Virtual Teleportation (VT) in Real-Time

Conception and Realisation with Low-Cost Hardware

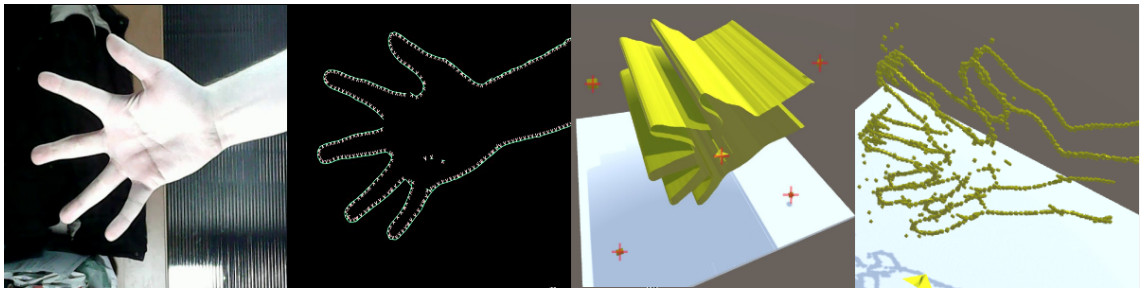

Abbildung 1: Four steps of virtual teleportation

Bachelor Thesis

Presented to

the **"Hochschule der Medien Stuttgart"**

In Fulfillment of the Requirements for the Degree

Bachelor of Science

by

Alexander Georgescu

Matriculation number: 28272

January 2017

Supervisors:

Prof. Walter Kriha

Andreas Stiegler

Period: October the 24th, 2016 until January the 23th, 2017

Declaration of Authenticity

I declare that all material presented in this paper is my own work or fully and specifically acknowledged wherever adapted from other sources. I understand that if at any time it is shown that we have significantly misrepresented material presented here, any degree or credits awarded to us on the basis of that material may be revoked. I declare that all statements and information contained herein are true, correct and accurate to the best of my knowledge and belief.[1]

Ort, Datum

Unterschrift

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.[2]

Ort, Datum

Unterschrift

[1] Koc09, S. 2.
[2] Rie02, S. 42.

Abstract

This paper presents and explains multiple concepts and possibilities of implementing software to realize virtual teleportation in real-time with low-cost consumer equipment. The relation and relevance of this technology for the market of virtual reality are investigated.

The variants are elaborated and argued to chose one best fitting the defined objective based on the pricing-strategy matrix to aim for maximal market penetration and reach the mass market. Consequently an evaluated solution based on two-dimensional (2D) camera data is described accurately within the main part of this paper and enables an implementation in the same way.

This writing accompanies a reference-implementation of the described principle and will repeatedly refer to its source code. However, for the purpose of illustration and to avoid restriction to a predefined programming language, only pseudo code will be used in the document itself.

Eventually the resulting software is re-evaluated to compare with the defined objectives and goals of virtual teleportation and virtual reality. The result is that a primary introduction to consumers is feasible in near future after an increase of accuracy of the graphical outcome.

Keywords: virtual teleportation, virtual reality, VR, software, informatics, algorithms, solution, developement

Kurzfassung

Diese Arbeit präsentiert und erklärt mehrere Konzepte und Möglichkeiten zur Implementierung von Sofware welche virtuelle Teleportation unter Nutzung von low-cost consumer equipment realisiert. Der Bezug und die Relevanz dieser Technik für den Bereich der Virtuellen Realität wird dargelegt.

Die Varianten werden untersucht und bewertet um eine optimale Lösung zu finden, hinsichtlich der Zielsetzung basierend auf der Preisstrategie Matrix mit dem Ziel die Marktdurchsetzung zu maximieren. Darauffolgend wird im Hauptteil der Arbeit, ein, auf zweidimensionalen Kameradaten basierender Lösungsansatz ausführlich beschrieben um eine Implementation in Form einer Software zu ermöglichen.

Zu diesem Dokument existiert eine Referenzimplementation welche dem beschriebenen Prinzip folgt und zur Veranschaulichung verlinkt wird. Zum Zwecke der Veranschaulichung und die Einschränkung auf eine bestimmte Programmiersprache zu vermeiden, wird nur Pseudocode im Dokument selbst verwendet.

Abschließend wird der Lösungsansatz anhand der Referenzimplementation, hinsichtlich der Zielvorgabe sowie im Bezug auf die Relevanz im Bereich von virtual teleportation und virtual reality bewertet. Das Resultat ist, dass bei Erhöhung der Genauigkeit, in naher Zukunft eine erste Verbreitung in Haushalten erwartbar ist.

Stichwörter: virtuelle Teleportation, virtuelle Realität, VR, software, Informatik, Algorithms, Entwicklung

Contents

Chapter 1

The Concept of Virtual Teleportation (VT)

The following sections give an insight into the way virtual reality (VR) and virtual teleportation (VT) are understood within this document and provides a base of knowledge to comprehend the concepts and goals.

1.1 Definition

Traditionally, teleportation describes the idea of matter or its particles being transferred from one position to another without applying forces to have actual movement. As more concrete concept it's the "method of transportation in which matter is converted into minute particles or into energy at one point and re-created in original form at another."[1].

Therefore VT describes the process of achieving the same effect in a virtual environment. That means the influences of the target object on the environment - for example the way it absorbs or reflects light - is being digitized into enough information to allow its recreation in the virtual reality.

1.2 VT as a Component of Virtual Reality (VR)

VR is the general concept of deceiving persons of an alternate reality. It describes "a computer simulation of a real or imaginary system that enables a user to perform operations on the simulated system and shows the effects in real-time."[2]

Therefore VT is a subsection of the broad field of VR and at the creation time of this document it is an aspect which has reached the stage of being feasible through software very recently.

[1] Eng11.
[2] Eng16.

1.2.1 Purpose for the Goal of Immersion

The immersion is a significant factor for the effectiveness of virtual reality as it describes how well the perception of the artificial environment has succeeded on the user. On the other side, a break of immersion can directly harm the experience.[3]

To achieve the goal of maximal immersion it is naturally not sufficing to display only virtual elements as the users environment. The person requires an accurate representation of the physical body as well or of physically existing and manipulable objects.

Virtual teleportation can be a significant part of the methods to reach this goal because it provides the data for the VR system required to represent real objects or users in the virtual environment in real-time. So called "tactical immersion" which requires a direct and intuitive input by the user can be achieved this way.[4]

1.3 Current Solutions

Although at least one company, facebook ([5]), has announced ressearch in this field and one can assume that it is not the only one, currently there does not seem to exist any publicly available solutions for virtual teleportation in real-time.

1.3.1 Microsoft™Holoportation

A project which has achieved significant resonance in public media although not being directly available as a product, is Microsoft™'s Holoportation project. It has achieved an accuracy sufficing to impress many people and a delay-time between real input and the virtual representation which is short enough to be perceived similar to real-time.[6]

However, the equipment consisting of 3D cameras it uses, is not easily available to the average consumer. The exact types are not disclosed publicly but similar cameras with the required capabilities are only used in business environments and for other purposes.

[3] cf. Ada04, Paragraph. 4.
[4] cf. Ada04, Tactical Immersion.
[5] cf. DOn15.
[6] cf. Mic.

Chapter 2

Objectives for Implementation

This chapter arguments the objectives which influenced the choices made when evaluating different variants of implementing the software for VT.

2.1 Pricing Strategies

Following the basic principles of market economy there are two main factors which determine a general classification of strategies for positioning a product on the market:

Price
: The cost for the customer

Quality
: The reward for the customer

The Cost-Quality or Pricing-Strategy-Matrix visualizes this concept.[1]

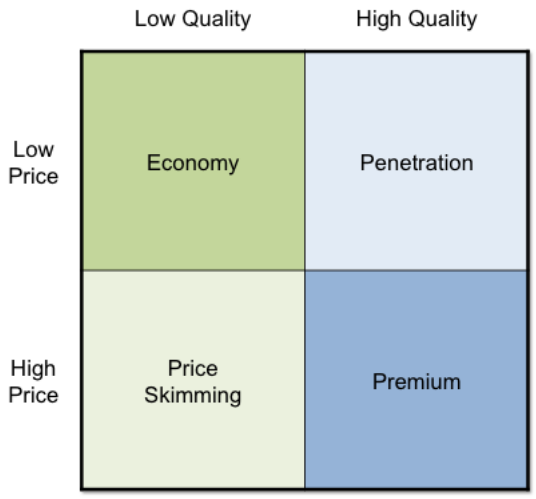

Figure 2.1: [Pricing Strategy Matrix 201]

[1] cf. 201, Summary.

1. **Economy pricing**
   Costs are kept to a minimum and price is set as low as possible because there is no difference in the products in the market.
2. **Penetration pricing**
   A low price is used to capture market share. The price is often raised once market share is gained.
3. **Price skimming**
   When a product is introduced to the market a high price is set that limits the volume of sales but still produces a high return. Often used to recover investment costs by targeting 'early adopters' who are less price sensitive. The price is then reduced as competitors enter the market..
4. **Premium pricing**
   A high price is used to encourage the perception of quality. Usually where there is a dominant brand and competitive advantage.

