recent advancement in technology. The term MR encompasses any and all systems that merge virtual and real worlds to produce environments and visualizations where physical and virtual objects coexist and interact. The first general term for such a system, augmented reality (AR), was coined in 1990, AR defines system where computer-generated sensory information (sound, video, etc.) is used to enhance, augment or supplement a real world environment.

The full description and formalization of the MR systems was published in 1994 by Paul Milgram [36]. It was not until 1999 and the release of the ARToolkit [37] by the Nara Institute of Technology that the bulk of the computing community had access to real time blend of real and virtual data. Most of the AR applications running on web browsers produced during the early 2000s were developed based on the ARToolkit. MR has evolved with the technology used to produce it, and the uses for it have multiplied. Nowadays, in the embedded system era, a single device can have most, if not all, the sensors and computing power to run a good MR application. This has led to a plethora of uses, from training and military simulation to simple videogames.

Any system that mixes data from a virtual and real environment to produce a joint world can be classified in the spectrum proposed by Milgram [36]. Most systems can be divided into a capture component and display components. The capture component is usually some kind of image capture device, a single camera [38], a stereo pair [39], or one or more depth sensors [40]. The display component show the user the resulting environment, these can be screens, projectors, head mounted displays (HMD), etc.

The success of the Oculus Rift has revitalized the development of mixed reality systems using HMD. Relevant examples are the MR systems developed by University of California aimed at tele meeting [41]. The aim of these systems is to create a full 3D model of a person and place them in a shared environment with other people. Microsoft has announced the HoloLens [42]. This system is a view-through mixed reality display that places virtual objects on top of the real world using a semitransparent screen. Many of the recent MR developments use head mounted displays; these are a natural interface for systems that try to achieve presence, (the feeling of ownership towards a body), but this is not always the best option. The system developed uses a more familiar interface for a camera operator. The display and the capture systems are attached to a commercial shoulder mount.

The virtual camera concept was proposed in the context of computer graphics and presented at the ACM Siggraph conference [43]. It was described as a “cyclops” device which renders a virtual scene according to a set of sensors attached to a camera tripod. The concept has evolved through the years and has been applied to videogames [44] and movies [45]. This evolution has been linked to the advancement in technology: better rendering capabilities, better sensing techniques, more powerful computers, etc. The most common use of the device is to visualize camera motions through a virtual environment in media productions. For example, the movie Avatar used a virtual camera to visualize and plan virtual camera movements [46].

The virtual camera captures the position and orientation of the camera inside a space and returns a render of the image from a point of view relative to it. Many systems have been used to estimate the pose of the camera. For example, accelerometers and gyroscopes in smartphones or motion capture in movie studios.

In the University of Kiel, a mixed reality camera [35] was developed in 2008. This camera is also based on Time of Flight technology. The system scans the scene and creates a model of the background prior to the operation. Once in operation, the camera is fixed and the mixed space is created.

Previsualization (previs) is the name given by the movie industry to any system that allows the director and the staff to view an approximation of the end result before actually shooting the scene. It helps save time by minimizing the errors and the iterations necessary to materialize the view of the director.

In order to improve on current previs virtual cameras, the system developed for this work takes into account the position of the objects in the space. The real objects in the space are then mixed with the virtual objects, and a complete 3D joint space is produced. Real objects can occlude virtual objects and vice versa. The end result is a mixed reality rendered in place for the virtual camera system. This allows a movie director to explore how the real characters will blend in with a virtual background. This can help reduce errors and corrections done in postproduction. Thus reducing the cost and making the production cycle faster.

The thesis is structured as follows: First, the theoretical background is presented. This contains both the mathematical description of the spaces and the algorithms that use this information to provide the end result. Second, the system is described. An algorithm block diagram and a computational block diagram are drawn and explained. Third, the results and performance metrics are presented. Finally the conclusions and future work are discussed.

2. BACKGROUND

This chapter covers the theoretical foundation of the work documented in this thesis. Some sections use commercial devices and proprietary software whose inner workings are not public. To provide theoretical context in these cases, example implementations are shown and explained.

2.1 Geometric descriptions

In order to describe the system, a vocabulary has to be established to describe the geometry of the scene and the image capture process. The main tools to accomplish this are Euclidean and projective geometry. These are very rich fields, and the interested reader can find a more complete description in [1]. In this section, the necessary geometric primitives to describe the system as well as the terms used in the rest of the text are presented.

2.1.1 Points

A point in 2D space can be represented by a vector in the form

$$\vec{p} = (u, v)^T \in \mathbb{R}^2, \tag{1}$$

where u and v are the coordinates of the point and $\mathbb{R}^2$is the 2D Euclidean space. Another representation is the so called homogeneous coordinates where the point is represented in the form

$$\vec{p} = w(u, v, 1)^T \in \mathcal{P}^2, \tag{2}$$

where w is a scalar value and $\mathcal{P}^2$is the 2D projective space. In $\mathcal{P}^2$ all points related by a scale factor are considered equivalent, furthermore the point $(0,0,0) \notin^T \mathcal{P}^2$.

To transform from Euclidean to projective space, an additional dimension is introduced by adjoining a 1 at the end of the vector. To transform back, the projective vector $(a, b, c)^T$ must be divided by the third coordinate to achieve the form $(u, v, 1)^T$ and then drop the third coordinate to get the point in $\mathbb{R}^2$. Finally homogenous points of the form $(u, v, 0)^T$are called points at infinity; these do not have a representation in$\mathbb{R}^2$.

In 3D, points are represented by their Euclidean coordinates

$$\vec{x} = (x, y, z)^T \in \mathbb{R}^3. \tag{3}$$

It is useful to use homogeneous coordinates with the last coordinate initialized by 1. This simplifies the conversion from homogenous to Euclidean coordinates, for the previous point it would be

$$\vec{x} = (x, y, z, 1)^T \in \mathcal{P}^3. \tag{4}$$

2.1.2 Transformations

A transformation is a mapping between two vector spaces that preserves addition and scalar multiplication; in geometry, it is a map between frames of reference. Table 1 shows an example of some simple 2D transformations, these are explained below. They are ordered from lower to higher hierarchy. In other words, all transforms contain the previous ones and all transforms preserve the same geometric properties as the latter ones.

Table 1. *Transformations and summary of characteristics*

Transformation	**Matrix** (for augmented vector) **2D**	**3D**	**Preserves**	**Example**
Translation	$\begin{bmatrix} I_{2x2} & t_{2x1} \\ 0_{2x1}{}^T & 1 \end{bmatrix}_{3x3}$	$\begin{bmatrix} I_{3x3} & t_{3x1} \\ 0_{3x1}{}^T & 1 \end{bmatrix}_{4x4}$	Orientation	
Rigid motion	$\begin{bmatrix} R_{2x2} & t_{2x1} \\ 0_{2x1}{}^T & 1 \end{bmatrix}_{3x3}$	$\begin{bmatrix} R_{3x3} & t_{3x1} \\ 0_{3x1}{}^T & 1 \end{bmatrix}_{4x4}$	Lengths	
Similarity	$\begin{bmatrix} sR_{2x2} & t_{2x1} \\ 0_{2x1}{}^T & 1 \end{bmatrix}_{3x3}$	$\begin{bmatrix} sR_{3x3} & t_{3x1} \\ 0_{3x1}{}^T & 1 \end{bmatrix}_{4x4}$	Angles	
Affine	$[A]_{2x3}$	$[A]_{3x4}$	Parallelism	
Projective	$[H]_{3x3}$	$[H]_{4x4}$	Straight lines	

Translation

A translation is written as $\vec{p}' = \vec{p} + t$ or in matrix form:

$$\vec{p}' = [I_{2x2}\ t_{2x1}]_{2x3}\vec{p}, \tag{5}$$

using Euclidean coordinates or

$$\vec{p}' = \begin{bmatrix} I_{2x2} & t_{2x1} \\ 0_{2x1}{}^T & 1 \end{bmatrix}_{3x3} \overrightarrow{p}, \tag{6}$$

using homogenous coordinates, ***I*** is the 2x2 identity matrix and **0** is the 2x1 zero vector. The 3x3 matrix allows translations to be presented as matrix multiplications and allows integrating operations into a single algebraic notation.

In 3D, translations are represented by a 3x1 vector in the same manner as in 2D, and they can be presented as a matrix operation:

$$\vec{x}' = \begin{bmatrix} I_{3x3} & t_{3x1} \\ 0_{3x1}{}^T & 1 \end{bmatrix}_{4x4} \vec{x}. \tag{7}$$

A translation preserves the orientation of straight lines.

Rigid motions

A rigid motion is a translation and a rotation, in 2D it is written as $\vec{p}' = R\vec{p} + \vec{t}$, or in matrix form

$$\vec{p}' = [R_{\mathbf{2x2}}\ t_{2x1}]_{2x3}\overrightarrow{p}, \tag{8}$$

where $R_{\mathbf{2x2}}$is an orthonormal rotation matrix, $RR^T = I$ and $|R| = 1$.

For an angle θ clockwise,

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{9}$$

This can be written in homogenous notation as

$$\vec{p}' = \begin{bmatrix} R_{\mathbf{2x2}} & t_{2x1} \\ 0_{2x1}{}^T & 1 \end{bmatrix}_{3x3} \vec{p}. \tag{10}$$

In 3D, a rigid body motion is written as $\vec{x}' = R\vec{x} + \vec{t}$, or

$$\vec{x}' = [R_{\mathbf{3x3}}\ t_{3x1}]_{3x4}\vec{x}, \tag{11}$$

where R_{3x3} has the same properties as the 2D rotation matrix. The matrix R is not trivial to parameterize. There are several possibilities, listed below.

Elemental rotations [2]

An elemental rotation is a rotation around one of the axes of the coordinate system in $\mathbb{R}^3$.In matrix form they are:

Rotation of φ degrees around the *x* axis:

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix}. \tag{12}$$

Rotation of θ degrees around the *y* axis:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}. \tag{13}$$

Rotation of ψ degrees around the *z* axis:

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{14}$$

Euler angles [3]

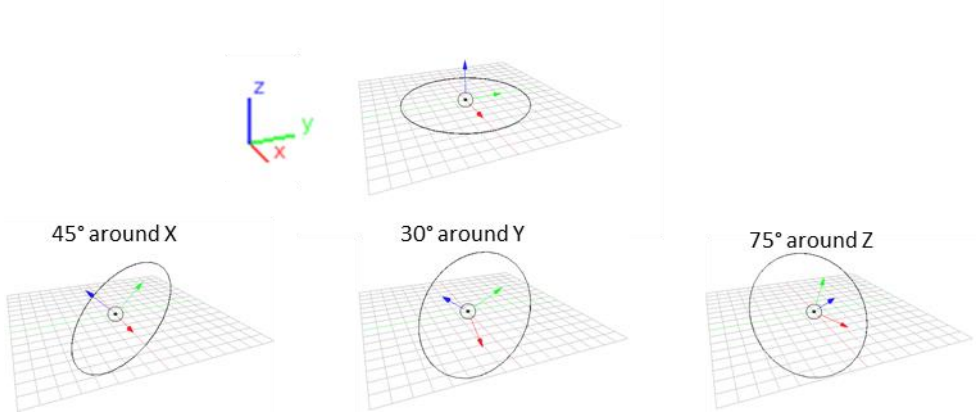

Figure 1. *Sequence of XYZ Euler angles (45°, 30°, 75°)*

Euler introduced a representation for the orientation of rigid bodies in 3D Euclidean space as a sequence of three elemental rotations, see Figure 1. There are 12 sequences of rotations that can be called an Euler angle, and the order influences the result. Also it is not always possible to move smoothly in the parameter space. In particular, Euler

angles are susceptible to a loss in freedom of smooth motion called gimbal lock. Gimbal lock is a configuration in Euler angles where the first local axis aligns with the last global axis, see Figure 2, this removes one degree of freedom form the operator. In practice it creates a point where the camera has to go "around" a point instead of a smooth motion. This problem is avoided by choosing the axes in such a way that the gimbal lock position is rarely or never achieved, for example, a camera will almost never point straight up or straight down, hence an Euler description with the *X* axis in the middle is preferred.

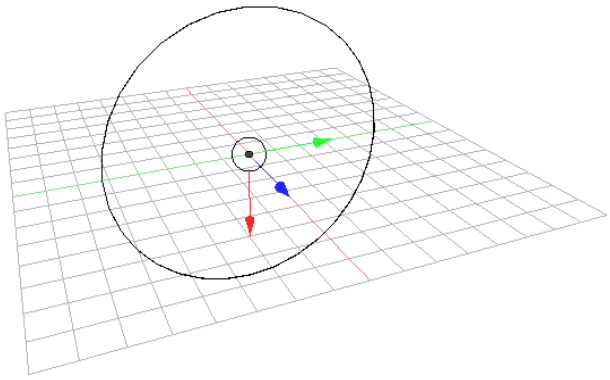

Figure 2. *Gimbal lock. XYZ Euler rotation (0°, 90°, 0°), Note how the local X axis is aligned with the global Z axis. This removes one degree of freedom.*

Axis angle [3]

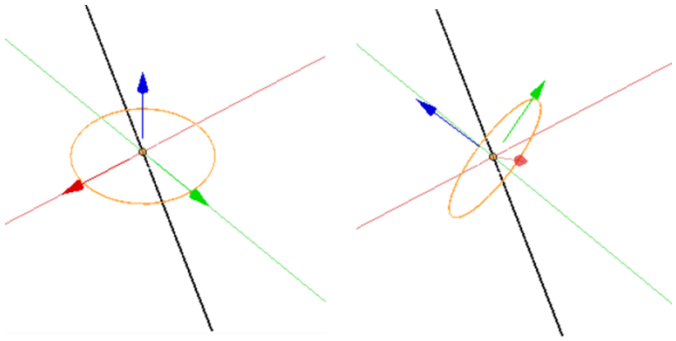

Figure 3. *Axis (0.576,-0.046, g0.816) in black; Left, angle (0°); right, angle (99°), same rotation as achieved in Figure 1.*

Any 3D rotation can be represented by a rotation of an angle θ around a particular is $\vec{n}$, see Figure 3. This representation is minimal, and it does not need any constrains on the parameters. The matrix is derived using the Rodriguez formula [4]

$$R = I + \mathbf{n_{cr}} \sin\theta + \mathbf{n_{cr}^2}(1 - \cos\theta), \tag{15}$$

where $\mathbf{n_{cr}}$ is the matrix version of the cross product operator of the vector $\vec{n}$,

$$\mathbf{n_{cr}} = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}. \tag{16}$$

Unit quaternions [5]

The unit quaternion is derived from the axis-angle representation [3]. It is a vector of length 4 that has norm 1 and is written $\vec{q} = (q_x, q_y, q_z, q_w), \vec{q} = \|1\|$. Quaternions have the advantage that small changes in the rotation will yield small changes in the quaternion, which overcomes some of the problems presented by Euler angles. The representation is unique with the exception that $\vec{q}$ and $-\vec{q}$ represent the same rotation.