Quote 1: [About main pricing strategies 201]

2.2 Market for VR and VT

As of the date of this document, the hardware of VR equipment is a significant investment compared to the major part of other consumer oriented computer hardware. On average the cost is comparable to the computer itself and much higher than other input devices. Examples of current products on the VR market are the Oculus Rift, the HTC VIVE and the Microsoft™Holo Lens.

When referring to the general consumer market this results in a classification of their strategy as price skimming or premium pricing. Therefore economy customers are barely targeted and there are chances for market penetration through a low-cost approach. As VT is a component of VR (cf. VT as a Component of Virtual Reality (VR)) this concept can be transposed.

2.3 Requirements for Market Penetration

Certain objectives have to be fulfilled to realize a product on the market with the goal of high market penetration. As described by the pricing-strategies, the relation between quality and cost is determining the customers. A low price is required to target the economy field and a sufficing level of quality is required to achieve market penetration. Due to the natural copyability of software the hardware is the first hurdle to lower the investment cost for the customer.

Chapter 3

Approaches for Realizing VT

The following sections expands the theoretical idea of VT into realizable concepts and evaluates them based on the previously developed objectives.

3.1 Desired Data

The technical challenge of VT is to determine whether a location is part of the surface of an object or not for every possible location in a real three-dimensional (3D)-space. This data is the goal of any technology with the purpose of realizing VT. It forms the representation of the visually perceivable objects and therefore can be displayed in a virtual environment to virtually achieve the effect of teleportation.

The following pseudo code visualizes this in a programmatical form.

```
//Variables
x, y, z; // Coordinates
data[][][]; // Three-dimensional array: Result

// Call to update data
function scan_real_world()
{
   for(every real world point)
   {
      if (point(x,y,z) is on surface of a physical object)
         data[x][y][z] = true;
      else
         data[x][y][z] = false;
   }
}
```

```
// Call to visualize virtually
function display_in_virtual_world()
{
   for(every virtual point)
   {
      if (data[x][y][z] == true)
         draw_virtual_surface(x, y, z);
   }
}
```

Of course a complete concept requires to maintain further information about the visual appearance of the surface at every point.

As for now there are two main, basic principles to capture the information required to compute the desired data described previously.

3.2 Using Depth Data Input

One method requires the input of depth data. That means instead of an image consisting of color- or illumination-values, the camera system perceives the distance between the object and itself. Two cameras placed opposed to each other are the minimum required to gain the data enabling to reproduce the 3D-object in virtual space.

This is the method used by Microsoft™'s realization (cf. Microsoft™Holoportation).

3.2.1 Advantages

Accuracy

The accuracy of this method is depending without a limit on the quality of the cameras. A possibly optimal input will provide an optimal reproduction in the virtual environment.

Computation effort after input

With the depth-data as the prepared input further computations require relatively low effort compared to other methods.

3.2.2 Disadvantages

Cost

3D cameras with a sufficing accuracy are at the time of this document priced in ranges not reasonable for use in consumer products.

Unavailability

The cameras are predestined for business use and not available through the

same ways as 2D cameras are.

Computation effort to prepare input

The 3D cameras require expensive computation of stereoscopy to acquire the sufficing quality.

Additional color camera

An additional color camera is required to perceive the texture of the objects enabling to reproduce it in the virtual environment.

3.3 Using Image Data Input

An alternative method is based on 2D cameras providing the input as images of color values. The reproduction in the virtual environment bases on the information where the edges of the object are on the image of every camera. The following image visualizes the concept:

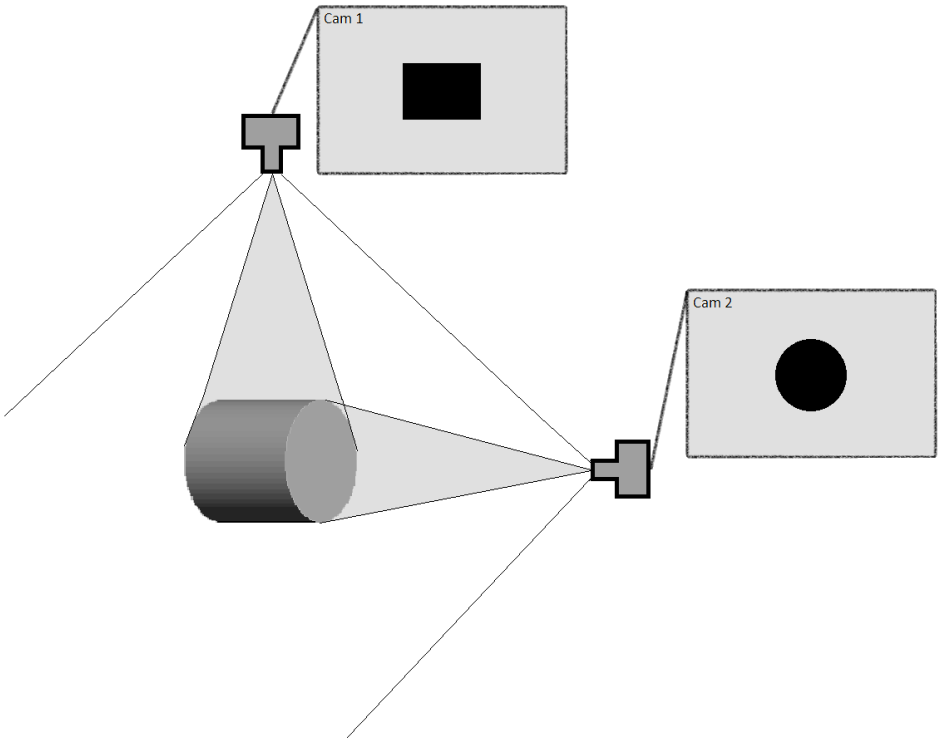

Figure 3.1: Perception of a cylinder by two cameras

3.3.1 Advantages

Cost

2D cameras are starting at a relatively low price of 8€ per piece. When a computer is available, this is the only required investment in hardware to enable the input of VT.

Availability

The cameras can be acquired easily and are constantly available.

3.3.2 Disadvantages

Multiple cameras

To achieve sufficing results more than two cameras are required and fixations are needed for every camera.

Limited accuracy

The input data does not directly provide depth information. The edge information results in limited accuracy on concave objects and in certain orientations independently on the accuracy of every camera in particular. More cameras increase results.

Computation effort exponentially increasing with camera number

The algorithm requires computations for every input image related to all input images. Therefore the effort increases exponentially.

3.4 Choice and Justification

The objectives (Requirements for Market Penetration) focus on minimizing cost whilst maintaining a sufficing degree of accuracy and quality. Therefore the method based on image data appears the more feasible variant to fulfill the goal of market penetration because the required hardware is easily available to customers and keeps the monetary investment low. Additionally it efficiently scalable with the number of cameras providing the input.

Chapter 4

Implementation

The following sections describe the implementation of the previously presented (Using Image Data Input) and justified (Choice and Justification) method. Pseudo code is used to visualize complex components and algorithms.

4.1 Overview of Concept

4.1.1 Input Data

The algorithm bases solely on the input from multiple digital cameras and the information how those cameras are positioned around the targeted areas.

The cameras provide data as a video stream, meaning a constant series of 2D images consisting of the perceived pixel colors. Additional information is provided by the user after installing the cameras and before the algorithm can begin. For every camera, this set of information consists of the following set of information:

Position
: Coordinate of the camera lens in 3D-space

Orientation
: The direction the camera points at

Field of view
: How much the camera can observe (in degrees horizontally and vertically)

Resolution
: How many pixels the camera provides (horizontally and vertically)

4.1.2 Conditions and Recommendations

The method requires a couple of conditions to be fulfilled to achieve the best quality. The reason for those conditions will be explained through the algorithm.

All cameras have to be oriented towards one point in space and the distance to this point should be similar for all cameras. The distance between the cameras should be as high as possible to observe the target area from the most efficient positions. Furthermore, the "field of view" may not differ too significantly.

The computations are performed on static images, therefore the output is updated at the same rate of the input. Therefore a higher input is recommended. twenty frames per second and above provide an acceptable result. In this context it is important to provide sufficient illumination onto the target area because the input rate of some camera models depend on light.

Due to the method of background-removal (cf. Background Removal) the area visible by the cameras should be as static as possible to avoid errors and artifacts due to falsely classified pixels.

4.1.3 Processing

The algorithms works with a set of encompassing steps to lead from the in-out data to an output. The following components are required:

1. Receiving camera input and preparing the frame as an easily accessible set block of data.
2. Removing the background of the image based on a comparison image.
3. Computing the contour of everything visible one very camera.
4. Projecting the cameras into the virtual space as planes oriented the same way the real cameras are oriented.
5. Simulating the contours for every camera as unlimited rays originating from the corresponding coordinates vertically on the planes in 3D-space.
6. Compute the intersections of every ray from every camera with every ray from the other cameras.
7. Classify the intersections into "start-points" and "end-points".
8. Transfer all the areas between "start-points" and "end-points" for every ray into polygon-coordinates.

This process is repeated for every frame while the application is running.

4.1.4 Output

The resulting set of polygons in virtual 3D-space (polygon-coordinates) for every frame are the virtual representation of the real object surface as perceived by the cameras. This data set can be visualized through rendering techniques for suited

output devices like monitors or directly through VR-glasses like the mentioned Oculus Rift, the HTC VIVE or the Microsoft Holo Lens.

4.2 Provided Software Components

To reduce the complexity of practical implementation it is possible to rely on known libraries or other preexisting software which provides data for the previously described core of the application (cf. Processing) or uses its output. This document will provide two widely available, adequately platform-independent suggestions for this purpose.

4.2.1 Camera Input

The software providing the camera input requires to fulfill the following criteria to provide a base for the further implementation:

Versatility
To achieve an optimal experience the system needs to handle a large number of different cameras. Therefore it needs to handle different types of encoding and connection methods.

Parallelism
The input from multiple cameras needs to be provided at the same time.

Low delay
The input frames must be provided as fast as possible.

Easily accessible output
The data structure of the output must be processable by the further implementation as efficiently as possible.

Matching license model
The attached license must be compatible with the planned license of the overall implementation

The "Open Source Computer Vision Library" (OpenCV) fulfills the criteria and is an universal set of algorithms and data structures centered around the acquisition, computation and procession of computer graphics.[1] It provides functionality to work with image data in a low-level way and therefore is ideal for the highly time critical aspect of VT.

Furthermore it is published under a "BSD license" and therefore enables a developer to use it accademically as well as commercially for free.[2]

[1] cf. CVb.
[2] cf. org.

The contained class "VideoCapture" of OpenCV enables to connect with various types of sources for video data. Sources can be physical cameras as input devices but also prerecorded video files.[3] The API prepares direct access to individual, decoded frames in memory and can form the base input as required for the further parts of the application for VT.

Many alternatives are available. Examples are the video component of "Processing", the dedicated libary VideoInput, VideoMan and many others. A direct low-level implementation of the camera access is an option as well. Every possibility has advantages and disadvantages exceeding the range of this document and if possible they should be evaluated before making a choice. The most significant reason for the choice as a main example in this document is mainly preexisting knowledge about OpenCV of the author.

4.2.2 Polygon and VR Renderer

As described, the output (cf. Output) is a set of polygon-coordinates in virtual 3D-space which need to be visualized to finalize the effect of VT. Visualization, here called "rendering" is a complex process which is described in many papers and realized in many preexisting systems. To maximize the possibilities of an implementation of VT it can be helpful to use a publicly available system with high notable presence on the market.

A fixed set of criteria need to be followed:

Compatible data connection with implementation
: Output and input (for example commands and settings) of the VT implementation needs to be transferable.

Rendering engine for polygons
: The system needs to render the polygon-coordinates from the VT system for monitors as well as VR-glasses.

Compatibility with VR-systems
: Support for common systems like the Oculus Rift and the HTC VIVE

Matching license model
: The attached license must be compatible with the planned license of the overall implementation

The game-engine "Unity 3D" is a possible option. It provides all described requirements for rendering and compatibility and supports different VR devices.[4]

[3] cf. CVa.
[4] cf. Uni, VR overview.

An alternative is the "Unreal Engine 4" which has the benefit of being open-source in the programming language C++.

4.3 Camera Processing

Once the data of a frame from the cameras has been provided by the camera software (cf. Camera Input) this image needs to be processed. The following steps are performed independently for every camera in particular.

Figure 4.1: Example image

4.3.1 Background Removal

First step requires to separate the actually desired content on the image from unimportant information. To perform this in an efficient way a comparison technique is used.

Upon start of the application or during an user-triggered calibration process the image is saved in memory. This has to happen when the area supposed to be "teleported" virtually is empty.

Figure 4.2: Background image for the example

To process a new frame every pixel is compared to the saved background image. If the color difference exceeds a preset limit the pixel is classified as "object" and otherwise counts as "background". The result is a binary image with the same resolution as the original frame.

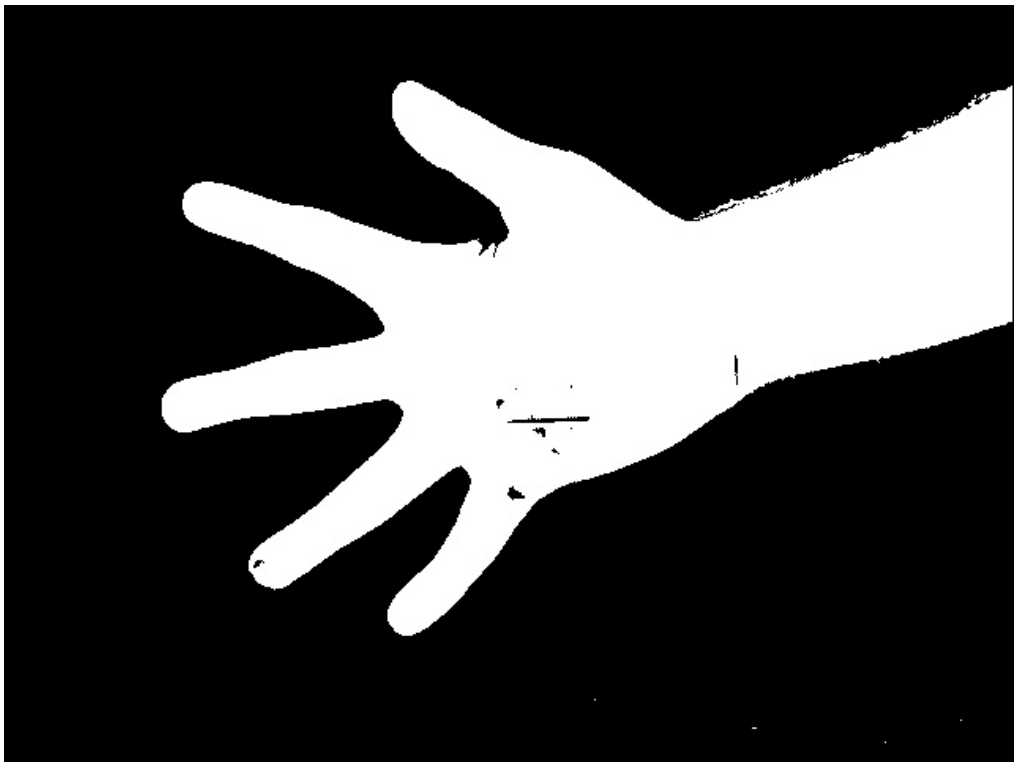

Figure 4.3: Image classified. White: object - Black: background

```
//Variables
background[]; // Array: pixel colors
nonbackground_binary[]; // Array: Non-Background pixels
difference_limit; // Max color difference for background

// Call to save the background
function calibrate_background(camera_input[])
{
   for(every pixel IN camera_input)
   {
      // Save the pixel color int he background array
      background[index] = camera_input[index];
   }
}
// Call to compute the binary result
function compute_background_binary(camera_input[])
{
   for(every pixel IN camera_input)
   {
      // Compute the difference of colors
      dif = (camera_input[index] - background[index]);
      if (dif > difference_limit)
         nonbackground_binary[index] = true;
      else
         nonbackground_binary[index] = false;
   }
}
```

4.3.2 Contour Pixel Computation

The relevant information required for the further processing to achieve the set of 3D polygons of the object surface are the edges between pixels classified as "background" and ones as "object". Therefore the next step involves determining pixels which are part of the edge. Another classification process is performed on the binary image and classifies pixels with a neighbor differing from itself, as "edge" and all other pixels (homogenous areas) as "non-edge". The result is a second binary map.

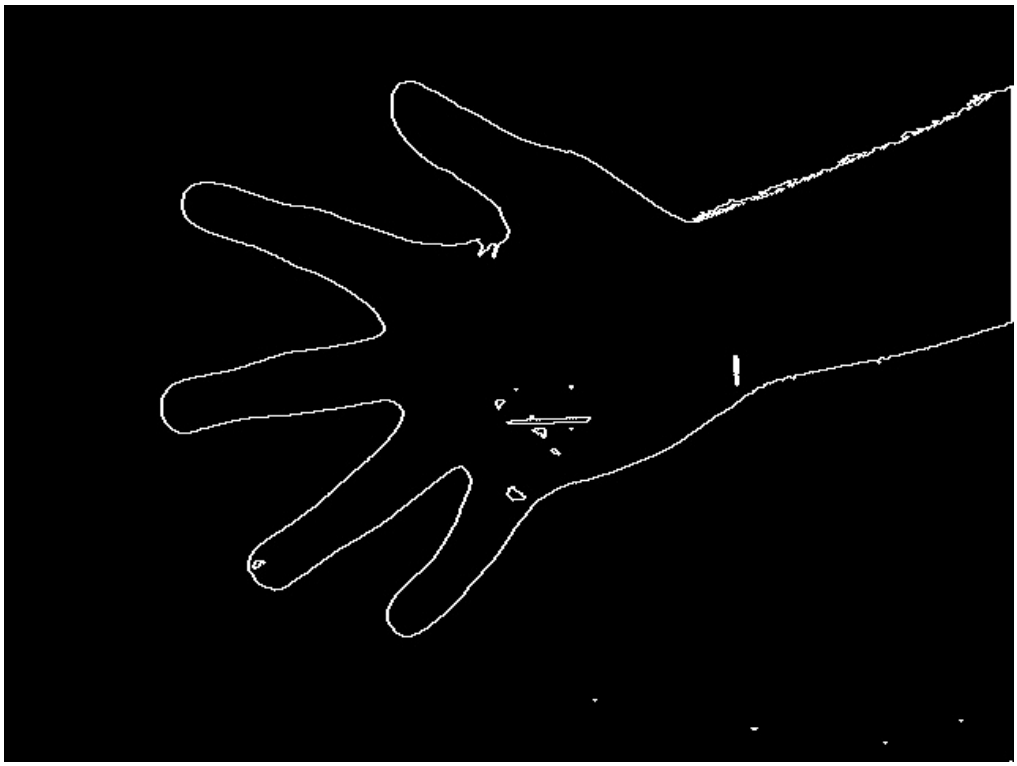

Figure 4.4: Contour image for the example

```
//Variables
contours_binary[]; // Array: Contour pixels (bool)

// Call to compute the contour pixels
function compute_contours(nonbackground_binary[])
{
   for(every boolean IN nonbackground_binary)
   {
      // If the current pixel is not background
      if (nonbackground_binary[ind])
      {
         if (any surrounding pixel is background)
            contours_binary[index] = true;
         else
            contours_binary[index] = false;
      }
   }
}
```