The rotation matrix can be obtained from

$$R = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix}. \tag{17}$$

The rigid body transforms preserve the lengths of line segments.

Similarity transform

The similarity transform adds a scale factor to the local coordinates before the rigid motion and is written as $\vec{p}' = sR\vec{p} + \vec{t}$ or in matrix form

$$\vec{p}' = [sR_{\mathbf{2x2}}\ t_{2x1}]_{2x3}\vec{p}, \tag{18}$$

where s is a scalar value.

In 3D, a similar representation is used. Similarity transforms preserve angles between lines.

Affine transform

The affine transform is written as

$$\vec{p}' = \tilde{A}\vec{p}, \tag{19}$$

where $\tilde{A}$ is the augmented matrix

$$\tilde{A} = \begin{bmatrix} A_{2x3} \\ \mathbf{0}_{1x2} \quad 1 \end{bmatrix}, \tag{20}$$

and A_{2x3} is an arbitrary 2x3 matrix.

The 3D representation is similar, using an arbitrary 3x4 matrix.

Perspective transform

Also known as projective transform or homography, it is written as

$$\vec{p}' = H\vec{p}. \tag{21}$$

H is homogeneous, hence two matrices related by a scale factor are considered equivalent.

3D to 2D projections

Now that both 2D and 3D operations are properly defined, it is important to formalize the projection of a 3D scene into a 2D image plane. The most common way to do so is with a linear projection matrix. This matrix can be calculated in several ways, a comparison is listed in [3]. The two matrices used in this work are discussed below. In this section, $\vec{p} = (u', v', 1)^T$ or $\vec{p} = (u, v)^T$ refers to the 2D coordinate and $\vec{x} = (x', y', z', 1)^T$ or $\vec{x} = (x, y, z)^T$ refers to the 3D coordinate.

Orthography

An orthographic projection just drops the *z* component of the $\vec{x}$ vector to produce the 2D vector, in matrix form

$$\vec{p} = [\boldsymbol{I}_{2x2}|\mathbf{0}_{1x2}]\vec{x}, \tag{22}$$

Or

$$\vec{p} = \begin{bmatrix} \boldsymbol{I}_{2x2} & \mathbf{0}_{2x2} \\ \mathbf{0}_{2x1} & 1 \end{bmatrix}\vec{x}. \tag{23}$$

Orthography is used to simplify computations when the camera is far away from the object. In particular it greatly simplifies pose estimation using singular value decomposition [6].

Perspective

The most well-known projection in computer graphics and computer vision is the 3D perspective. It simply divides the 3D coordinates by the *z* component and takes the first two, or in matrix form

$$\vec{p} = [\, \boldsymbol{I}_{3x3} \quad \boldsymbol{0}_{3x1}]\vec{x}. \tag{24}$$

Normalizing $\vec{p}$ to get the Euclidean coordinates yields the perspective transform.

2.2 Pinhole camera model

Since both sensing technologies are, in essence, cameras, it is important to establish a model for this kind of technology. The Pinhole camera model [1] is a simple model that describes the geometric relation between a 3D scene and a 2D image in a camera, this model is presented below.

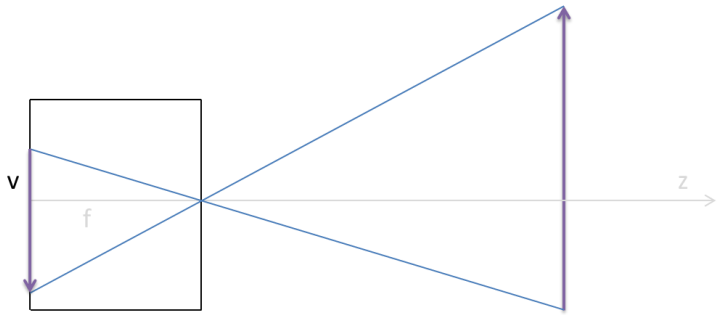

Figure 4. *2D representation of the pinhole camera model*

A camera can be represented by the transformation that maps points in the surface of 3D objects to the image plane. Figure 4 shows the pinhole geometry model in 2D, and the equation is

$$(u, v) = \left(f_x \frac{x}{z}, f_y \frac{y}{z}\right). \tag{25}$$

A simple transformation can place the objects in the correct reference frame. This is where the optical center of the camera is the origin, the *z* component is perpendicular to the image plane and the *y* component is parallel to the *v* component of the image plane, see Figure 5. This transformation is called the extrinsic matrix.

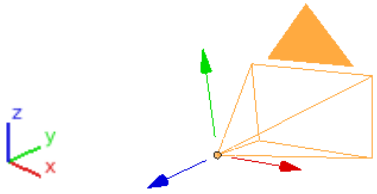

Figure 5. *Left to right; Color convention, Camera with associated axis*

The 4x4 matrix is

$$E = \begin{bmatrix} R_{3x3} & t_{1x3} \\ \mathbf{0}_{3x1} & 1 \end{bmatrix}, \tag{26}$$

R_{3x3} and t_{1x3} describe the necessary rotation and translation to put the points in the camera space. It can be easily calculated by using the pose data, as

$$E = \begin{bmatrix} R_{c\,3x3} & t_{c\,1x3} \\ \mathbf{0}_{3x1} & 1 \end{bmatrix}^{-1}, \tag{27}$$

$R_{c\,3x3}$ and $t_{c\,1x3}$ describe the pose of the camera in the global space.

Next, the points have to be projected to the actual image plane, see Figure 5. This is done with the calibration or intrinsic matrix. The calibration matrix describes the internal parameters of the camera, for example focal length, optical center, etc. in matrix form

$$K = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}, \tag{28}$$

f_u and f_v are the focal lengths for the horizontal and vertical axis, these values are ideally the same, but, if the pixels are not perfect squares, the difference compensates for that distortion factor.(c_u, c_v) are the coordinates of the origin of the image plane. s encodes the possible skew if the sensor is not perpendicular to the optical axis.

Assuming $f_u = f_v = f$ and $s = 0$

$$K = \begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix}. \tag{29}$$

This intrinsic matrix is an implementation of the pinhole model, shown in Equation 4.

Putting them together, the camera matrix is

$$P = \begin{bmatrix} K & \mathbf{0}_{1x3} \\ \mathbf{0}_{3x1} & 1 \end{bmatrix} E. \tag{30}$$

And the complete model is

$$\vec{p} = P\vec{x}. \tag{31}$$

2.3 Depth capture and processing

The system capture is divided into sensing, alignment and filtering. These three stages are first put into the context of the project and then briefly described below.

2.3.1 Sensor

The system uses two kinds of sensing technologies to capture the information from the scene, namely a color (RGB) camera and a depth camera. The technology for a color camera is assumed to be well understood and thus, not explained in this document. The depth camera, on the other hand, is a relatively new technology in which the sensing paradigm has evolved continuously in the last few years. Some of these sensing paradigms including the one used in the current prototype are presented below.

Depth cameras measure the distance from the sensor to each point that projects onto it. This provides a so called 2.5D representation of the scene [8]. Several technologies provide this representation; a comparison is presented in [9]. Below is a list of depth imaging techniques. Since this project uses a Time of Flight camera, this system will be described in detail.

Stereo triangulation [8]

Given two images of the same scene from different points of view, it is possible to estimate the distance and produce a 2.5D image. In order to do this, it is necessary to find the point correspondences between the images. The relative difference between the pixels, called disparity, encodes the distance of the pixel. Specifically the disparity is inversely proportional to the point depth.

Structured light [10]

Structured light is a variation of the stereo triangulation. It uses one or more projectors and one or more cameras. A structured image is projected onto the scene and captured by the cameras. The relative positions of the structure between the projected and the captured images can be treated as the disparity in a stereo system.

LIDAR [11]

By illuminating a pulsed laser to the target and analyzing the reflection (time, phase shift), the distance to the target can be estimated. LIDAR systems use a mechanical scanner to sweep the laser and obtain an image.

Time of flight (ToF) [11]

ToF is similar to LIDAR in the sense that is analyses the reflection and measures the time the signal took to go to the target and return. However, ToF cameras do not need a scanning system; the distances of all the pixels are measured at the same time.

The system built in this project uses the Kinect v2 [12] sensor. This sensor uses indirect ToF technology. In the Kinect, every pixel has two sensors, both turning on (taking photons) and off (rejecting photons) at a high rate (>10GHz).There is a 180 degrees phase shift between the two sensors. This means that when one is on, the other one is off. An IR LED source is flashing in phase with one of the sensors. The light travels to the objects in front and returns to the sensor after a short time (about 7 nanoseconds for an object 1 meter away). This received pulse is read by both sensors as it can be seen in Figure 6. The ratio of intensity of the two sensors is proportional to the distance.

At some point, the distance may be so long that the pulse returns after the second sensor is on and arrives on the first sensor's window, as seen in the last row of Figure 6. This produces an ambiguity. To solve this, the period of the sensors and laser is increased. However, this reduces accuracy due to thermal noise in the sensor. In the real system, the Kinect takes a big period measurement with no ambiguities and then a high precision measurement, using the first to solve the ambiguities of the second.

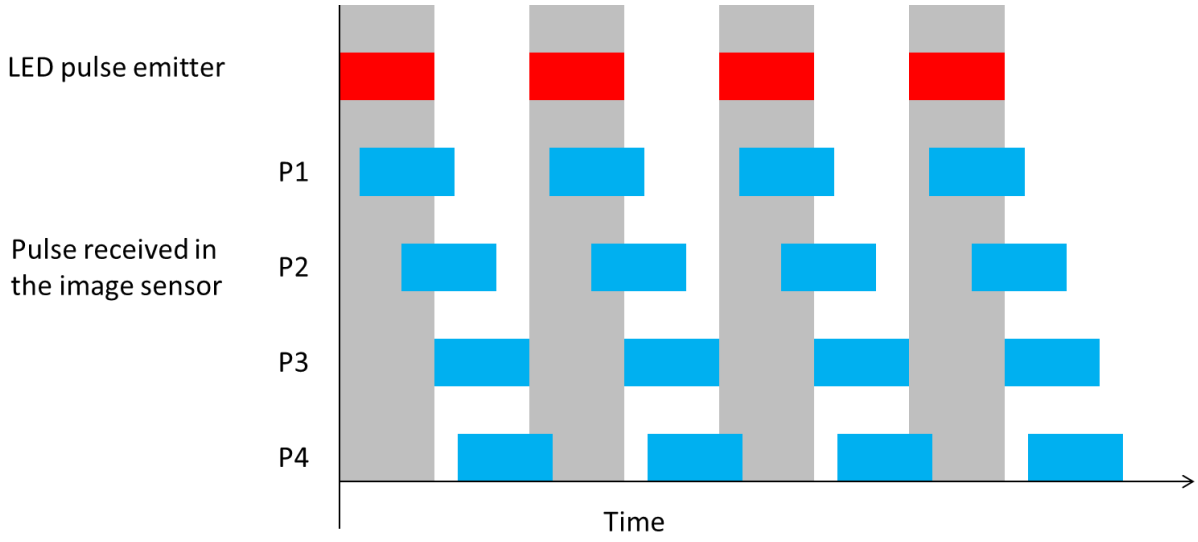

Figure 6. *Timing graph of ToF camera functionality, gray bars indicate the first sensor being on. The distance is proportional to ratio of exposure. P1 is the pulse from an object close and P2 to P4 are increasingly farther away. P4 produces ambiguity as the ratio is very similar to P2 but times are clearly different.*

Regardless of the capture technology, the result is some form of a distance map. This image encodes the Euclidean distance from the sensor to the point. The first step is to transform the distance map (distance to a point) into the depth map (distance to a plane) as seen in Figure 7.

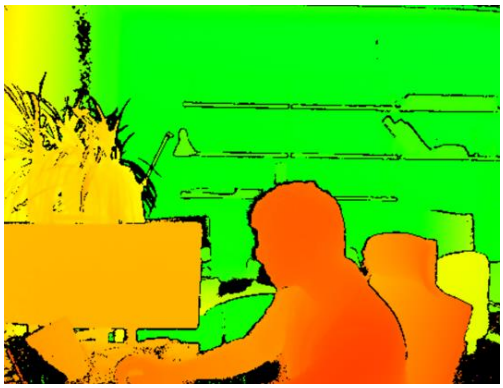

Figure 7. *Sample Depth image, Depth encoded into color*

First both coordinate axes are placed in the same unit, i.e. the correction factor f_y is applied to the coordinates themselves to make both dimensions use the same unit. Hence, both dimensions are projected by the same focal length f_x

$$v' = (v - c_y)\frac{f_x}{f_y}, \tag{32}$$

where $\frac{f_x}{f_y}$ is the ratio of the focal lengths calculated in the calibration, v' it's the new vertical coordinate which has the same units as the horizontal coordinates.

The distance in the focal plane is

$$r = \sqrt{v'^2 + (u - c_x)^2}. \tag{33}$$

Now the triangle similarity shown in Figure 8 produces

$$Z = D\frac{f_x}{\sqrt{f_x^2 + r^2}}, \tag{34}$$

Z is the depth value. If this is done for every pixel, the Depth image is created.

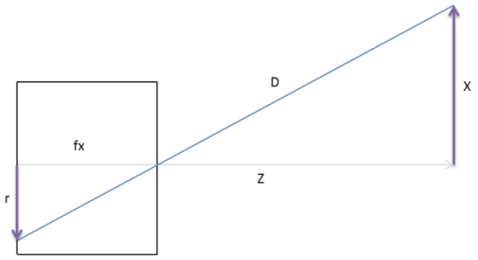

Figure 8. *Similar triangles in pinhole camera model*

2.3.2 Multimodal alignment

Once the color and depth images are obtained they have to be mapped to one another in order to unify the information and make proper use of it. This problem is generally referred to as multimodal alignment. The goal is to combine the information from disparate sources in order to produce a model that is more complete than the sources used separately. The specific stereo-color alignment procedure is presented below.

The depth and color images are two different samples of the same scene from two slightly different perspectives. In order to correctly manipulate the data, it is important to relate the pixels between the images and obtain a unified reconstruction. The images have different points of view, different size and different optical properties. The matching process creates a new depth image where there is a one to one mapping between the color and the depth. It has the following steps:

3D coordinates estimation [1]

Given an image D where each pixel contains the depth information, the calibration matrix K can be inverted and used to estimate the point in 3D space relative to the camera. From Section 2.2, the calibration matrix K fulfils

$$(u', v', w)^T = K(x, y, z)^T, \tag{35}$$

$$(u, v) = \frac{1}{w}(u', v')^T, \tag{36}$$

with the range image,

$$D(u, v) = z = w, \tag{37}$$

so

$$\vec{x} = (x, y, z)^T = D(u, v)K^{-1}(u, v, 1)^T. \tag{38}$$

Projection to camera plane

The set of points are relative to the optical center and orientation of the depth camera. In order to project them to the color camera, they must be transformed into its own reference frame. The transformation between two cameras is a 3D rigid transformation defined by the relative position and orientation of the cameras.