4.3.3 Contour Key-Points Computation

As mentioned in the processing overview (cf. Processing) rays originating from the edges need to be computed in the end. That means the more complex the contour is, the more computation will be required. Therefore the contours need to be simplified To enable the program to work in real-time. For this purpose the contour needs to be reduced to "key-points" instead of using every pixel.

This simplification happens by dividing the binary contour image into a grid of square fields. The size of those fields is predefined beforehand and determines how accurate the resulting set of polygons will be. Larger squares reduce accuracy and also reduce computation effort. The algorithm determines a position-average of all contour-pixels for every square individually. That means calculating the center of gravity. The result are a set of "key-points" approximating the original contour.

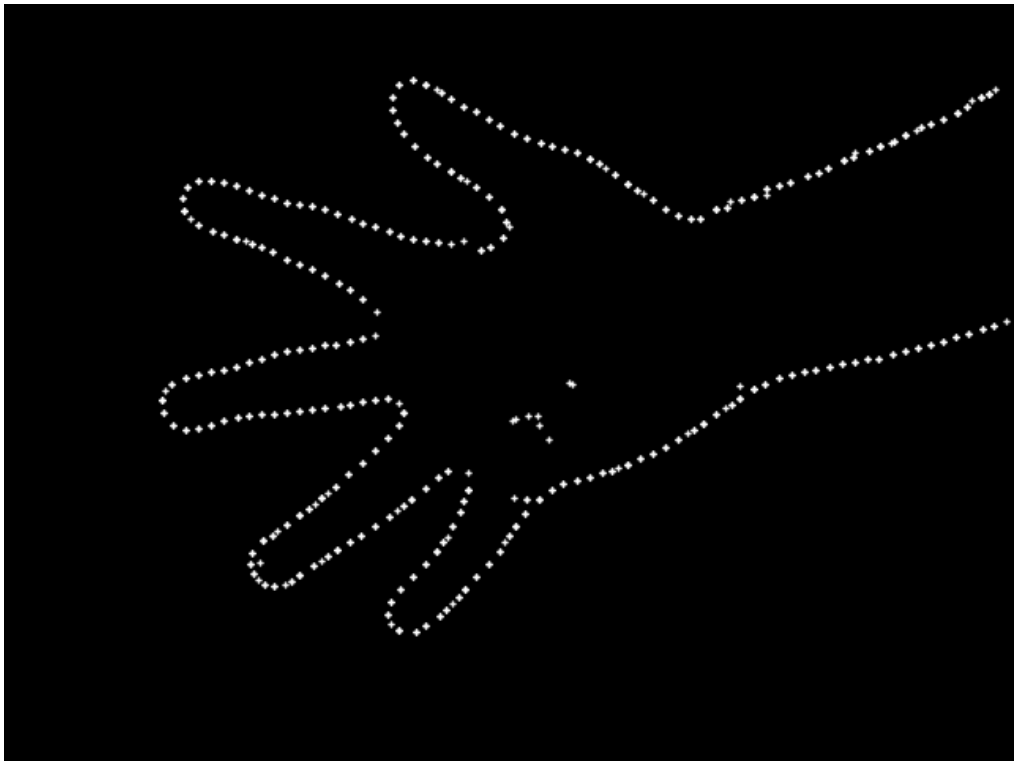

Figure 4.5: Grid with the gravity center of every field as white pixels

```
//Variables
simplified_grid[][] // Two-Dimensional grid
cellsize // Width and height of a cell in the grid
minimum_pixel // Min number of pixels for a keypoint
keypoints[] // Vector: For keypoints

// Call to prepare the grid
function prepare_grid(contours_binary[])
{
   for(every cell of cellsize*cellsize pixels)
   {
      x_sum;
      y_sum;
      pixels;

      // Calculate the sum of all contour pixel coordinates
      for(every pixel IN the cell)
      {
         // If the pixel is a contour pixel
         if (contours_binary[index])
         {
            x_sum += X_VALUE_FOR_INDEX;
            y_sum += Y_VALUE_FOR_INDEX;
            pixels += 1;
         }
      }

      if (pixels > minimum_pixel)
      {
         // Calculate the center
         keypoint_x = x_sum/pixels;
         keypoint_y = x_sum/pixels;

         // Add a new keypoint
         keypoints.add(new keypoint(keypoint_x, keypoint_y));
      }
   }
}
```

4.3.4 Edge Segments Computation

This set of "key-points" on the image needs to be transformed into actual segments or lines to complete the simplification of the contour. Those segments consist of a start-key-point and an end-key-point. They are computed through a trivial method by checking all key-points and finding the most valid neighbor.

The algorithm simulates how it could follow the contour by moving from key-point to key-point. It utilizes the idea that real objects perceived by the camera

wont have jagged structures within the distance of one square field on the simplified grid. Therefore it will prefer the key-point which results in the lowest change of direction and continue that way. Key-points which have been checked once are marked as visited and will be ignored in further encounters.

The simulation ends when it reaches a key-point without any further key-point in a neighbor cell to continue. If this is the first or second key-point on this simulation, the segment is omitted because it cannot enclose a visible object and therefore must be an error. The result of this algorithm is a set of segments which form a smoothed outline defined by a comparably low number of points instead of pixels.

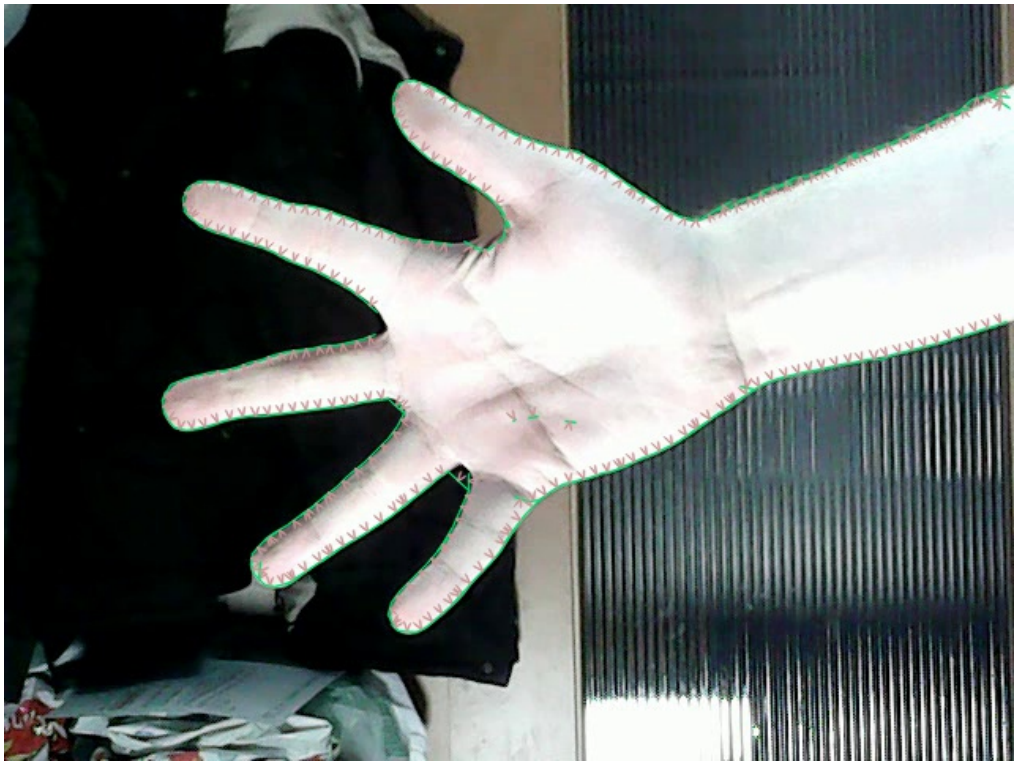

Figure 4.6: Smoothed segments (green) ontop of the original image

```
//Variables
segments[] // Vector: For segments

// Call to compute the segments from keypoints
function compute_segments(keypoints[])
{
   for(every keypoint IN keypoints[])
   {
      if (keypoint[index] has not been visited before)
      {
         Mark keypoint as visited;

         // Start with any direction
         last_direction = ANY_DIRECTION;


         while (there is an unvisited keypoint nearby)
         {

            possible_directions[];

            for(every unvisited keypoint nearby)
            {
               possible_directions[index] =
                  DIRECTION_TO THIS_KEYPOINT;
            }

            if (there is a direction SIMILAR to
               last_direction IN possible_directions)
            {
               // Set to the new direction
               last_direction = NEW_DIRECTION;

               // Add the the new segment
               segments.add(new segment(
                  keypoint[index], keypoint[NEW_INDEX]
               ));
            }
         }
      }
   }
}
```

This method is not the only possible approach for the problem of finding successful contours. Many algorithms are basing on finding fragmented contours and fits them into cycles to achieve the desired information.[5] For the purpose of VT those approaches produce more information than required at the cost of far higher effort of computation.

4.3.5 Inside-Outside Determination

The previously described algorithm has the downside that the resulting set of data does not contain the information which part of the image is inside and which is outside of the objects detected by the camera. The segments require a classification whether inside is on their "right", if looking from the start- to the end-key-point, or on their "left".

For determining this, another grid of square fields of the same size like for the key-points is required. It bases on the first binary image with pixels classified in "object" and "background". Every field contains the value how many pixels are "object". The result is that fields inside the object have a very high value whilst all outside a low one. Fields crossing the contour are in-between.

Figure 4.7: Dark gradient: Few pixel inside object. Bright gradient: Many pixels inside object

Based on this grid every previously computed segment compares whether fields on their right or their left have a higher value and therefore is more likely inside the object seen by the camera. This allows for the required classification.

[5] cf. Ale, Introduction.

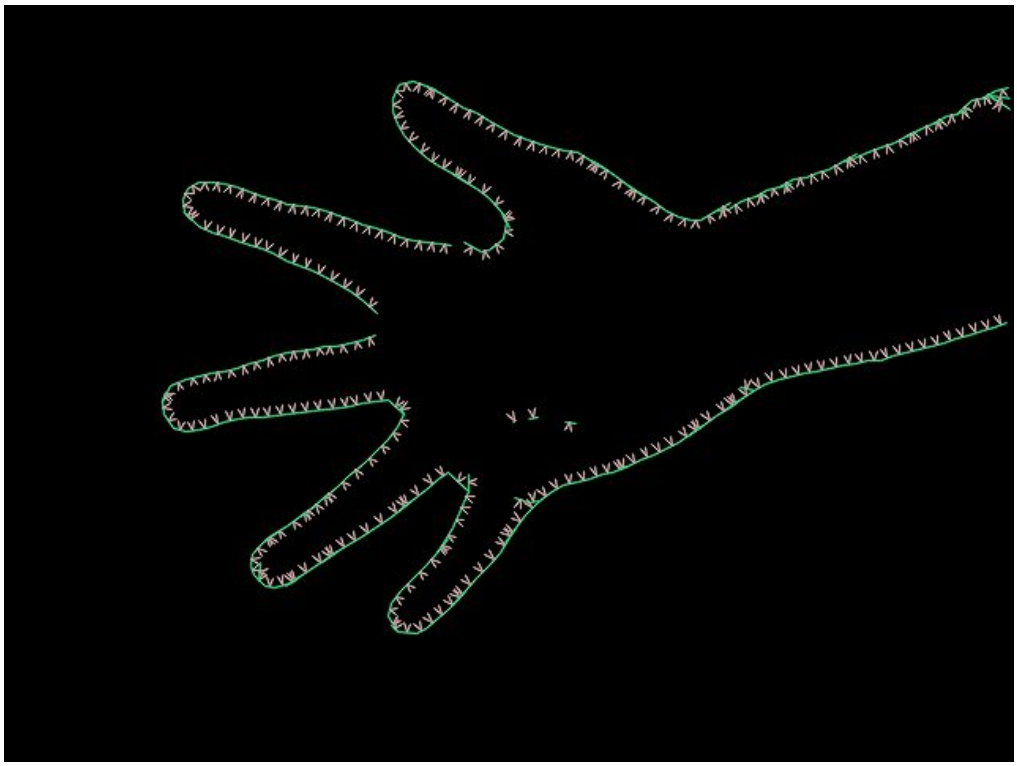

Figure 4.8: Green: Smoothed segments - Orange: Direction towards inside

4.3.6 Possible Optimizations

There a several possibilities to improve the accuracy and results of the algorithms. Their practical investigation exceeds the purpose and range of this document though. For the sake of completeness a small overview:

Dynamic background

The process of removing the background (cf. Background Removal) can be made dynamic by gradually adjusting to slow change in the background. This way changing illumination can be removed as a source of error.

"Object" / "background" classification

The determination whether pixels count as inside or outside an object (cf. Background Removal) can increase its reliability by using other comparison techniques than directly comparing colors. Alternatively a different method based on "Superpixel Grouping" can replace the need for a background image for comparison.[6]

Contour pixels

The determination of contour pixels (cf. Contour Pixel Computation) can be improved at cost of computation effort by comparing more pixels than only the direct neighbors.

[6] cf. Ale.

One key-point in more than two segments

The algorithm computing the edge segments (cf. Edge Segments Computation) can allow a key-point to be part of more than two segments by not marking them as visited by the simulation along the contour. This results in cycles which need to be filtered out.

4.4 Transfer of Segments to Stripes

Once the segments have been computed for every camera this 2D data has to be transferred into the virtual 3D-space. The base for this is the virtual camera-port for every camera in 3D-space. The camera-port is a square in space simulating that the camera is not a small point but a large display which perceived the objects exactly vertical. The normal of this plane is the orientation of the camera. This approximation reduces the effort of computation later on.
As referred for the processing overview (cf. Processing) the contour-segments are forming "pairs of rays". Because every segment has a certain length, the ray-pairs are representing partially limited planes, determined by two simple, endless rays - one originating from the start-point of the segment and one from its end-point. This pair of parallel rays will be named a "stripe" in this document.

The origin points for the rays of every stripe can be found on the camera-port through the following formula:

$$POINT = ORIGIN + ORIENTATION * VECTOR(-PORT\_WIDTH/2 + X, -PORT\_HEIGHT/2 + Y, 0)$$

ORIGIN

3D-vector position of the camera .

ORIENTATION

Quaternion of the cameras orientation. This is the normal of the camera-port.

PORT_WIDTH

Width of the camera-port in the virtual 3D-space. This size determines how width the resulting representation of the objects will be.

PORT_HEIGHT

Height of the camera-port in the virtual 3D-space. This size determines how high the resulting representation of the objects will be.

X X-value on the virtual camera view to calculate the point for. This can be the X-coordinate of a starting- or end-point of a segment.

Y Y-value on the virtual camera view to calculate the point for. This can be the Y-coordinate of a starting- or end-point of a segment.

Multiplication *

Rotates a vector by the quaternion.

The direction of the simple rays and therefore the direction the resulting stripes extend towards is identical to the orientation of the camera and therefore to the normal of the camera-port. This also means that all stripes from one camera have the same direction and the computation effort of the remaining steps is reduced by this approximation.

The following image shows how the contours of the example image are simulated on the camera-port which is visualised by the four red crosses in the corners and the arrow in the middle. For illustration purpose they are limited in length as well.

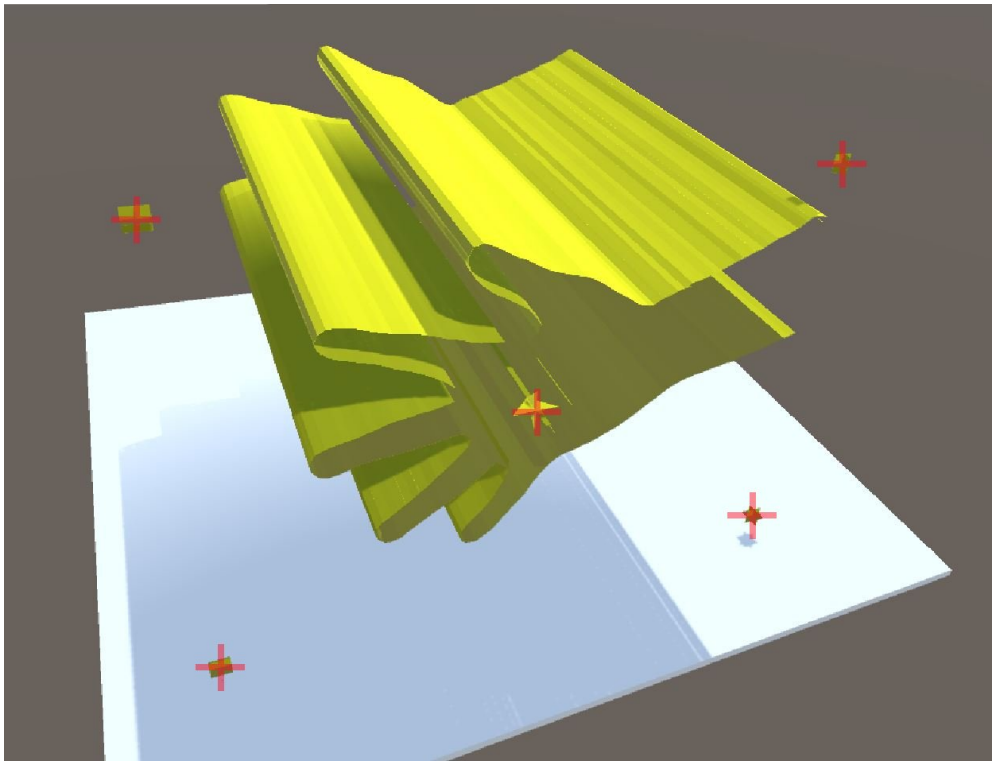

Figure 4.9: Camera-port (red) with visualized rays (yellow) along the contours oriented towards the viewer

4.4.1 Optionally Automated Camera Homography Computation

The required relative positions of the cameras in real 3D-space can also be computed through a calibration process. An option is to use a calibration-object with a well defined pattern. Through a process of image-recognition applied to a frame from every camera the individual homographies between the views of the cameras can be determined as well as their distance to the object. The result is a higher flexibility of the entire VT system because the position of the cameras do not need to be provided by the user.