Multi-sensor systems need to know the relative position and orientation of their individual components. These measurements can be done by hand or built in with specified parameters (such is the case of the Kinect). However, there are several calibration methods [3] [19] to achieve this and to account for errors that develop over time.

Let C be the inter-camera calibration matrix, and K_2 the intrinsic matrix of the color camera, then the depth image projected onto the image plane of the color image is

$$(\ddot{u}, \ddot{v}, \dot{w})^T = K_2 C (x, y, z, 1)^T, \tag{39}$$

$$\vec{p} = (\dot{u}, \dot{v})^T = \frac{1}{\dot{w}} (\ddot{u}, \ddot{v})^T, \tag{40}$$

where $\vec{p}$ is the position of the given depth point in the image plane of the color camera.

This stage produces a list of points in the form $(\dot{u}, v; z)^T$

Re-sampling of projected data

The depth image is on the same image plane as the color image, but it's not aligned with the pixel grid. Furthermore, the two cameras rarely have the same resolution. In the Kinect v2 for example, the depth sensor has 512x424 pixels while the color camera has 1920x1080 pixels. Each point in the destination grid has to be estimated from the non-uniformly located points in the depth image. This process is referred to as resampling [3] [14].

Some pixels visible in the color image are not visible in the depth image. These points are called occlusions [1] [13]. There are several ways to handle occlusions. The simplest treatment is to ignore them, if some assumptions are made about the relation between nearby pixels more sophisticated processing can be done. The next section discusses these operations.

2.3.3 Filtering

Once the new depth image is created, some distortions are present. Some are inherent to the depth capture itself (missing pixels) and others are created by the multimodal alignment (occlusions). Apart from those errors there is sensor noise in both images. In order to make the resulting image visually appealing, something has to be done about these errors. This kind of problem is solved by selecting the correct information and filling the wrong information with the best guess. This process is called a filter and it is described below.

In signal processing, a filter is a process that removes an unwanted component [15] from the signal of interest. In this case, the depth signal has noise coming from the sensor [16] and is missing information due to occlusions. The filter's task is to reduce the noise and deal with the occlusions. Below some image processing filtering schemes are described.

For the following descriptions: I is the intensity of the RGB color matrix (one color channel of the luminance channel in luminance-chrominance color space representation), D is the input depth matrix, and C is the output depth matrix. All matrices have the same size. The operator$*$ is the 2D convolution.

Linear filter [15]

The assumption is that each pixel is assumed to have a value similar to its neighbors in space. The filter is implemented with a 2D convolution as

$$C = D * W, \tag{41}$$

where W is the weight matrix. W is a Gaussian matrix, the size and variance of the matrix are parameters that define the effect of the filter. One example of weight matrix is

$$W = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix},$$

which is a Gaussian window with variance 1. Note that W sums to one, and the pixels closer to the center add more information. The Gaussian filter is a linear separable filter; this means that it can be done as a 1D linear filter in one dimension and then repeat in the other dimension. The filter has a blurring effect on the image as seen in Figure 9.

Figure 9. *Left, original image. Right, Gaussian filtered image.*

Median filter [15]

The median filter assumes that the signal is corrupted with outliers rather that with additive noise. It performs the nonlinear median operation over a window. The window size is the design parameter of the filter. The Filter removes outliers and preserves edges; the effect can be seen in Figure 10.

Figure 10. *Left, Image corrupted with salt and pepper noise. Right, Corrupted image after median filter.*

Bilateral filter [17]

The bilateral filter is a nonlinear, edge preserving smoothing filter for images. It is a weighted average filter, similar to the Gaussian explained above. However, the weight is not only based on spatial distance, but also in intensity distance. It can be written as

$$C(p) = \frac{1}{T} \sum_{p_i \in R} D(p_i)\, f(\|D(p_i) - D(p)\|)\; g(\|p_i - p\|), \tag{42}$$

R is the window of the filter; f and g are the kernels of the intensity difference and distance metric respectively. These kernels are arbitrarily selected, a common choice is to use Gaussian functions. *T* is the normalization term

$$T = \sum_{p_i \in R} f(\|D(p_i) - D(p)\|)\; g(\|p_i - p\|), \tag{43}$$

which ensures that the filter preserves image energy.

Joint bilateral filter [18]

The depth image usually suffers from noise and errors close to the edges of the color image. Furthermore there is not enough information to perfectly align them in the depth image. Information from the color image is put into the filter to properly correct those errors and reduce the noise. This information is introduced as

$$C(p) = \frac{1}{T} \sum_{p_i \in R} D(p_i)\, f(\|D(p_i) - D(p)\|)\; g(\|I(p_i) - I(p)\|), \tag{44}$$

with

$$T = \sum_{p_i \in R} f(\|D(p_i) - D(p)\|)\; g(\|I(p_i) - I(p)\|). \tag{45}$$

Note the similarity with the bilateral filter. The intensity distance is done relative to the difference in color rather than in depth. This aligns the edges of the depth image to the color. The smoothing effect and edge aligning are shown in Figure 11.

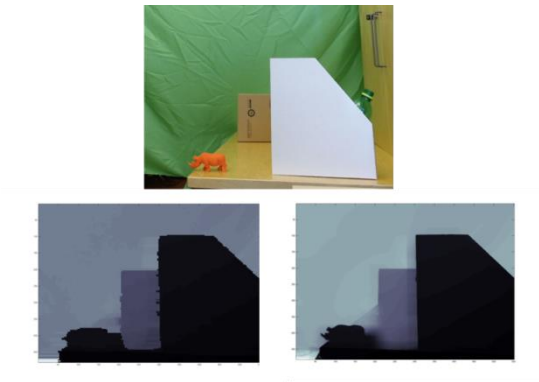

Figure 11. *Joint Bilateral filter, Top, Color image. Bottom, left, original depth image. Right, bilaterally filtered image, note the accuracy of the edges.*

2.4 Optical motion capture

Optical tracking as used in this project consists on identifying the 6 DOF of a rigid body by capturing a constellation of markers through several cameras. The used solution, *Natural Point OptiTrack system* [26], is a proprietary program so the exact algorithm used is not public. However, published solutions to most problems do exist and a set of those is presented below.

2.4.1 Blob detection

In most optical tracking systems, the markers are either passive IR reflectors or active IR emitters. These are captured via a camera with an IR-pass filter, the end result being an image similar to Figure 12.

Figure 12. *Image captured by the IR camera, the markers are not brighter than the rest of the scene due to overpower of IR LEDs, this was done to make the image clearer.*

The first step is to eliminate anything that is not a marker. In order to do this, we can assume that the markers are brighter than anything else in the scene. This assumption holds if the space and equipment are properly calibrated and setup. Given this, a thresholding of the image should separate the markers and other IR bright objects from the rest of the scene. The thresholding value is dependent on the lighting conditions and can be adjusted manually to fit the situation. An example of the segmentation is shown in Figure 13.

Figure 13. *Image segmented, bright spot on top is another camera, and the constellation is in the middle.*

After segmentation, the image is either binary (full thresholding) or grayscale (keep gray values of markers). The markers appear as bright ellipsoids in contrast with a dark background. These are referred to as blobs. The task is to extract the coordinates of the centers of the blobs.

For this there are many solutions, a detailed comparison is given in [28]. Some of the main ideas and their downsides are presented in Table 2

Table 2. *Blob detection algorithms and their known issues*

Algorithm	**Known problems**
Matched filter/template	Only pixel precise results.
Watershed detection	Noise sensitive.
Structure tensor analysis	Only circular structures and computationally expensive.
Scale space analysis	Computationally expensive.
Sample edges and use shape descriptors	Computationally expensive.

Since the user typically can control the lighting conditions, the power of the noise can be controlled. Hence, a simple solution can be used.

A simple solution proposed in [32] is to use a matched filter to initially estimate the center, and then if a blob is found, calculate the center of mass of brightness within the small window. This can be done very fast and can be integrated with the aforementioned segmentation. However, issues with this solution arise as the blobs get closer to each other, but given enough cameras, this problem can be ignored.

2.4.2 3D reconstruction

The input for 3D reconstruction is the list of blob locations for each camera. Assuming passive markers, correspondences between them are still unknown. This is a classic multiple view geometry problem and is referred to as triangulation [29]. The projection matrices for each camera are known, i.e. that the system is calibrated.

As it was described in Section 2.3, each point on an image plane corresponds to a line that passes through the optical center and the point in 3D space(see Figure 14); hence, one would expect all lines coming from different cameras corresponding to the same point to intersect in the 3D location of the marker. However, due to noise and calibration errors, these rays do not always intersect.

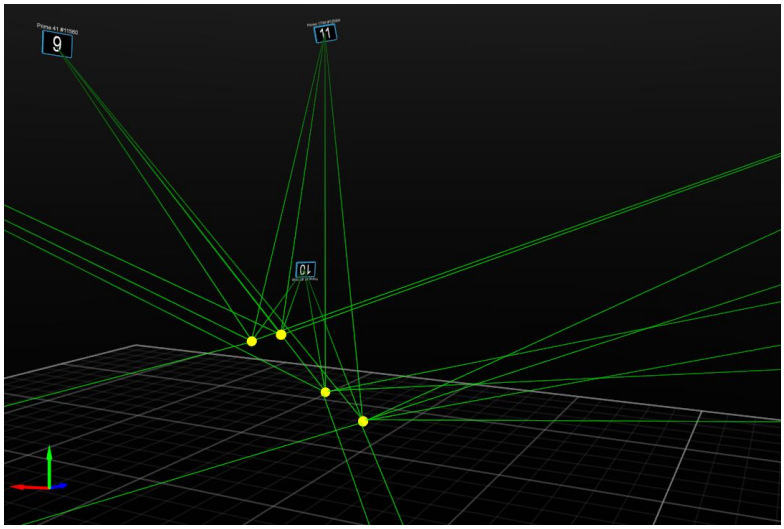

Figure 14. *Points in 3D space and the projected rays into each camera.*

The error is calculated as the distance between the points in the image plane and the projections of the estimated solution on the same image plane. This is called the reprojection error. If this error is minimized, the estimator will be the maximum likelihood estimator (MLE)

The setup for the Iterative least squares (LS) solution is stated below.

Let

$$\vec{x} = (x, y, z, 1)^T, \tag{46}$$

be the position of the center of the marker in homogeneous space. And let

$$\vec{u} = (u, v, 1)^T, \tag{47}$$

be the coordinates of the projection into a camera plane. The camera matrix *P* describes the mapping from the 3D space to the image plane as in

$$\vec{u} = P\vec{x} = (u', v', w)^T = \frac{1}{w}(u, v, 1)^T. \tag{48}$$

Let $\overrightarrow{P_i}$ be the ith row of the matrix *P*, *P* can be written as:

$$P = \begin{bmatrix} \overrightarrow{P_1} \\ \overrightarrow{P_2} \\ \overrightarrow{P_3} \end{bmatrix}. \tag{49}$$

Equation (49) can be divided into 3 equations as follows:

$$\frac{u}{w} = \overrightarrow{P_1}\vec{x}, \tag{50}$$

$$\frac{v}{w} = \overrightarrow{P_2}\vec{x}, \tag{51}$$

$$\frac{1}{w} = \overrightarrow{P_3}\vec{x}, \tag{52}$$

Substituting w from the third equation into the first two the following system is left:

$$\begin{cases} \overrightarrow{P_3}.\vec{x} * u = \overrightarrow{P_1}\vec{x}, \\ \overrightarrow{P_3}.\vec{x} * v = \overrightarrow{P_2}\vec{x}. \end{cases} \tag{53}$$

This system has 3 unknowns (x, y, z) but only two equations, so one camera is not enough. By using *N* cameras, a system of *2N* equations and two unknowns is built. Hence, an overdetermined system is built and any LS solution (SVD, Euler) can produce a solution. However, that solution minimizes the following cost (squared) for each coordinate in the image plane

$$Cost1 = \left(\overrightarrow{P_3}.\vec{x} * u - \overrightarrow{P_1}.\vec{x}\right)^2 + \left(\overrightarrow{P_3}.\vec{x} * v - \overrightarrow{P_2}.\vec{x}\right)^2. \tag{54}$$

This has no geometric meaning and is different from the MLE cost. The MLE cost is the squared distance (D) of the point $\vec{u}$ and the projection of $\vec{x}$, or in equation form

$$D = \vec{u} - P * \vec{x}. \tag{55}$$

The MLE cost can be written as:

$$Cost2 = \left(u - \frac{\overrightarrow{P_1}.\vec{x}}{w}\right)^2 + \left(v - \frac{\overrightarrow{P_2}.\vec{x}}{w}\right)^2. \tag{56}$$

By substituting Equation 53 in Equation 56:

$$Cost2 = \left(\frac{\overrightarrow{P_3}.\vec{x} * u - \overrightarrow{P_1}.\vec{x}}{w}\right)^2 + \left(\frac{\overrightarrow{P_3}.\vec{x} * v - \overrightarrow{P_2}.\vec{x}}{w}\right)^2. \tag{57}$$

Minimizing this cost attains the MLE of the system. Moreover, unlike cost1, cost2 is projection invariant. However, w is not known, and since it's inversely related to the distance between the image plane and the center of the marker, finding the right *w* is the same as locating the point in 3D space.

An iterative solution was proposed by [20] where *w* is initialized as 1 for every camera and then the LS estimation is performed. Then the result is used to find *w* for every camera and the LS estimation is performed again. The result converges to the MLE in most of the configurations. Some caveats are particular to 2 camera systems where points close to the epipolar line cause the system to not converge.

In systems where more than one marker is seen at the same time by all cameras, matching (labeling the dots in every camera) is performed by minimizing the average cost for all points.

2.4.3 Model fitting and pose estimation

After the 3D position of the markers is found, the next step is to estimate the actual position and orientation of the objects that have the markers attached to them. This process has two different steps: model fitting and pose estimation. In model fitting, a set of points is divided into clusters and each cluster is matched with a corresponding structure. In pose estimation, the position and orientation of a structure with respect to a fixed axis is estimated.

Model fitting is a common problem in computer vision. Its goal is to map an unknown model to a template. There are many available solutions that are proven to work [21]. Since in this context there is always only one model present, the solution is not relevant and thus, not presented. The second half, pose estimation, is also a common computer

vision problem. It is generally considered to be solved, but there is no single solution that fits all requirements. For large clouds, iterative closest point [22] is commonly used; for small clouds, like the ones present in this system, analytical methods [3] [23] are a feasible solution.

The following derivation uses Euclidean geometry (see Section 2.1), so points in 3D space are represented by their (x, y, z) coordinates. Rotations are represented by a 3x3 matrix *R* such that

$$R^T R = R R^T = R^{-1} R = I, \tag{58}$$

$$\det(R) = 1, \tag{59}$$

with I being the identity matrix and $\det(\,)$ being the determinant of a matrix. A translation is represented by a vector *t* with the corresponding translations for each coordinate. Finally a scale is represented by a scalar s.

Let A and B be two sets of points in two reference frames that are related by a translation *t* and a rotation. Since both reference frames have the same scale,

$$b_i = R a_i + t, \qquad b_i \epsilon B, a_i \epsilon A. \tag{60}$$

By using least squares, the function to minimize is

$$\frac{1}{n} \sum_{i=1}^{n} (R a_i + t - b_i)^2, \tag{61}$$

where

$$n = |A| = |B|. \tag{62}$$

In order to calculate *t*, we calculate the mean vectors.

$$\bar{a} = \frac{1}{n} \sum_{i=1}^{n} a_i \,, \tag{63}$$

$$\bar{b} = \frac{1}{n} \sum_{i=1}^{n} b_i \,. \tag{64}$$

Then it can be determined as

$$t = \bar{b} - R\bar{a}. \tag{65}$$

Substituting Equation (65) into Equation (61) yields

$$\frac{1}{n}\sum_{i=1}^{n}\left(Ra_i + \bar{b} - R\bar{a} - b_i\right)^2. \tag{66}$$

Now both reference frames are translated to have their origin in the mean of their respective point set, in equation form,

$$\begin{aligned} a_i' &= a_i - \bar{a}, \\ b_i' &= b_i - \bar{b}. \end{aligned} \tag{67}$$

Substituting Equation (67) into Equation (66) produces

$$\frac{1}{n}\sum_{i=1}^{n}(b_i' - Ra_i')^2 = \frac{1}{n}\sum_{i=1}^{n}(b_i' - Ra_i')^T(b_i' - Ra_i'), \tag{68}$$

$$\frac{1}{n}\sum_{i=1}^{n}(b_i' - Ra_i')^T(b_i' - Ra_i') = \frac{1}{n}\sum_{i=1}^{n}\left(b_i'^T b_i' - b_i'^T Ra_i' - (Ra_i')^T b_i' + (Ra_i')^T Ra_i'\right). \tag{69}$$

Given the following equivalences, from Equation (58) ,

$$(Ra_i')^T Ra_i' = a_i'^T R^T Ra_i' = a_i'^T a_i', \tag{70}$$

$$(Ra_i')^T b_i' = b_i'^T Ra_i', \tag{71}$$

Equation 69 can be written as

$$\frac{1}{n}\sum_{i=1}^{n}\left(b_i'^T b_i' + a_i'^T a_i' - 2 * b_i'^T Ra_i'\right). \tag{72}$$

Since $b_i'^T b_i'$ and $a_i'^T a_i'$ are constants, minimizing Equation (71) is the same as maximizing

$$\frac{1}{n}\sum_{i=1}^{n}\left(b_i'^T Ra_i'\right). \tag{73}$$

Expanding and summing gives the following function

$$\frac{1}{n}\sum_{i=1}^{n}\left(b_i'^{T} R a_i'\right) = tr\left\{R^T \frac{1}{n}\sum_{i=1}^{n} b_i'\, a_i'^{T}\right\} = tr\{R^T C\}, \tag{74}$$

where *tr*{} refers to the trace of a given matrix and C is the cross-dispersion or correlation matrix, defined as

$$C = \frac{1}{n}\sum_{i=1}^{n} b_i'\, a_i'^{T}. \tag{75}$$

Now the singular value decomposition [24] is calculated as

$$C = UWV^T, \tag{76}$$

where *U* and *V* are orthogonal matrices and *W* is diagonal matrix containing the singular values of *W*. substituting these into the function to maximize

$$tr\{R^T UWV^T\} = tr\{V^T R^T UW\}. \tag{77}$$

Grouping into a new matrix $Q = V^T R^T U$,

$$tr\{V^T R^T UW\} = tr\{QW\}. \tag{78}$$

Q must be orthogonal as it is a product of orthogonal matrices. As such, its diagonals must be all less than or equal to one. Since W is a diagonal matrix and Q is orthogonal, the value of the trace reaches a maximum when Q is an Identity matrix, hence

$$I = V^T R^T U. \tag{79}$$

Since all matrices are orthogonal, Equation (78) rearranges to

$$VIU^T = VV^T R^T UU^T, \tag{80}$$

$$VU^T = R^T, \tag{81}$$

$$R = UV^T. \tag{82}$$

R is the LS estimate of the rotation matrix that matches the model constellation with the sensed one.

2.5 Multilateration

The system integrates the internal calibration of the cameras with the external characteristics of the marker constellation; this requires a relation between position data from the motion capture and distance data from the ToF camera. This problem can be formulated as multilateration, explained below.

The multilateration problem is common in big scale positioning systems [25]. The problem is the estimation of a point given its distance to a set of known points. The problem can be seen as finding the intersection of a set of spheres in 3D space, see Figure 15. In practice, since every sensor has an error attached to it, such a point may not exist.LS estimation is performed to calculate the point.

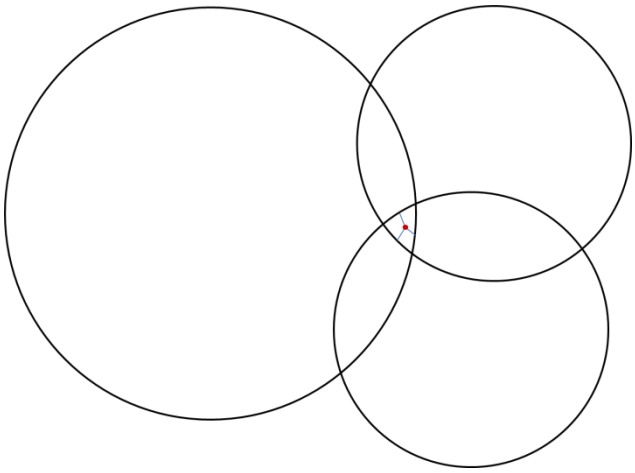

Figure 15. *Multilateration in 2D, red point is estimate of intersection between three circles.*

Let the point $\vec{x} = (x, y, z)^T$ be an unknown point and the set of n known points $\vec{x}_i$, $i < n$, assume the distance

$$d_i = \|\vec{x} - \vec{x}_i\|, \tag{83}$$

is also known.

Stacking Equation (82) for all points yields

$$\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} \sqrt{(x-x_1)^2 + (y-y_1)^2 + (z-z_1)^2} \\ \sqrt{(x-x_2)^2 + (y-y_2)^2 + (z-z_2)^2} \\ \vdots \\ \sqrt{(x-x_n)^2 + (y-y_n)^2 + (z-z_n)^2} \end{bmatrix}. \tag{84}$$

Squaring both sides and expanding the equations

$$\begin{bmatrix} d_1^2 \\ d_2^2 \\ \vdots \\ d_n^2 \end{bmatrix} = \begin{bmatrix} x^2 - 2xx_1 + x_1^2 + y^2 - 2yy_1 + y_1^2 + z^2 - 2zz_1 + z_1^2 \\ x^2 - 2xx_2 + x_2^2 + y^2 - 2yy_2 + y_2^2 + z^2 - 2zz_2 + z_2^2 \\ \vdots \\ x^2 - 2xx_n + x_n^2 + y^2 - 2yy_n + y_n^2 + z^2 - 2zz_n + z_n^2 \end{bmatrix}. \tag{85}$$

Subtracting the last equation from all the rest eliminates the squared unknowns; the last equation is then removed from the system,

$$\begin{bmatrix} d_1^2 - d_n^2 \\ d_2^2 - d_n^2 \\ \vdots \\ d_{n-1}^2 - d_n^2 \end{bmatrix} = \begin{bmatrix} 2xx_n - 2xx_1 + x_1^2 - x_n^2 + 2yy_n - 2yy_1 + y_1^2 - y_n^2 + 2zz_n - 2zz_1 + z_1^2 - z_n^2 \\ 2xx_n - 2xx_2 + x_2^2 - x_n^2 + 2yy_n - 2yy_2 + y_2^2 - y_n^2 + 2zz_n - 2zz_2 + z_2^2 - z_n^2 \\ \vdots \\ 2xx_n - 2xx_{n-1} + x_{n-1}^2 - x_n^2 + 2yy_n - 2yy_{n-1} + y_{n-1}^2 - y_n^2 + 2zz_n - 2zz_{n-1} + z_{n-1}^2 - z_n^2 \end{bmatrix}. \tag{86}$$

Moving the known values to the left side of the equation produces the system

$$\begin{bmatrix} d_1^2 - d_n^2 - x_1^2 + x_n^2 - y_1^2 + y_n^2 - z_1^2 + z_n^2 \\ d_2^2 - d_n^2 - x_2^2 + x_n^2 - y_2^2 + y_n^2 - z_2^2 + z_n^2 \\ \vdots \\ d_{n-1}^2 - d_n^2 - x_{n-1}^2 + x_n^2 - y_{n-1}^2 + y_n^2 - z_{n-1}^2 + z_n^2 \end{bmatrix} = \begin{bmatrix} 2xx_n - 2xx_1 + 2yy_n - 2yy_1 + 2zz_n - 2zz_1 \\ 2xx_n - 2xx_2 + 2yy_n - 2yy_2 + 2zz_n - 2zz_2 \\ \vdots \\ 2xx_n - 2xx_{n-1} + 2yy_n - 2yy_{n-1} + 2zz_n - 2zz_{n-1} \end{bmatrix}. \tag{87}$$

which can be rewritten as

$$\begin{bmatrix} d_1^2 - d_n^2 - x_1^2 + x_n^2 - y_1^2 + y_n^2 - z_1^2 + z_n^2 \\ d_2^2 - d_n^2 - x_2^2 + x_n^2 - y_2^2 + y_n^2 - z_2^2 + z_n^2 \\ \vdots \\ d_{n-1}^2 - d_n^2 - x_{n-1}^2 + x_n^2 - y_{n-1}^2 + y_n^2 - z_{n-1}^2 + z_n^2 \end{bmatrix} = 2 \begin{bmatrix} x_n - x_1 & y_n - y_1 & z_n - z_1 \\ x_n - x_2 & y_n - y_2 & z_n - z_2 \\ & \vdots & \\ x_n - x_{n-1} & y_n - y_{n-1} & z_n - z_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{88}$$

The matrix system has the classic form

$$2A\vec{x} = r \tag{89}$$

Since there are $n - 1$ equations, at least 4 points are necessary to estimate the correct position; by using the SVD and the Pseudoinverse, any number of points larger than 3 can be used.

2.6 Computer Graphics and OpenGL

2.6.1 Overview

Computer graphics is a broad term that encompasses a large variety of tools methods and approaches. In this document, the focus is on the description and display of 3D objects.

The model describes the 3D geometry in a way that the computer can understand. There are many different ways of representing models, but in this work the focus is on polygon meshes and point clouds.

A polygon mesh is a set of polygons that approximate the surface to be modeled, see Figure 16. Triangles are often used for their generality and simplicity; however, many different primitive polygons exist. A triangle is described by the position of is vertices. This representation is the most used to described virtually generated objects.

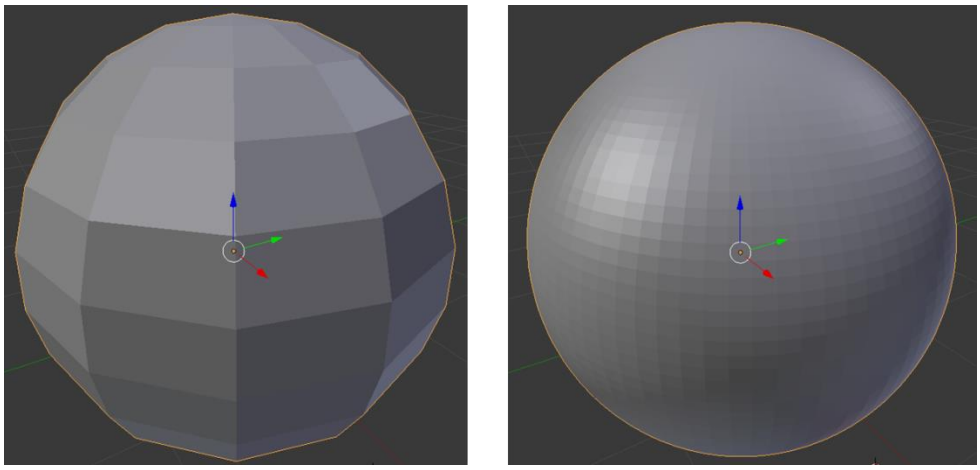

Figure 16. *Polygon mesh with rectangles as primitive polygons. Left, low polygon count. Right, high polygon count.*

Point clouds are sets of points that sample the surface geometry, see Figure 17. Each point is rendered as a disc in the camera. This representation is the natural description of most 3D sensing technologies, including the Microsoft Kinect.

Both descriptions start as a set of points, these are called vertices. Apart from the geometry, each vertex can have more properties. These include texture coordinates, normals, bump map data, etc.

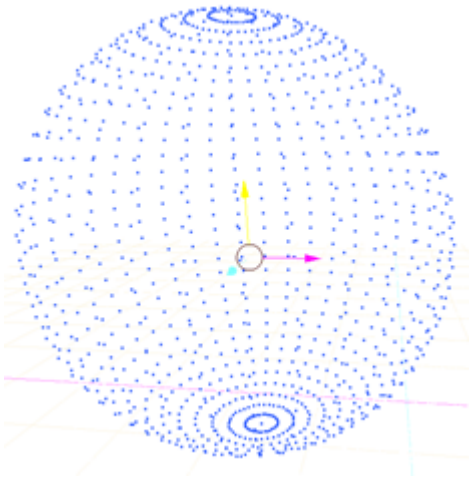

Figure 17. *Point cloud of a sphere*

Triangle meshes model smooth surfaces as sets of plane sections. Point clouds model them as a dense set of points. The more triangles, the more refined is the mesh and the less blocky it appears. There are smoothing [31] and interpolation [30] algorithms that improve the approximation to some extent.

These concepts are very general and there are many implementations of a basic computer graphics pipeline. Next, the OpenGL implementation of these concepts is presented.

2.6.2 OpenGL primitives

OpenGL is an API designed to render 2D and 3D vector graphics, it is used to interact with a GPU and achieve hardware accelerated rendering.

The OpenGL environment contains many methods and classes, a complete description can be found at [33]. In this document, the modern approach to basic mesh rendering described in [34] is presented

Before defining the primitives, the concept of Vertex Stream is introduced. A Vertex stream is an ordered list of vertices; these can be created by the user in a vertex specification of created by a geometry shader.

The vertex specification allows the user to create a set of vertices and add attributes defined by the user. These attributes define what information the vertex needs to properly render.