4.5 Computation of Object Surface

The next steps are basing on the prepared ray-pairs/stripes from all cameras for the current frames.

4.5.1 Intersection of Stripes

To eventually compute which sections on every stripe are part of the surface of the virtually reproduced 3D-object all possible intersections between the stripes need to be determined. Because the stripes from one camera are parallel they only have to be checked against every stripe from all other cameras.

The actual algorithm requires the following steps for every permutation of stripes:

1. Computing the intersection line of the two unlimited planes represented by the stripes. As the limitations by the starting-point-ray and the end-point-ray are ignored, "two planes P1 and P2 are either parallel or they intersect in a single straight line L."[7] The following steps are omitted if the planes are parallel and therefore no collision line is obtainable.

2. Determining the positions along the plane-intersection line L where it intersects the four simple rays of the stripes.**??**. "When the two lines or segments are not parallel, they might intersect in a unique point."[8]

[7] Sun, Intersection of 2 Planes.
[8] Sun, Non-Parallel Lines.

3. The classification how the stripes intersect each-other is made by the order of the ray-intersections with line L. There are three possibilities for stripe A and stripe B:

   I.) A and B miss each-other completely when the start- and the end-rays of one stripe are intersecting L before or completely after the other stripe intersects.

   II.) A and B overlap partially if an intersection on L with the start-ray of A happens in-between the start- and end-rays of B but the end-ray of A intersects L afterwards.

   III.) A is completely inside B when start- and end-ray of A are both intersecting L in-between the start- and end-rays of B.

   Every possibility can be mirrored by exchanging A and B.

4. In the case if I.) there are no collisions. The positions of intersection along the edges of the stripes are saved in the other cases.

The following two images show the results of the example image together with a counterpart from a second camera.

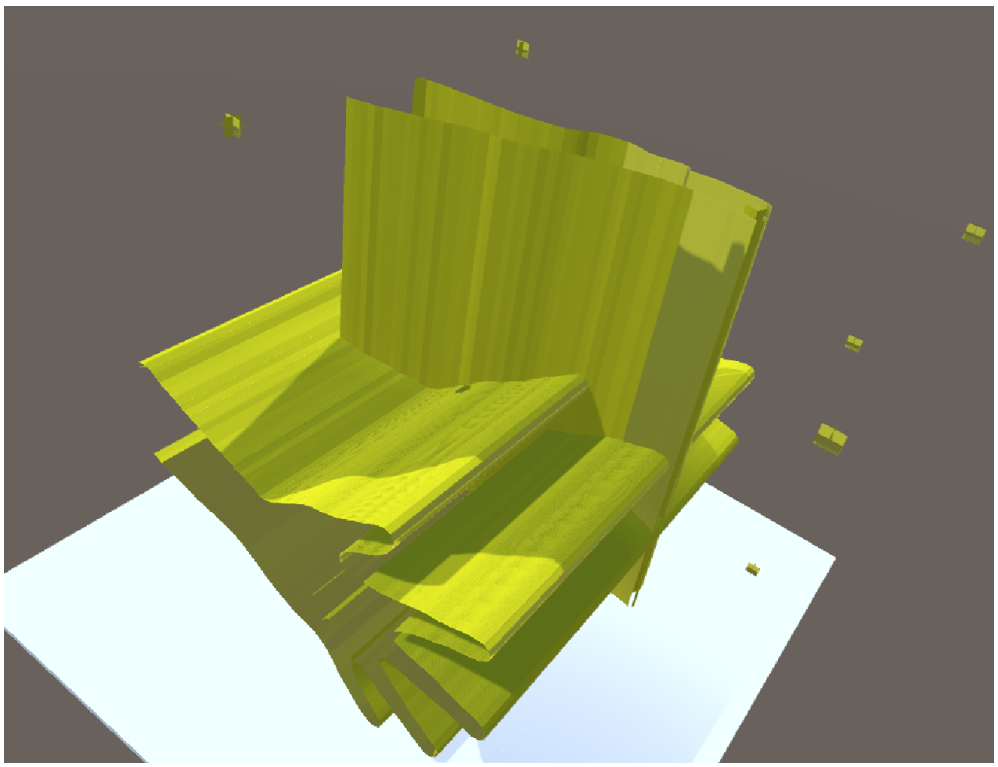

Figure 4.10: Example of a horizontal and a vertical set of stripes from two cameras

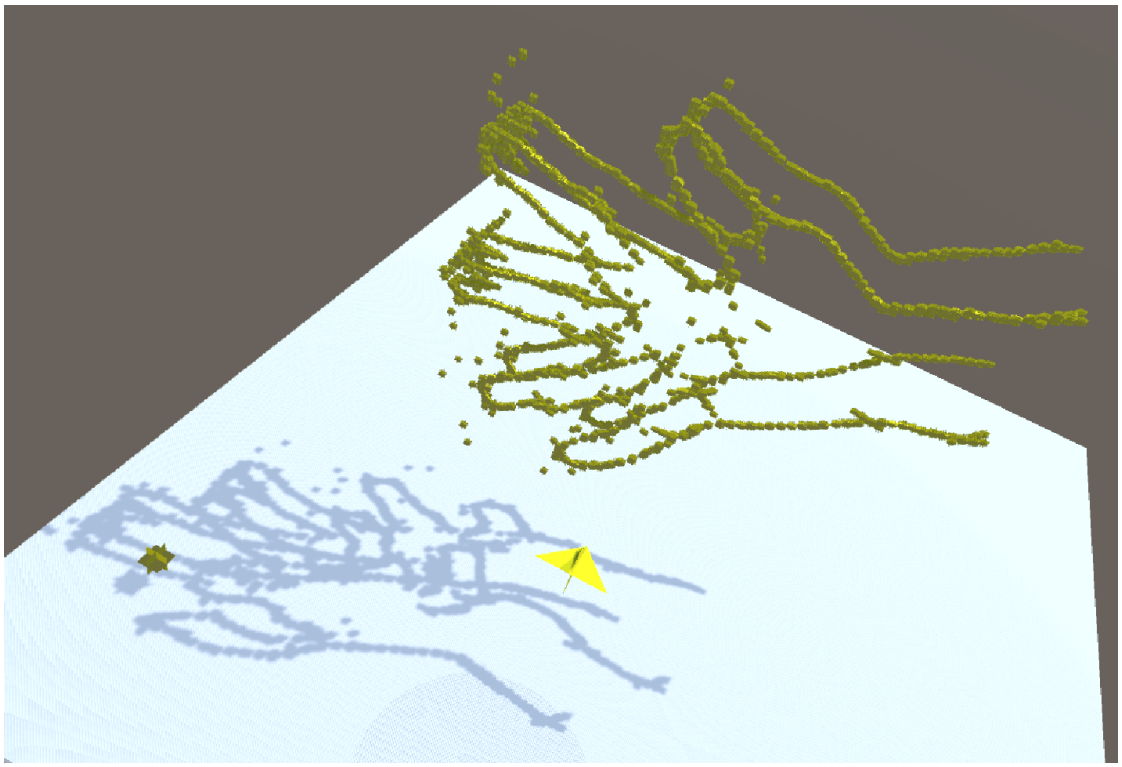

Figure 4.11: Intersections visualized for the example image together with a corresponding image for a second camera

```
// Calculate the intersections for a given camera
// own_stripes: The stripes produced by that camera
// other_stripes: The stripes of all other cameras
function calculate_intersections(
   own_stripes[], other_stripes[])
{
   for (every stripe IN own_stripes)
   {
      for (every stripe IN other_stripes)
      {
         // Compute the intersection line of
         // the own stripe with the other
         intersection_line = intersection(
            own_stripes[own_index], other_stripes[own_index]
         );

         // The position on the intersection_line where
         // it intersects the start_ray of the own stripe
         own_pos_start = intersection(
            intersection_line,
            own_stripes[own_index].start_ray
         );

         // The position on the intersection_line where
         // it intersects the end_ray of the own stripe
         own_pos_end = intersection(
            intersection_line,
            own_stripes[own_index].end_ray
         );


         // The position on the intersection_line where
         // it intersects the start_ray of the other stripe
         other_pos_start = intersection(
            intersection_line,
            other_stripes[other_index].start_ray
         );

         // The position on the intersection_line where
         // it intersects the end_ray of the other stripe
         other_pos_end = intersection(
            intersection_line,
            other_stripes[other_index].end_ray
         );
```

```
      // If the two stripes are partially or
      // completely overlapping each other
      // (Variant II. or III. )
      if ((own_pos_start > other_pos_start)
         && (other_pos_end > own_pos_start))
         {
            // Save the current intersection into the
            // vector of intersections for this array
            own_stripes[own_index].intersections.add(
               THIS_INTERSECTION);
         }
      }
   }
}
```

4.5.2 Division into Start-Edges and End-Edges

Similar to the computation of the segments from the 2D image the information which pairs of intersections on a particular stripe are actually on the surface of the virtual object and which are not, is missing. Principally it is possible to classify intersections into "start-edges" and "end-edges" by alternating between the two possibilities. This does not yield an acceptable level of quality because of the natural inaccuracy of the camera input cameras. A single error at the start of a strip suffices to produce an incorrect classification for all further intersections of the same strip.