Primitives

The primitive used tells OpenGL how to interpret the vertex stream and what to draw. This interpretation can be points, triangles or lines. Even these can be interpreted in many ways; for example the triangles can be disconnected or connected. OpenGL con-

tains plenty of primitives. However, for the scope of this document, the three more basic are briefly described.

Each primitive is defined, its interpretation of the vertex stream introduced and the rasterization procedure stated. Rasterization is the division of a primitive into fragments. Each fragment is usually pixel sized and inherits properties from its parent primitive.

The point primitive interprets each vertex as a point, and attach to it a point sprite. Each point is rasterized as a square of a user defined size. The size describes the side of the square in pixels and the position is the center of such square.

The line primitive interprets vertices as ends of line segments; they can be defined to be separate, a chain, or a loop. Lines are rasterized as rectangles, whose width in pixels is defined by the user. The properties of the fragments are interpolated from the properties of the vertices.

A triangle is a primitive formed by three vertices; they can be interpreted in several ways, separate, stripped or fanned. The order of listing the three vertices defines the direction of the face. Usually only the front of triangles are rendered, however this can be changed by the user. The front face is rasterized as the pixels inside the area of the triangle. The properties of the vertexes are interpolated to become the properties of the fragments.

A texture is an openGL object that contains one or more images; an image is an array of pixels. Textures are mapped into primitives via the texture coordinate attached to the vertices. The fragments inherit the texture coordinates of their parent primitive trough an interpolation of the texture coordinates.

2.6.3 GLSL shaders

A shader is a program that runs on a parallel computing device, a GPU in this case. There are many kinds but for the scope of this document, only two will be presented, the vertex and fragment shaders.

The vertex shader is a program that runs on every vertex in a given vertex stream. This modification is usually connected to attributes or textures. This shader usually moves the objects and places them in the camera plane as a 2D vertex stream. The output stream is converted to the primitive and rasterized to produce a fragment set. This fragment set is the input of a fragment shader.

The fragment shader is a program that operates on every fragment. It usually contains the texturing and coloring algorithms. The user can choose to dump that texture into the screen and see the result and depending on the use it may be correct. In order to keep

the occlusions correct, several solutions may work. The most basic is called the painters algorithm, this simply draws the farther objects first and then the closer ones on top. This works but is not easy to integrate with rapidly changing environments. The other popular algorithm is called Z-buffering. Z-buffering consists in saving the depth value (distance from the point to the camera plane). Then on a given pixel, only the fragment that is closer to it is rendered.

3. IMPLEMENTATION

The system was implemented in the CIVIT motion capture studio. This section describes the system from three perspectives: Hardware, software and Algorithm. The descriptions are summarized into block diagrams with a reasonable level of abstraction. More literal descriptions are present in the documentation of the code.

The hardware composition is shown in Figure 18. In order to describe the internal system, the algorithm is presented separate from the software used. The algorithm description is grouped in information processing units, these are not necessary implemented in the same system or covered in the same block in the computational description.

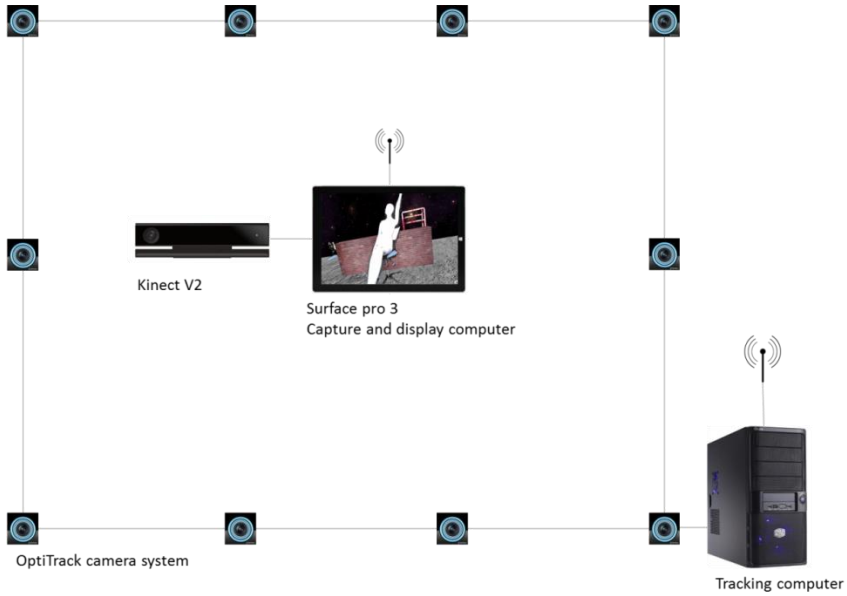

Figure 18. *Hardware diagram, only information links are shown, power connections are ignored.*

3.1 Algorithm description

In order to keep track of the processing, three different coordinate spaces will be defined.

The camera coordinate space (CCS) is fixed to the moving camera. Its origin is in the optical center, the *z* coordinate is perpendicular to the image plane and the *y* coordinate is in the vertical direction of the image plane.

The real coordinate space (RCS) is defined by the motion capture calibration and is static. The positions and orientations given by the motion capture system are all in the RCS.

The virtual coordinate space (VCS) is defined by the modelling tool used to create the virtual objects. All the virtual objects are in the VCS.

Figure 19 shows the algorithmic block diagram which shows the flow of information from the sensor to the display.

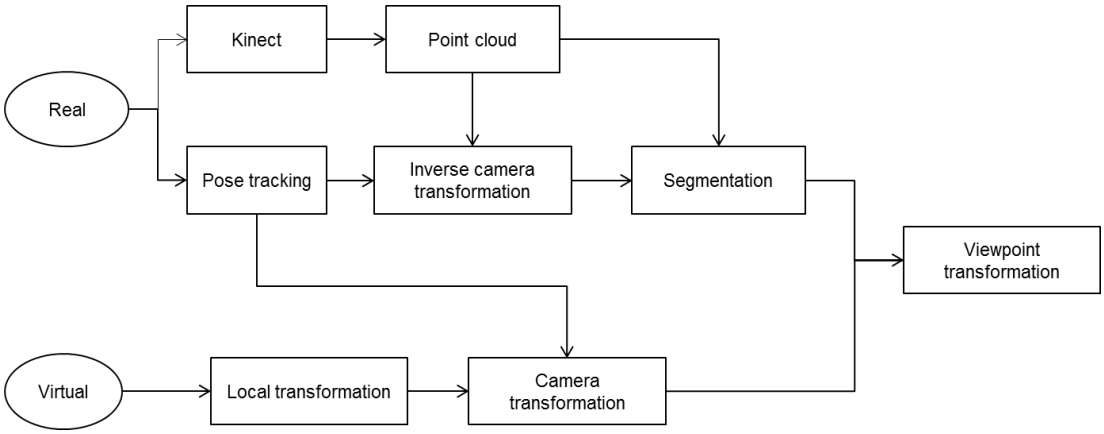

Figure 19. *Algorithm block diagram.*

Real

The Real space is the volume inside the capture studio. It contains all objects inside the volume, the marker constellations and the RCS definition. Section **Error! Reference source not found.** contains images of the studio used in this case for capture.

The motion capture software is calibrated to have all camera poses relevant to an arbitrary reference frame selected by the user; this becomes the origin of the RCS.

Kinect

The Kinect V2 device is used to sample the real scene. A 1920x1080 color camera image and a 512x424 ToF depth camera image are produced. The Kinect delivers the Color image and the aligned and resampled depth image. The calibration of the system has three different phases presented below.

Pose Tracking

OptiTrack camera system is used to estimate the pose of the camera. The camera is assumed to be a rigid body, its 6 DoF pose is tracked using optical motion capture (see Section 2.4). The marker constellation used in this case is shown in Section **Error! Reference source not found.**.

The calibration of the pose tracking is done according to the OptiTrack system instructions. The position of the cameras is estimated according to a user defined point in the space.

A final calibration is done to estimate the position of the camera sensor relative to the marker constellation. This document refers to this process as the external calibration. This calibration is designed for this system and uses the multilateration estimation (see Section 2.5). A brief description is presented below.

External Calibration

Different from extrinsic calibration, the goal of the external calibration is to find the position of the camera sensor. The position is in reference to the marker constellation that is fixed to the body of the system. Since the final image is reprojected from the color camera point of view, the output is an estimation of the color camera sensor relative to the marker constellation.

To setup the calibration, the camera is fixed to a stable mount. A second constellation of markers is placed in the visible space in front of the camera; the configuration used is shown in Section **Error! Reference source not found.**.

Table 3 shows the data necessary to perform the calibration and how to obtain it.

Table 3. *Data to perform external calibration*

data	Format	description	Source
Kinect capture	$K = (u, v, z)$	(u, v) are the coordinates of the visible markers in the depth image. Z is the depth measurement.	Kinect depth image
Camera constellation N markers	$C_i = (x, y, z)$ $i < N$	The list of positions of the camera markers	OptiTrack camera system
Calibration constellation M markers	$S_i = (x, y, z)$ $i < M$	The list of positions of the second markers	OptiTrack camera system
Intrinsic and Stereo calibration data	List of parameters $[c_x , c_y]$ is the principal point. f_x , f_y are the focal lengths.	The result of the previous calibration steps	Camera and stereo calibration

First, the points in the Kinect capture have to be transformed back into distances to the sensor (see Section 2.3). Using triangle similarity in Figure 20, the distance can be calculated as

$$D(\mathrm{u},\mathrm{v},\mathrm{Z}) = \left\| \left(Z\frac{u-c_x}{f_x}, Z\frac{v-c_y}{f_y}, Z \right) \right\|, \tag{90}$$

where the calibration data is from the depth camera.

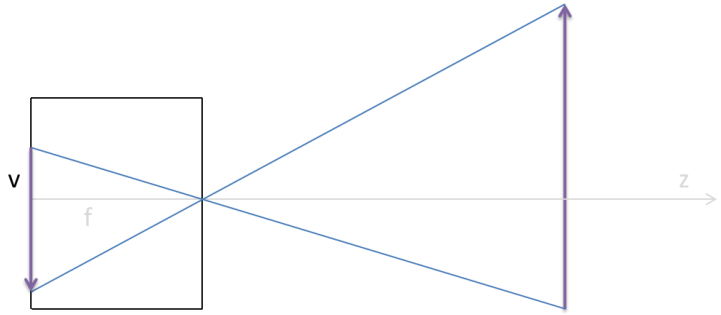

Figure 20. *2DPinhole camera model*

The set of points of the calibration constellation *S* and their corresponding distances can be treated as a set of spheres. The center of the sensor should intercept all of them. To find the MLE of the intersection of the spheres, we use the multilateration LS algorithm (see Section 2.5)

$$\begin{matrix} D \\ \mathrm{S} \end{matrix} \rightarrow [Mlat] \rightarrow \vec{d}, \tag{91}$$

where $[Mlat]$ is the LS algorithm for the multilateration problem, and $\vec{d}$ is the MLE estimate of the position of the sensor.

Using the translation parameter of the stereo calibration$\vec{t}$, the position of the color sensor $\vec{x}$ can be estimated as

$$\vec{x} = \vec{d} + \vec{t}. \tag{92}$$

The translation parameter has to be given in the correct direction (from depth to color). If it's not, inverting the extrinsic calibration matrix will yield the correct translation vector.

Finally, the camera marker constellation is redefined with the found camera sensor as the origin. Since the orientation remains the same, it is done by subtracting the sensor position from the marker position,

$$\dot{C}_i = C_i - \vec{x} \quad , \quad i < N, \tag{93}$$

where $\dot{C}$ is the corrected camera constellation.

The set of corrected camera marker positions can be put into the tracking software. This redefines the rigid body with the estimated camera sensor as the origin.

Real Point Cloud

The point cloud is built by calculating the 3D coordinates $[x, y, z]^T$ of the points in the depth image (see Section 2.2)

$$\vec{p} = [x, y, z]^T = \left[Z\frac{u - c_x}{f_x}, Z\frac{v - c_y}{f_y}, Z \right]^T, \tag{94}$$

where (u, v) are the coordinates of the pixel, Z is the depth measurement and the rest are the parameters extracted from the calibration matrix.

The depth image is already aligned to the color image in the earlier processing. Hence, the texture mapping is the same as the grid of the image. The output is the point cloud in the CCS.

Inverse camera transform

Some points in the real point cloud will be outside of the capture volume. These points are of no interest, and should not be rendered. To find out which points are to be ignored all the points are transformed from the CCS to the RCS using

$$\vec{p}' = P\vec{p} \qquad P = \begin{bmatrix} R & \vec{t} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{95}$$

Cropping

Points whose corresponding positions in the RCS are outside of the capture volume are discarded from the render. The rest are rendered normally.

Virtual

The virtual space is a set of 3D objects, described in mesh format (see Section 2.6). Each object has a texture and a rigid body position associated to it. The set of objects forms the virtual scene and is defined in the VCS.

Local transformation

Virtual objects are transformed from the VCS to the RCS using their own associated rigid body matrix.

$$\vec{v}' = T\vec{v}, \tag{96}$$

where $\vec{v}$ is the vector of coordinates of a vertex in the mesh description and $\vec{v}'$ in the vector of coordinates in the RCS. This builds the virtual room.

Camera transformation

The camera transformation puts the virtual objects in the CCS

$$\vec{\vec{v}} = P^{-1}\vec{v}', \tag{97}$$

where *P* is defined in Equation 95.

Viewpoint transformation

The viewpoint transformation puts the objects in the image plane of the virtual camera. Since both the real and the virtual objects are in the CCS, only the intrinsic matrix is necessary.

Render

The pixels are painted with the corresponding texture from the object. Z buffering lets the renderer know which object is closer to the camera and which ones are occluded. This blends the virtual meshes and the point cloud in the same 3D space with correct occlusion order.

3.2 Software implementation

The implementation of the algorithm presented in the previous chapter has several distinct parts. Figure 21 shows the block diagram that describes the system. Commercial SDKs and software are considered black boxes and are only described in terms of inputs and outputs.

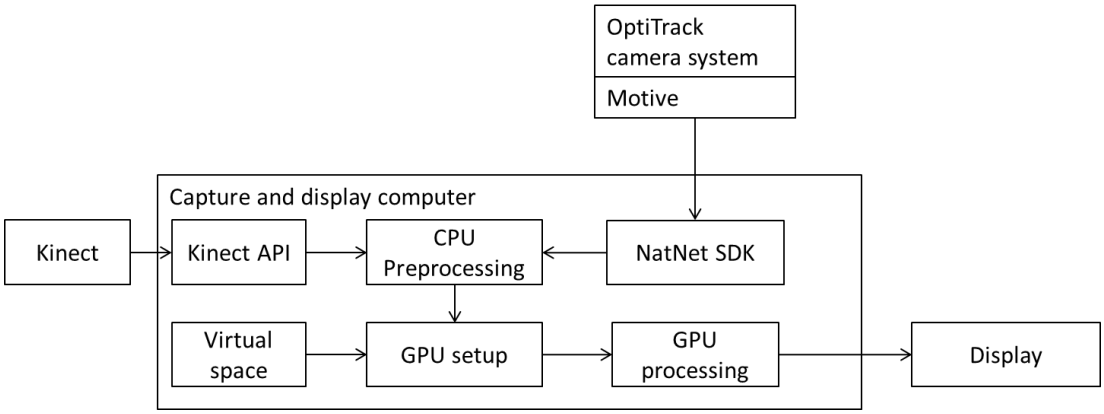

Figure 21. *Computational Block Diagram*

Kinect SDK

The Kinect SDK [48] allows the user to interface the Kinect v2 hardware. It drives the capture and returns the pointer to whichever image the user requires. This can be color, depth or infrared. The API also performs the mapping from the depth camera to the color camera. In Figure 22, the flowchart of the processing done in the SDK is shown.