The information for every intersection on the stripe can be reproduced from the segment associated with the stripe which caused the intersection. The segment has the classification stating whether "inside" is on the right or on the left (cf. Inside-Outside Determination). Additionally the actual result whether an intersection on a stripe is starting a section on the perceived 3D objects surface or is ending it, depends on the origins of the stripes and therefore on the orientation of the two cameras initially producing this intersection.

A certain intersection for stripe A with stripe B is a:

"Start-Edge" when

The origin of stripe A is on the RIGHT side of the plane formed by the stripe B by taking into account how the start- and the end-point of the associated segment are oriented.

AND

B is classified as having the "inside" on the LEFT.

OR

The origin of stripe A is on the LEFT side of the plane formed by the stripe B by taking into account how the start- and the end-point of the associated segment are oriented.

AND

B is classified as having the "inside" on the RIGHT.

"End-Edge" when

The origin of stripe A is on the RIGHT side of the plane formed by the stripe B by taking into account how the start- and the end-point of the associated segment are oriented.

AND

B is classified as having the "inside" on the RIGHT.

OR

The origin of stripe A is on the LEFT side of the plane formed by the stripe B by taking into account how the start- and the end-point of the associated segment are oriented.

AND

B is classified as having the "inside" on the LEFT.

4.5.3 Filter to Start-End Pairs

With the division into "start-edge"- and "end-edge"-intersections as computed from the possibly erroneous or inaccurate camera input it is possible to omit invalid intersections. Along one stripe "start-edges" and "end-edges" have to alternate after starting with a "start-edge" and it must end with an "end-edge" adequately. There are multiple ways to handle occurrences contradicting this principle like successively arranged same-classified intersections:

I.) Ignore all further intersections along the stripe until a matching one occurs.

II.) Ignore all preceding intersections until it matches with the current intersection.

III.) Compute a new intersection from the average of the same-classified intersection.

All variants result in an output of similar quality because errors produced by the camera are not predictable.

4.5.4 Direct Computation of Quads

The final step to produce the desired set of data (Desired Data) requires to transform the stripes and the classified intersections into polygons forming 3D-space. Because the stripes have a width and the valid intersections always form quadrangles it is possible to use "quads" as an intermediate data form.

The algorithm handles every stripe independently and computes a quad from every pair of "start-edge" and "end-edge". The four 3D-coordinates for the quad are the following:

1. "Start-edge"-Intersection along the ray of the "start-point" of the associated segment
2. "Start-edge"-Intersection along the ray of the "end-point" of the associated segment
3. "End-edge"-Intersection along the ray of the "start-point" of the associated segment
4. "End-edge"-Intersection along the ray of the "end-point" of the associated segment

4.5.5 Alternative Computation of Quads

The method of directly computing the quads from the intersections along the start- and end-rays has the downside that partially overlapping stripes are having the same result as if they were intersecting completely. This increases the possible inaccuracy of the3D-result to the maximal width of all stripes. This is double the diameter of one grid-field used for the key-point simplification (cf. Contour Key-Points Computation) because it determines the maximal length of one segment.

This error can be omitted by saving the information how much of the strip has overlapped when it was only partially. Afterwards the stripe needs to be split into sub-stripes for every partial intersection before computing the quads for every sub-stripe. The graphic shows how a constellation of three intersections results in multiple sub-stripes.

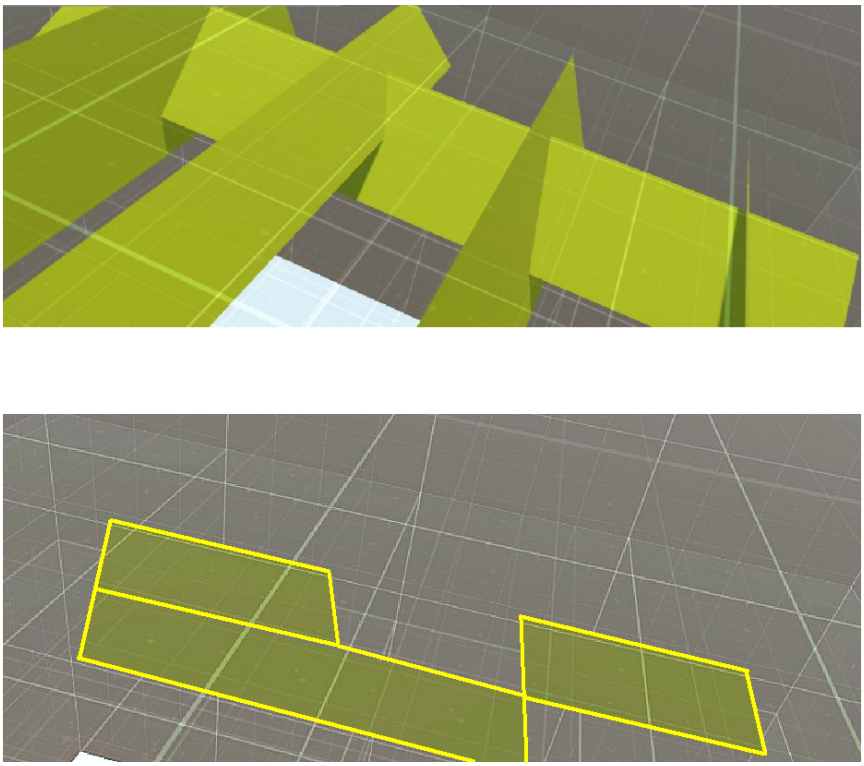

Figure 4.12: Top: Entire stripes - Bottom: Eventually visible sub-stripes

The algorithm producing the sub-stripes needs to be executed for every stripe with valid intersections. This significantly increases the computation effort of the whole process.

4.6 Computation of Object Texture

The set of polygons in the virtual representation of the real objects require texture to achieve the effect of VT.

An intersection of stripe A from camera cA with the stripe B from camera cB means that when this intersection results in a corner of a quad for A, the glsquad will display the texture of the frame of camera cB. The corresponding UV-coordinate of this corner of the quad is the origin of the ray from cB which has resulted in this intersection.

The same procedure can be applied to every corner to enable correct texture-rendering for every polygon/quad.

4.7 Performance Hints

Part of the concept of VT and the objectives for the implementation described by this document (cf. Objectives for Implementation) is the capability of performing the entire process in real-time with low-cost hardware. That requires an optimization

for performance. The following set of recommendations with examples can guide to achieve the desired performance:

Using Parallelism

At the date of this document the majority of computers used by customers are using two or more CPU-cores[9]. That means at least two threads can be executed at the same time by the hardware. The implementation can make use of that by splitting independent computations into multiple threads.

The entire camera processing (cf. Camera Processing) can be executed in one thread for every camera or even further by dividing every frame into multiple areas.

After the contour-segments and stripes from all cameras have been prepared the intersections can be computed in multiple independent threads together with the final calculation of quads.

Avoid unneeded computations

Some computations can be cached in variables instead of recomputing them every time when needed. Examples are some values required for the algorithm which computes intersections of planes and lines.

Additionally before computing the intersection it is possible to check whether the two strips are in range of each other at all and omit the further calculation in that case.

Use low-level programming

"Higher programming languages" produce an overhead in computations and therefore are less suited for implementing an algorithm in an efficient. Low-level languages provide more control over the hardware like the memory and allow for more optimization.

Reduce non-continuous memory access

Modern hardware has efficient caching mechanisms to improve performance for computations. This takes best effect when data is stored consecutively in memory. Rapidly accessing memory which has been allocated at different times can reduce performance. Arrays containing separately allocated arrays are an example of that.

Use GPU computing

Parts of the algorithm like the computation of stripe-intersections or the final quads can be divided into more independent parts and threads than CPUs of currently available computers can handle efficiently. GPU-based processing makes use of the high number of processing units in graphic-cards and therefore can speed up process significantly.

[9] cf. Ste.