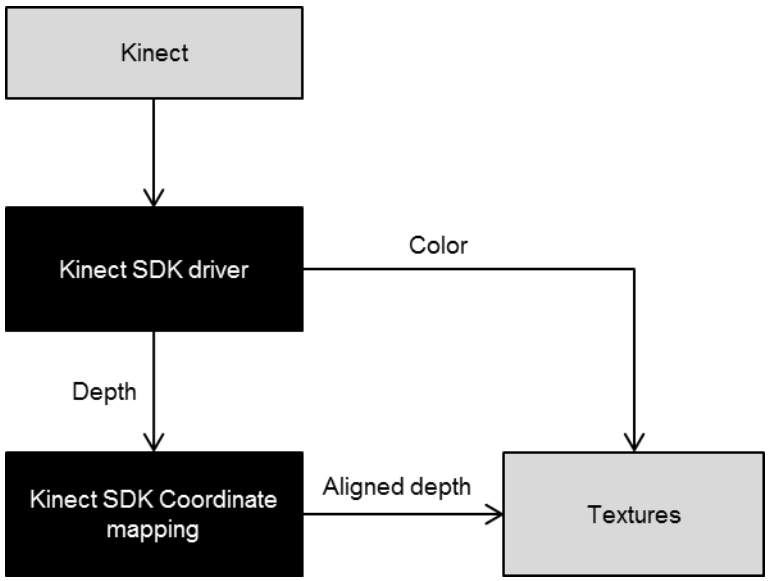

Figure 22. *Kinect process diagram*

NatNet SDK

Natural point provides an SDK that interfaces their software (Motive) and the user's application. The SDK sends the data packets through a previously setup wireless connection and the synchronization of the data transfer. The data is sent at a constant framerate set in motive (12fps, 30fps, and 60fps). The box diagram is shown in Figure 23.

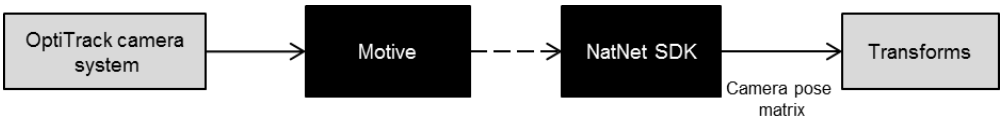

Figure 23. *Motion capture process diagram*

CPU Preprocessing

The preprocessing encompasses early processing of both the real scene and the virtual scene. Some of this is done once (setup); some of it is done every frame (draw).

Setup

The point cloud structure is built before starting the capture process. It is never modified in the CPU once it has been built. The virtual scene is preprocessed by applying any changes to the transforms (placing the objects in their initial position). The real scene is created as an empty template; a point cloud is built where all points have the same depth and are distributed in a rectangle. This is done to diagnose the System startup. If the Kinect is not connected or not delivering the images, the user will see a gray rectangle in front of him.

GPU input initialization and update

The flowchart in Figure 24 shows the steps to prepare the scene before rendering, this are all performed in CPU every frame. This process calls upon the Kinect SDK to deliver the color and depth and the motion capture to deliver the pose of the camera. If either of them has not produced a new frame of information, the previous one is used. The depth and color textures are loaded into the point cloud and the transformation is loaded into the virtual world.

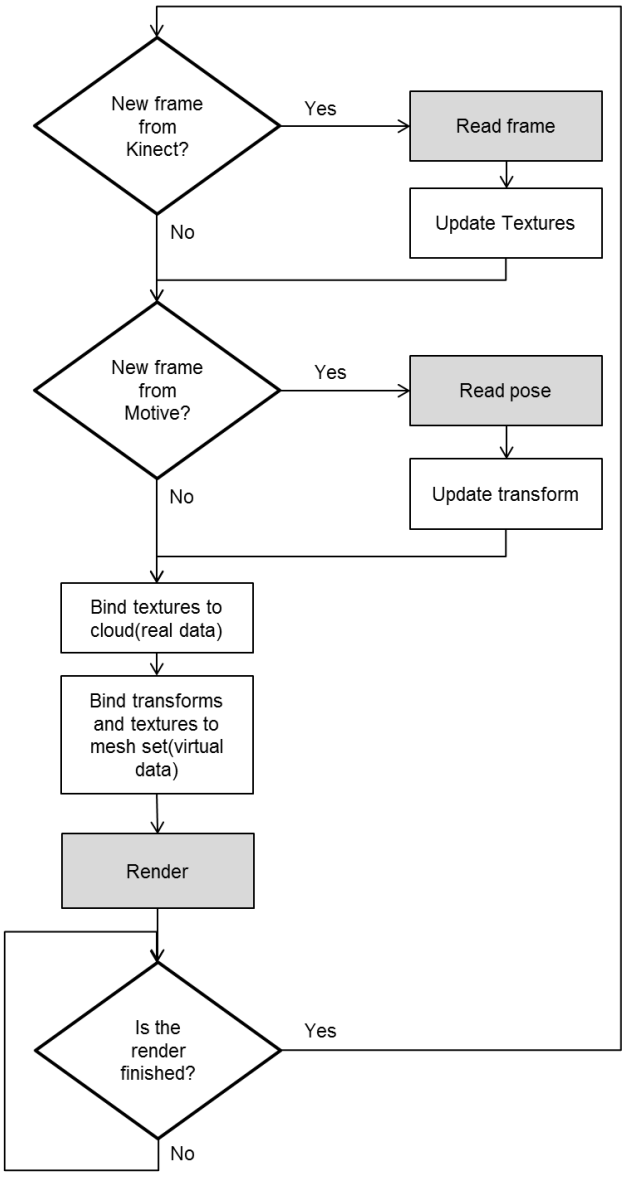

Figure 24. *Flowchart of preprocessing*

GPU processing

The processing in the GPU is programed in shaders (see Section 2.6). There are two sets of shaders, one for the real point cloud and one for the virtual objects. The point primitive is used to render the real point cloud. The triangle primitive is used to render the virtual objects, Figures 25 and 26 show the processing done by the GPU divided into the shaders.

It is important to note that the inner workings of the rasterizer are not included in the block diagrams, and that the blue dashed arrows mean transfer of information. This information is interpolated and sampled by the rasterizer.

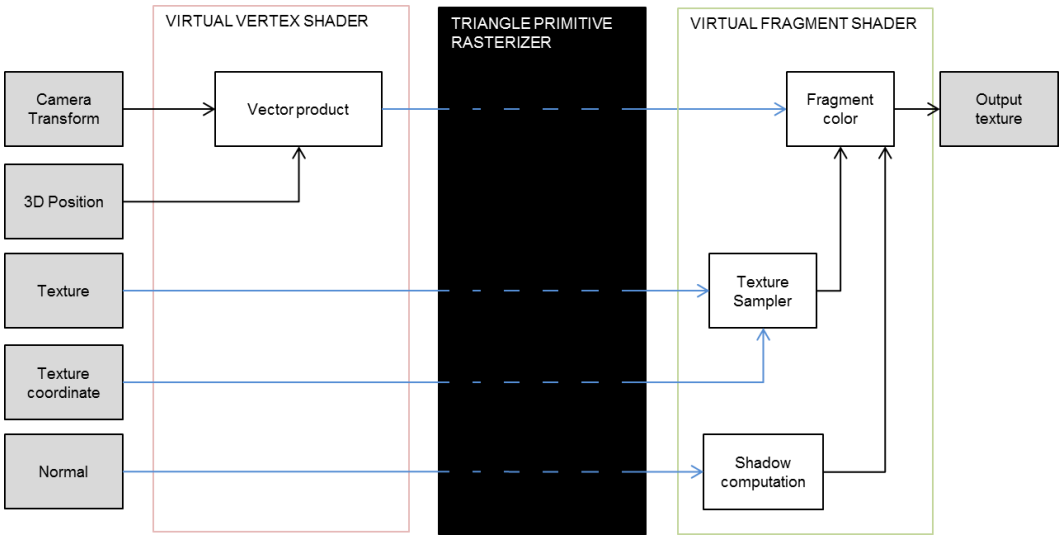

Figure 25. *Shader block diagram for virtual objects. The blue lines indicate information transfer while the rasterizer interpolates and samples the corresponding values for each fragment.*

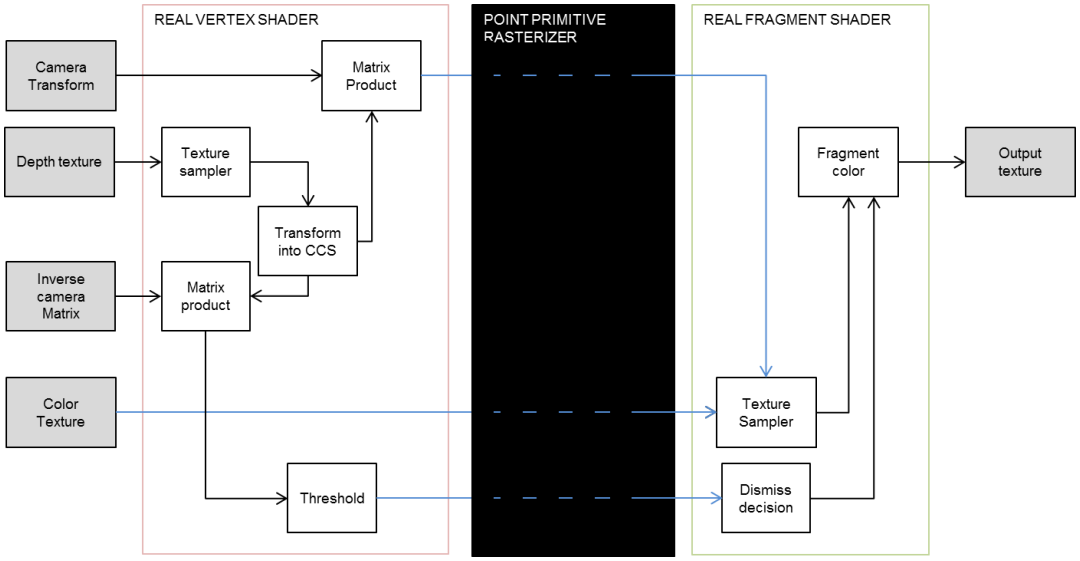

Figure 26. *Shader block diagram for real point cloud. The blue lines indicate information transfer while the rasterizer interpolates and samples the corresponding values for each fragment.*

4. RESULTS

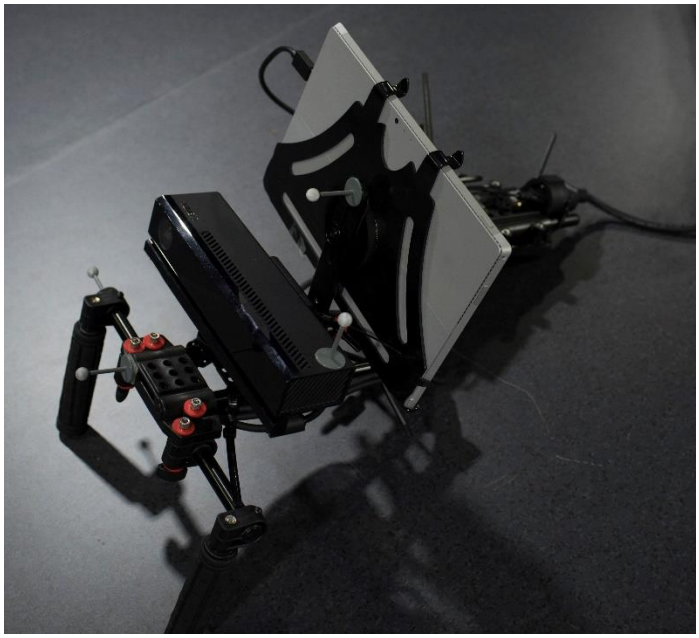

Figure 27. *Virtual camera*

The main result of the work is the actual camera seen in Figure 27. The prototype is available at CIVIT Motion Capture Studio. The components are a Kinect v2 for capture, a Microsoft Surface Pro3 for display and processing, four IR markers and a shoulder camera mount that holds the system together and allows the operator a natural operation of such a device.

The system is evaluated based on the following parameters: refresh rate, delay, and visual quality. First, the external calibration was designed specifically for this system; hence, the results of the calibration for the current prototype are shown in this chapter. Second, the offline experiments designed to measure the performance of the system are presented. These include the current system and the bilateral filter which is not currently in the real time system. Finally the real time performance is presented, this includes time parameters and example photos to evaluate the visual quality.

4.1 External Calibration

The external calibration was developed specifically for this particular system and the detailed results are discussed below. Figure 28 contains the setup used for calibration. Figure 29 shows the markers as seen by the mocap. The two sets are easily differentiated since the camera has four markers and the calibration constellation has six.

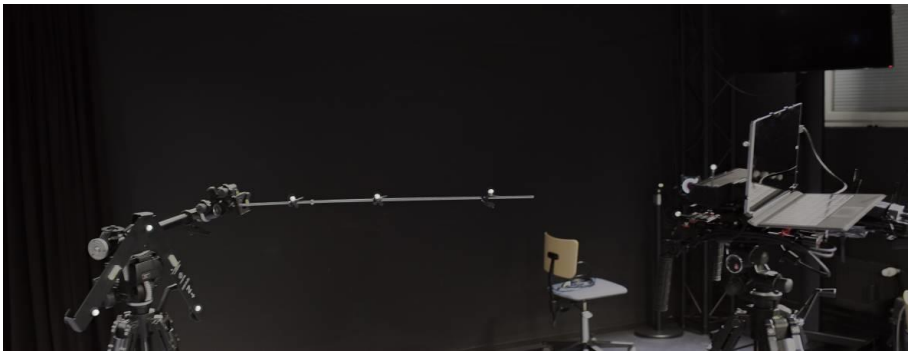

Figure 28. *Setup for joint calibration.*

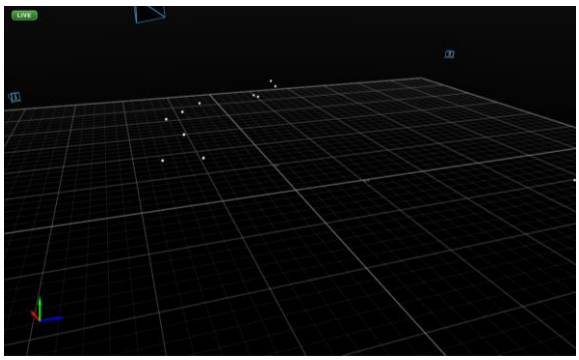

Figure 29. *Markers for joint calibration as seen by the mocap system.*

With this data, the external calibration is performed (see Section 3.1). The data is shown in Table 4 and the resulting model is shown in Figure 30. The origin of the constellation is now a good estimate of the optical center of the camera.

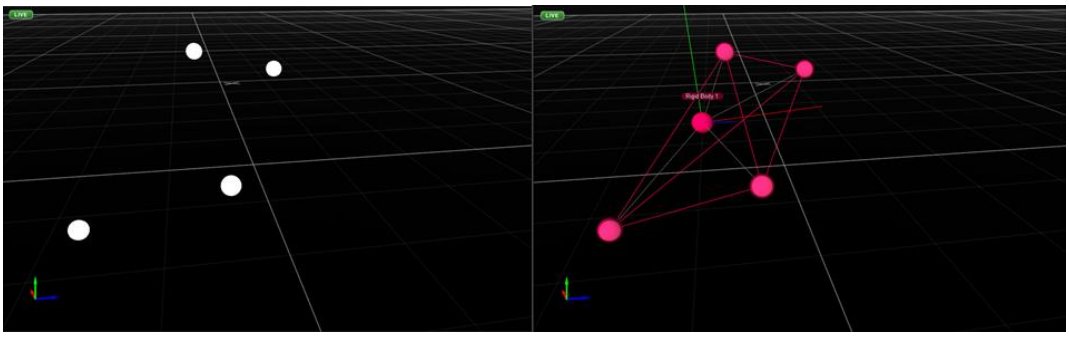

Figure 30. *Left, set of markers that define the position of the virtual camera. Right, set of markers with virtual origin. Ideally this origin corresponds to the optical center of the color camera.*

Table 4. *Joint calibration data, data is presented as a column of row vectors of either length three or length one.*

	Coordinates (mm)		
	x	**y**	**z**
Calibration Constellation (Motion capture)	-562.1	884.5	136.1
	-276.6	879.5	-199.9
	-123.5	876.3	-104.8
	-426.2	829.3	-543.5
	-795.5	940.7	-438.8
	-785.6	857.6	-315.5
Camera Constellation (Motion capture)	666.1	1441.2	-131.1
	645.7	1389.4	-024.9
	646.7	1243.7	-355.6
	549.8	1228.9	-191.6
Optical center of depth camera (Calibration result)	624.7	1434.2	-197.0
New camera constellation (Result)	41.3	6.9	65.9
	21.0	-44.8	172.1
	22.0	-190.5	-158.5
	-74.9	-205.3	5.4
	Distance (mm)		
Calibration constellation distances to camera (ToF sensor)	1.2854		
	1.0100		
	0.8657		
	1.1902		
	1.4735		
	1.4833		

4.2 Offline experiments

Due to performance constrains, some of the experiments were performed offline. Each experiment below is performed offline, the data acquisition and the processing performed are describes in this section.

Joint Bilateral filter

The online system has no joint bilateral filtering, and the distortions are included in the result. This causes some stray pixels appearing and disappearing near the edges of the objects. The joint bilateral filter deals with these distortions without introducing frame delays and aligns the edges to the color stream.

A test program is written in the Kinect SDK that simultaneously captures the depth and color and exports them into images as well as the transformation matrix provided by the mocap system. The depth is encoded into a bmp file as shown in Table 5.

Table 5. *RGB encoding 16 bit depth from Kinect*

R	**G**	**B**
Non valid pixel flag	Least significant 8 bits	Most significant 8 bits

The data is imported into MatLab, where the bilateral filter proposed in [49] is implemented. The filter is performed for several kernel sizes. For each size, the time and resulting depth are saved. These values are used to create several renderings of the same joint reality scene. Evaluating the performance is not straight forward, since most metrics do not translate into subjective “goodness”.

Since the main issue with the camera is the pixels outside of an object that appear in the same plane, the percentage of wrong pixels reduced from the original depth image is taken as the performance metric. A pixel is wrong if it is farther than 10 cm from its actual location. Figure 31 shows several joint spaces in which the input was filtered with different kernel sizes while Figure 32 shows the goodness metric computed.

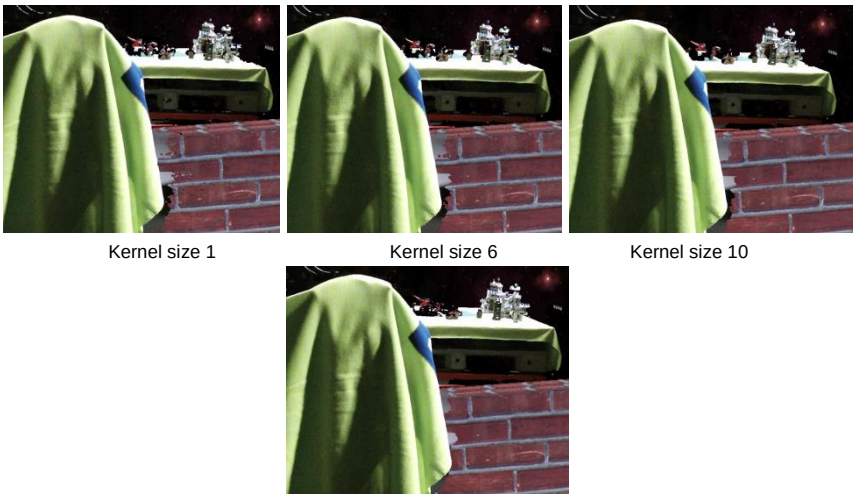

Figure 31. *Detail of result of bilateral filter. See how the edge starts to match the color edge.*

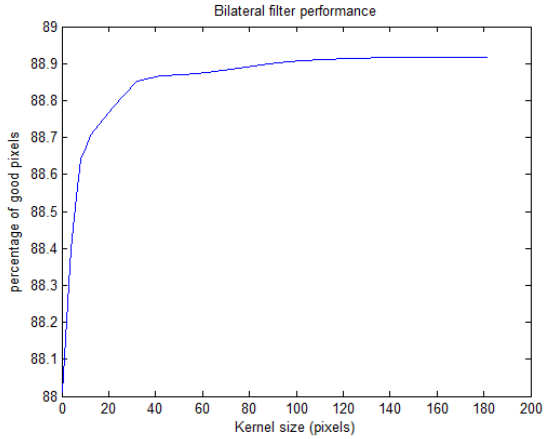

Figure 32. *Bilateral filter performance in the real scene.*

Since the test is performed in a desktop computer with more resources than the tablet, the time displayed in Figure 33 is shown in terms of factor of the first filter (kernel size 1). This shows how the time increases as the kernel size increases.

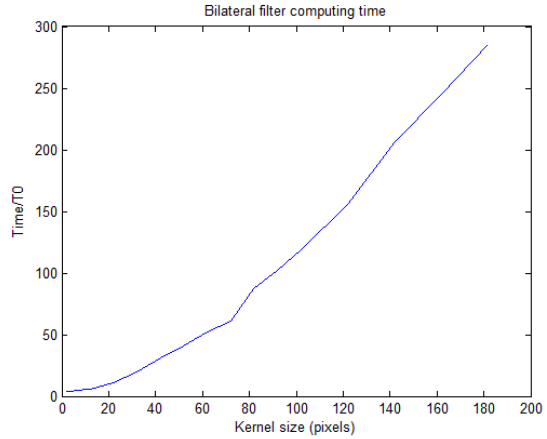

Figure 33. *Computing time vs kernel size for bilateral filter*

It has to be noted that the noise is rapidly changing and quite localized to the edges. This makes it distracting and damages the visual appearance. The filtered image is much smoother and subjectively more naturally looking.

Delay calculation.

Since the system is a union of different systems, calculating the entire delay is not straight forward. The experiment designed to measure the delay requires a separate camera. The camera used is a Basler acA1920 running at SD resolution at 60 fps. The camera is capable of higher resolution and faster capture. But, given the refresh rate of the display and the computing time necessary to perform the tracking over the entire video, it was configured to the aforementioned settings.

The camera is framed so it captures both the space and the render in the screen of the camera. A periodic motion is created in front of the camera with a highly contrasted round object. A white circle was hung from the ceiling and its pendulum oscillation against a black background was recorded in the high-speed camera. Figure 34 contains a frame of the video.

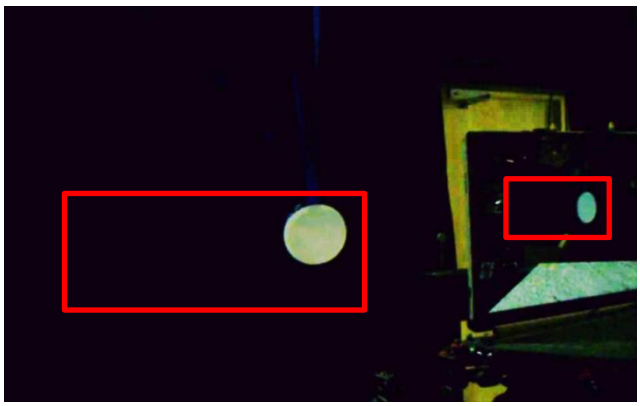

Figure 34. *Frame of delay calculating vide, sub-images are shown as red rectangles.*

The area of movement from both circles is estimated by inspection and for each frame; they are extracted and analyzed separately. The first step is to binarize both sub-images, the threshold is calculated according to Otsu's method [4]. Then, since there are no other white objects in the background of the sub-images, simple gravity center of the intensity values is used to estimate the center of each circle (see Section 2.4).

Since both cameras are aligned to be parallel to the ground, the horizontal coordinates of each circle are strongly correlated. The vertical coordinates have a much smaller range and double the frequency (see Figure 35) so they are not used for the delay estimation.

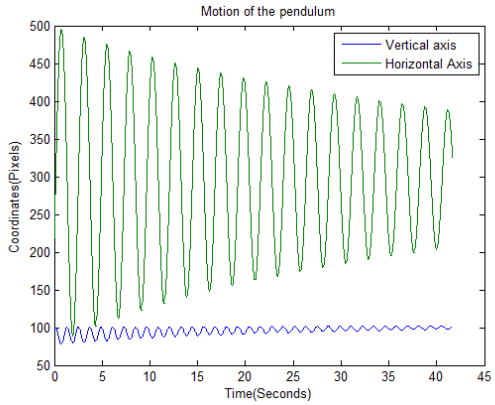

Figure 35. *Motion of the center of the pendulum*

The pixel position is irrelevant, what is important is the relative position in the pendulum path. For this, the horizontal position is scaled and translated so its highest value is one and its center is zero. This is done for both circles, see Figure 36.

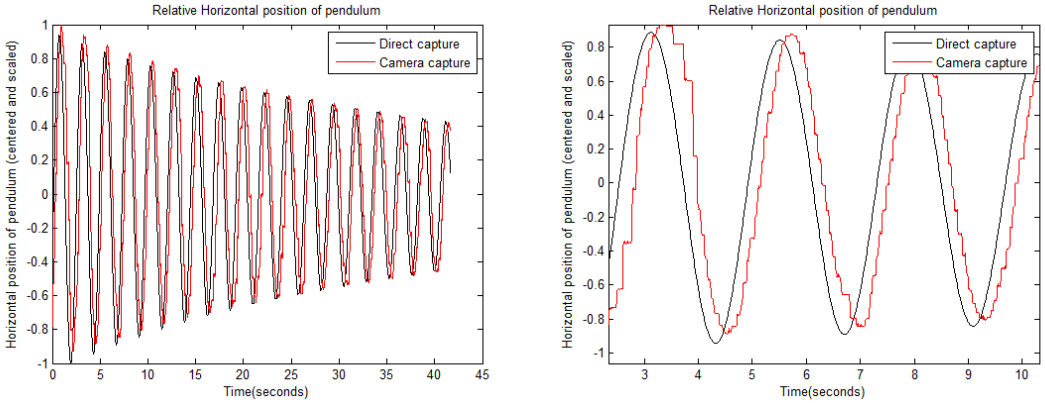

Figure 36. *Relative horizontal motion. Left, 40 seconds of motion. Right, closer look.*

The motion of the center in the camera capture has a staircase shape, this is due to the refresh rate of the camera system (15 fps) being smaller than the capture rate (60 fps). The staircase shape seen in the camera capture in Figure 36 is due to the sampling rate of the external camera (60 fps) being larger than the refresh rate of the system (15 fps).

This is similar to a typical digital to analog converter where quantization error at the outputs is removed using a low pass filter. However, due to the nature of the problem, this is not done. Filtering the signal would average the delay along the time axis and reduce the delay variance. Since the autocorrelation estimates the delay along the whole signal, it estimates the mean delay. In this case this mean is invariant to a time domain filter. Filtering the input will have no change and may introduce unwanted artifacts..

Given the two signals, the delay is estimated by using the cross-correlation signal, shown in Figure 37

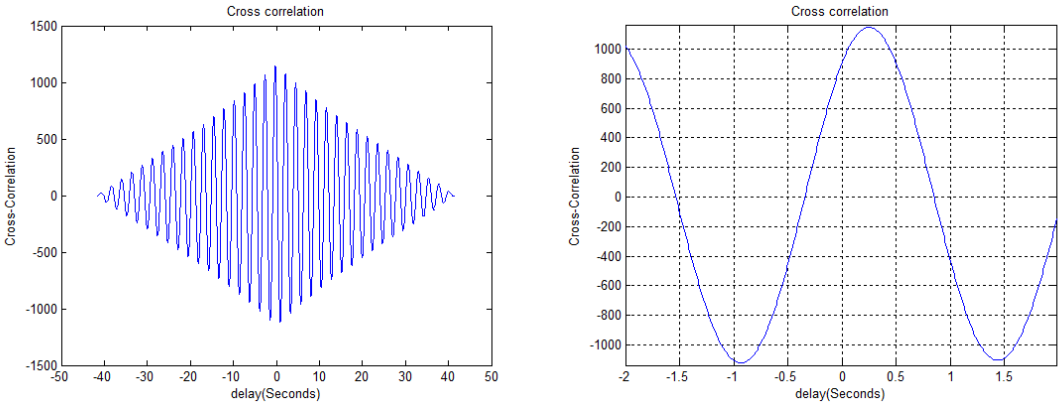

Figure 37. *Cross-correlation of the signals*

The highest peak of the cross correlation is at a delay of 250 ms, this is the estimated delay of the system.

4.3 Performance

The basic performance metrics are shown in Table 6.