4.8 Possible Improvement with Stereo-Cameras

The number of 3D details the implemented method can display is directly limited by the number of cameras which provide the strips forming the polygons and the method cannot reproduce concave objects when there is no perspective for the camera to perceive the concave section. To increase the accuracy and avoid a reduction of quality on concave objects it is possible to use stereoscopic camera input. By using pairs low-cost cameras the objectives can still be fulfilled.

Publicly available software components exist for this purpose. An example is LIBELAS.

The output of the stereoscopic computation is information about the the distance of every position on the input-image. This data can be used to split every quad into more than two polygons and provide structure to the surface of the virtually reproduced object. The depth-information can be directly mapped with the UV-coordinates of the texture.

Chapter 5

Objective Evaluation

The realized concept and the applied method can be compared to the initial objectives (cf. Objectives for Implementation) to determine how accurate the goal of market penetration could be fulfilled by using the right pricing strategy (cf. Pricing Strategies)

5.1 Hardware Cost

The cost in hardware required to use the implemented method amounts to lower than 8€ (8.6$) per camera at the date of this document. For achieving acceptable results at very least two cameras are required. Four cameras yield higher quality on complex objects. A higher investment in camera quality improves the result as well. The required amount remains in the range of the consumer and lies significantly lower than .

A computer is required for actual usage by the consumer. Today's sub average computers suffice for the application if the method has been implemented efficiently as recommended (cf. Performance Hints). Additionally the computer has to provide the possibility to attach the cameras to provide input.

5.2 VT Quality

The method does not yield a high level of detail without additional input improvement (cf. Possible Improvement with Stereo-Cameras). The quality of the result is sufficing for testing purpose and experimentation. Certain camera perspectives and objects can be found which are leading to a relatively accurate reproduction in the virtual environment. Despite low detail, the experience from increased immersion is resulting in a significant quality.

5.3 Conclusion

The general concept of VT can be visualized with the described method and the quality suffices for a first impression of the possibilities. Compared with the low required investment the yield is significant. Comparisons are not possible because at the date of this document no other products with the same target are available on the market.

When comparing to the speed of development of the VR market instead and taking the meaning of VT for the user-experience (cf. VT as a Component of Virtual Reality (VR)) into account, it can be predicted that a demand for VT is present and be fulfilled within few years. The implementation shows that an introduction on the market and an approach to consumer is feasible with this concept.

Glossary

2D two-dimensional. 2, 12, 14, 28, 33

3D three-dimensional. 10–12, 14, 15, 17, 21, 28–30, 33–35, 38, 41–43

algorithm
refers to a set of operations usually performed by a machine to fulfill a purpose. Software programs usually consist of algorithms. 14, 43

API
The "Application Programming Interface" is a method to enable the usage of a software-library from another program.. 17, 42

C++
is a low-level language for source code.. 18, 42

camera-port
a virtual plane projected by a camera.. 28, 29

encoding
in this case refers to the method of compressing video or image data.. 16

engine
in this context refers to a self-containing software component with a certain purpose. An example is a "game-engine" which serve as the base of a game.. 17, 43

hardware
refers to physical components of a machine. For example parts of a computer, input devices and output devices. 9, 13, 36, 37, 42

homography
in this case refers to the way two planes in 3D space are oriented to each other.. 29

immersion
in the cases relevant here describes the perception of being physically part of

something virtual or being part of the virtual reality. The intensity immersion can be compared and is influenced by many factors.. 7

input device
a category of computer hardware required to input information and commands to a computer. Examples are mouse and keyboard but also webcams and VT equipment. 9, 17, 41

low-cost consumer equipment
refers to equipment available without special requirements like buisness relations and requires very low investment. 2, 3

low-level
refers to a set of operations usually performed by a machine to fulfill a purpose. Software programs usually consist of algorithms. 16, 17, 37, 41

market penetration
describes how successful a product is by counting by sales numbers. 2, 9

normal
here is a vector aligned orthogonal on a plane or surface.. 28

OpenCV
The "Open Source Computer Vision Library" is a software component providing algorithms and data structures to perform computation with digital images. The source code is originally programmed in "C++" and provides an API in various other languages.. 16, 17

open-source
refers to the concept that a piece of software is freely available as unobstructed source code. Special licenses and copyright may apply.. 18

output device
a category of computer hardware required to provide information to the user. An example are monitors but VR equipment counts as well. 16, 41

polygon
is a set of three points in (here in 3D-space). The surface between the points is used for rendering. . 36, 38, 43

polygon-coordinates
coordinates of the points of a polygon.. 15, 17

pseudo code
is a form of source code designed for the human reader and is not usable on any machine. It allows to represent algorithms and concepts in a more efficient way than real source code. 2, 10, 14

quad
is a set of two polygons forming a rectangle on a 3D-plane.. 35–38

ray is a data-structure for a 3D environment consisting of a point of origin and an orientation or direction.. 15, 28–30

real-time
describes that something happens without a significant delay so it can be perceived by an user as if it were happening directly. 2, 6, 7, 22, 36

render
is the process of rendering.. 17

rendering
in this context refers to the process of turning 3D-data like polygons and textures into a two dimensional image. A special case is VR-rendering which requires that the same data is rendered into two images differing slightly by perspective to simulate a pair of eyes.. 15, 17, 36, 42, 43

software
refers to digital, non-physical components of a machine. For example programs and drivers. 9, 41, 42

source code
describes text written in a certain programming language. Such text can be interpreted either directly or indirectly by a computer to fulfill a purpose.. 2, 41–43

stereoscopy
is the concept of computing a depth-data from two images of the same scene with slightly different perspective.. 12

texture
refers in this case to the color of a surface or plane in 3D space.. 36, 43

Unity 3D
in this context refers to the game-engine "Unity 3D": https://unity3d.com/. 17

UV-coordinate

describes what coordinate on the associated texture-image a certain point of a polygon requires when rendering the polygon with texture.. 36, 38

virtual environment

describes the idea of virtually representing the appearance of a certain environment or an alternate reality. Various devices like head-mounted displays are used to deceive the senses of a person and allow an immersion into the virtual environment. 6, 7, 10–12

virtual reality

describes the concept of immersing people into digitally generated environment by shielding them off the real environment and artificially stimulate the senses. 2, 3, 6, 7, 42

virtual teleportation

describes the concept of digitalizing visual objects or structures and reproduce them for the human eye in a digital environment. 2, 3, 7

VR

virtual reality. 6, 7, 9, 16, 17, 40, 42, 43

VT virtual teleportation. 6, 8–10, 12, 16, 17, 25, 29, 36, 40, 42

Bibliography

[201] Business Set Free Ltd 2013. *Your Small Business Pricing Strategy.* URL: http://www.businesssetfree.com/pricing-strategy/.

[Ada04] Ernest Adams. *Postmodernism and the Three Types of Immersion.* July 2004. URL: http://designersnotebook.com/Columns/063_Postmodernism/063_postmodernism.htm.

[Ale] Sven Dickinson Alex Levinshtein Cristian Sminchisescu. *Optimal Contour Closure by Superpixel Grouping.* URL: http://www.cs.toronto.edu/~sven/Papers/eccv2010-closure.pdf.

[CVa] Open CV. *Open CV Class VideoCapture.* URL: http://docs.opencv.org/3.1.0/d8/dfe/classcv_1_1VideoCapture.html.

[CVb] Open CV. *Open CV Introduction.* URL: http://docs.opencv.org/3.1.0/d1/dfb/intro.html.

[DOn15] Jillian D'Onfro. *How Facebook plans to more or less build a teleporter.* Nov. 2015. URL: http://uk.businessinsider.com/facebook-oculus-plans-teleporter-by-2025-2015-11?r=US&IR=T.

[Eng11] American Heritage® Dictionary of the English Language. *American Heritage® Dictionary of the English Language, Fifth Edition.* 2011.

[Eng16] American Heritage® Dictionary of the English Language. *American Heritage® Dictionary of the English Language, Fifth Edition.* 2016. URL: https://ahdictionary.com/word/search.html?q=virtual+reality.

[Koc09] Kevin Kocher. *Finding Possible Solutions to Avoid Future Deficient Covers of Swiss Pension Funds.* Aug. 2009. URL: http://www.fhnw.ch/wirtschaft/dienstleistung/studierendenprojekte/olten/bisherige-projekte/bachelor-thesis-2009/finding-possible-solutions/finding-possible-solutions.

[Mic] Microsoft. *Holoportation.* URL: https://www.microsoft.com/en-us/research/project/holoportation-3/.

[org] OpenSource org. *The 3-Clause BSD License.* URL: https://opensource.org/licenses/BSD-3-Clause.

[Rie02] Wolf-Fritz Riekert. *Eine Dokumentvorlage für Diplomarbeiten und andere wissenschaftliche Arbeiten.* Apr. 2002. URL: http://v.hdm-stuttgart.de/~riekert/theses/thesis-arial11.doc.

[Ste] Steam. *Steam Hardware and Software Survey: December 2016.* URL: http://store.steampowered.com/hwsurvey/cpus/.

[Sun] Dan Sunday. *Intersection of Lines and Planes*. URL: `http://geomalgorithms.com/a05-_intersect-1.html`.

[Uni] Unity. *Unity - VR overview*. URL: `https://docs.unity3d.com/Manual/VROverview.html`.