Table 6. *System performance metrics.*

Refresh rate	15 fps
Delay	250 ms
Point cloud size	960x540
Texture resolution (internal)	1920x1080*

* Internal resolution of the color texture, output resolution depends on display and is very susceptible to noise.

The markers accuracy is given by the motion capture system. The LS algorithm returns the accuracy of the center estimate. For the orientation error, all possible orientations contained in the markers with the given variance are calculated and the worst case scenario is used as the error. These values are shown in Table 7.

Figure 38 shows the capture space, the IR cameras are attached to rafters in the ceiling and along the walls. The markers are placed in the camera in such a way that only one solution to the model fitting problem exists.

Table 7. *accuracy of motion capture*

Marker error	±0.9 mm
Position error	±0.5mm
Orientation error	±0.4°

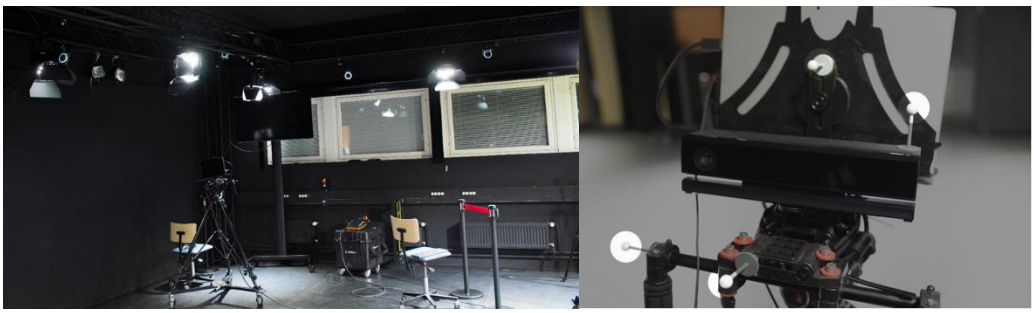

Figure 38. *Left, Motion capture studio, the blue rings are the IR cameras. Right, IR markers highlighted in virtual camera.*

Figure 39 shows the marker constellation as seen by the mocap software. For four markers, a proper constellation is not coplanar and does not contain isosceles triangles.

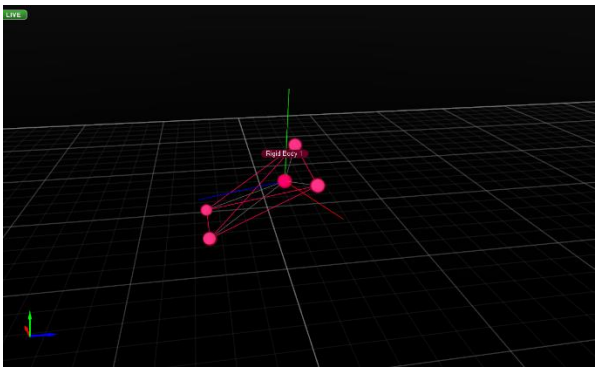

Figure 39. *Marker disposition in camera (shown by MOCAP).*

Figure 40 shows two angles of the same scene to show how the system is coherent in its mix of real and virtual space. The relative positions of objects are not lost when moving the camera around. Of course, this movement is continuous and the camera responds in an interactive feeling.

Figure 40. *Sample of position and orientation consistency.*

5. CONCLUSIONS AND FUTURE WORK

In this thesis, a complete system for previsualization of mixed real and synthetic content has been presented and documented. The system is a functioning prototype, a laboratory proof of concept that showcases the potential of data fusion and multimodal sensing in a movie production context.

The OptiTrack camera system and the Motive software are used to perform optical motion tracking. A total of 16 cameras are installed in the studio, and a four marker constellation is attached to the camera. The resulting matrix is transmitted via Wi-Fi to the camera. Every 33 milliseconds this matrix is updated.

In order to add the real data, a Kinect V2 camera is used. The depth and color feeds are aligned by the Kinect API and are provided to the system every 66 milliseconds. The system sends the frames as textures into the GPU and lets it do the rest of work.

In the GPU, the worlds are mixed and projected to the corrected position of the camera. The processing is done in such a way that minimizes the time spent between capture and display. The result is a mixed reality render of virtual and real worlds which updates fast enough to provide an "interactive" experience.

This system is still in an early development state. It can be labelled as a laboratory working prototype, but needs further work. In order to accomplish the desired state, some key milestones need to be achieved. These are listed in order from high to low importance in order to advance the project.

Depth edge filtering. The depth edge has errors. These errors cause an aesthetic issue where noise is present at the edge of a person in the mix. The challenge is to implement an edge aware filter that aligns the depth edges with the color edges. In this work, a joint bilateral filter was implemented. However, this implementation proved to be slow and was unsuitable for the system with its current processing power. Having the processing done in a local machine reduces the delay and helps the immersion. Having the processing done in a separate unit with more processing power enables more complex algorithms. However, the communication between the machines increases the delay.

Selective focus. Another tool that filmmakers use all the time is a selective focus. Since the world is already stored in 3D, the refocusing should be straight forward. It is important to find a fast implementation and integrate it with the rest of the processing.

Creative tool interface. In order to allow the prospective end users to use the system, it has to be interfaced with a tool that they are familiar with. To this end, there are two main kinds of systems: a game engine that allows game level modelers the freedom to implement any virtual scene and make complex interactions, or movie compositing software that enables filmmakers to blend the real capture data with their premade virtual world. The potential of utilizing Unreal engine as the visualization engine was explored in the early stages of the project, however it was found very demanding and the additional programming effort required to integrate all of the necessary components into it were ultimately deemed to be outside the scope of a thesis on signal processing.

Camera interface controls. In order to create the correct interface that gives the camera operator the full experience, controls are necessary. The basic controls are a focus barrel, a zoom controller and some lighting controls. Similar controls do exist in the Natural Points virtual camera, albeit the lack of 3D scene capture, and have been used in several movies.

6. BIBLIOGRAPHY

[1] Richard Hartley and Andrew Zisserman. 2003. *Multiple View Geometry in Computer Vision* (2 ed.). Cambridge University Press, New York, NY, USA.

[2] P. B. DAVENPORT. Rotations about nonorthogonal axes. AIAA Journal, Vol. 11, No. 6 (1973), pp. 853-857.

[3] Richard Szeliski. 2010. Computer Vision: Algorithms and Applications (1st ed.). Springer-Verlag New York, Inc., New York, NY, USA.

[4] Rodrigues, Olinde (1816), De l'attraction des sphéroïdes, Correspondence sur l'École Impériale Polytechnique, (Thesis for the Faculty of Science of the University of Paris) 3 (3): 361–385.

[5] Rotations, quaternions and double groups. By Simon L. Altmann, Clarendon Press, Oxford, 1986, 317 pp.

[7] Long Quan; Zhongdan Lan, Linear N-point camera pose determination, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.21, no.8, Aug 1999 pp.774-780.

[8] David Marr. 1982. Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. Henry Holt, New York, NY, USA.

[9] Sabry F. El-Hakim ; J. A. Beraldin ; Francois Blais; Comparative evaluation of the performance of passive and active 3D vision systems. Proc. SPIE 2646, Digital Photogrammetry and Remote Sensing '95, 14 (December 1, 1995).

[10] Scharstein, D.; Szeliski, R., High-accuracy stereo depth maps using structured light, in Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol.1, no., pp.I-195-I-202, June 2003.

[11] Gokturk, S.B.; Yalcin, H.; Bamji, C., A Time-Of-Flight Depth Sensor - System Description, Issues and Solutions, in Conference on Computer Vision and Pattern Recognition Workshop.pp.35-35, 27-02 June 2004.

[12] Kinect for xbox one, Microsoft Corporation. <Url: http://www.microsoft.com/en-us/kinectforwindows/purchase/ >

[13] Brown, M.Z.; Burschka, D.; Hager, G.D., Advances in computational stereo, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.25, no.8, pp.993-1008, Aug. 2003.

[14] Seok Lee; Seungsin Lee; Hochen Wey; Jaejoon Lee; Dusik Park, Depth resampling for mixed resolution multiview 3D videos, in IEEE Consumer Communications and Networking Conference, pp.827-832, 11-14 Jan 2013.

[15] Gonzalez, R. C., Woods, R. E., and Eddins, S. L. [2009]. Digital Image Processing Using MATLAB, 2nd ed. Gatesmark Publishing, Knoxville, TN.

[16] Lin Yang; Longyu Zhang; Haiwei Dong; Alelaiwi, A.; El Saddik, A., Evaluating and Improving the Depth Accuracy of Kinect for Windows v2, in IEEE Sensors Journal, vol.15, no.8, pp.4275-4285, Aug. 2015.

[17] Tomasi, C.; Manduchi, R., Bilateral filtering for gray and color images, in Sixth International Conference on Computer Vision, pp.839-846, 4-7 Jan 1998.

[18] He, Kaiming; Sun, Jian; Tang, Xiaoou, Guided Image Filtering, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.35, no.6, pp.1397-1409, June 2013

[19] Jiyoung Jung; Joon-Young Lee; Yekeun Jeong; Kweon, I.S., Time-of-Flight Sensor Calibration for a Color and Depth Camera Pair, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.37, no.7, pp.1501-1513, July 2015.

[20] Chartrand, R.; Wotao Yin, Iteratively reweighted algorithms for compressive sensing, in IEEE International Conference on Acoustics, Speech and Signal Processing. pp.3869-3872, March 31 2008-April 4 2008.

[21] B. Schwald and P. Figueiredo, Learning of Rigid Point-Based Marker Models for Tracking with Stereo Camera Systems, Workshop der GI VR/AR (Chemnitz), pp. 23–34. 2004.

[22] Besl, P.J.; McKay, Neil D., A method for registration of 3-D shapes, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.14, no.2, pp.239-256, Feb 1992.

[23] Umeyama, S., Least-squares estimation of transformation parameters between two point patterns, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.13, no.4, pp.376-380, Apr 1991.

[24] Trefethen, Lloyd N.; Bau III, David (1997). Numerical linear algebra. Philadelphia: Society for Industrial and Applied Mathematics. ISBN 978-0-89871-361-9.

[25] Bancroft, S. An Algebraic Solution of the GPS Equations, in IEEE Transactions on Aerospace and Electronic Systems, vol.AES-21, no.1, pp.56-59, Jan. 1985.

[26] Motive, Natural point OptiTrack camera system <Url:www.optitrack.com>.

[27] Collins, R.T., Mean-shift blob tracking through scale space, in Proceedings, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol.2, no., pp.II-234-40 vol.2, 18-20 June 2003.

[28] Brandner, M.; Ribo, M.; Pinz, A., State of the art of vision-based self-localization, in 1st International Workshop on Robotic Sensing, pp.6 pp.-, 5-6 June 2003.

[29] Hedborg, J.; Robinson, A.; Felsberg, M., Robust Three-View Triangulation Done Fast, in 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp.152-157, 23-28 June 2014.

[30] Xunnian Yang, Surface interpolation of meshes with shape optimization, in Proceedings, Geometric Modeling and Processing, pp.103-112, 2004

[31] Attene, M.; Falcidieno, B.; Rossignac, Jarek; Spagnuolo, M., Sharpen&Bend: recovering curved sharp edges in triangle meshes produced by feature-insensitive sampling, in IEEE Transactions on Visualization and Computer Graphics, vol.11, no.2, pp.181-192, March-April 2005.

[32] M. Mehling: Implementation of a Low Cost Marker Based Infrared Light Optical Tracking System; Supervisor: H. Kaufmann; Institute for Software Technology & Interactive Systems, 2006.

[33] Richard S. Wright, Nicholas Haemel, Graham Sellers, and Benjamin Lipchak. 2010. OpenGL Superbible: Comprehensive Tutorial and Reference (5th ed.). Addison-Wesley Professional.

[34] Prvulovic, M.; Durdevic, D.; Tartalja, I., SeeGL: Software tool for learning the OpenGL graphics library, in Telecommunications Forum pp.1604-1608, 20-22 Nov. 2012.

[35] Schiller, I.; Bartczak, B.; Kellner, F.; Kollmann, J.; Koch, R., Increasing Realism and Supporting Content Planning for Dynamic Scenes in a Mixed Reality System incorporating a Time-of-Flight Camera, in 5th European Conference on Visual Media Production, pp.1-10, 26-27 Nov. 2008.

[36] P. Milgram and A. F. Kishino (1994). Taxonomy of Mixed Reality Visual Displays. IEICE Transactions on Information and Systems. pp. 1321–1329.

[37] Kato, H., Billinghurst, M. (1999) Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99). October, San Francisco, USA.

[38] Davison, A.J.; Reid, I.D.; Molton, N.D.; Stasse, O., MonoSLAM: Real-Time Single Camera SLAM, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.29, no.6, pp.1052-1067, June 2007.

[39] Kanbara, M.; Okuma, T.; Takemura, H.; Yokoya, N., A stereoscopic video see-through augmented reality system based on real-time vision-based registration, in proceedings for IEEE Virtual Reality conference, pp.255-262, 2000.

[40] Schiller, I.; Bartczak, B.; Kellner, F.; Kollmann, J.; Koch, R., Increasing Realism and Supporting Content Planning for Dynamic Scenes in a Mixed Reality System incorporating a Time-of-Flight Camera, in 5th European Conference on Visual Media Production, pp.1-10, 26-27 Nov. 2008.

[41] Kurillo, G.; Bajcsy, R.; Nahrsted, K.; Kreylos, O., Immersive 3D Environment for Remote Collaboration and Training of Physical Activities, in IEEE Virtual Reality Conference, pp.269-270, 8-12 March 2008.

[42] Microsoft Hololens, URL: http://www.microsoft.com/microsoft-hololens/en-us

[43] W. Bradford Paley, A. 1998. Interaction in 3D Graphics. In Proceedings of SIGGRAPH 98, Annual Conference Series, 43--54.

[44] Lecuyer, A.; Burkhardt, J.-M.; Henaff, J.-M.; Donikian, S., Camera Motions Improve the Sensation of Walking in Virtual Environments, in Virtual Reality Conference, pp.11-18, 25-29 March 2006.

[45] Hansung Kim; Sakamoto, R.; Kitahara, I.; Toriyama, T.; Kogure, K., Vitual Camera Control System for Cinematographic 3D Video Rendering, in 3DTV Conference, pp.1-4, 7-9 May 2007

[46] B.-J. Lee, J.-S. Park, and M. Sung. Vision-based real-time camera match moving with a known marker. In Entertainment Computing - ICEC 2006. 2006.

[47] Raposo, C.; Barreto, J.P.; Nunes, U., Fast and Accurate Calibration of a Kinect Sensor, in 2013 International Conference on 3D Vision, pp.342-349, June 29 2013-July 1 2013.

[48] Microsoft Kinect sdk v 2.0. URL: https://www.microsoft.com/en-us/kinectforwindows/develop/

[49] Q. Yang, A non-local cost aggregation method for stereo matching, in CVPR, 2012, pp. 1402-1409.