Real-Time Free Viewpoint Synthesis using Stereo Camera Depth Map Estimation Hardware

MASTER THESIS

Jonathan Masur

Electrical and Electronic Section - Master Semester 4

Supervisor : Abdulkadir Akin
Professor : Yusuf Leblebici

Laboratoire de Systemes Microelectroniques (LSM)
July 4th, 2014
Contents

1 Introduction 2

1.1 Introduction ......................................................... 2
1.2 Camera model ......................................................... 3
1.3 Disparity and distance ............................................. 4

2 Generation of an intermediate image 6

2.1 Context ................................................................. 6
2.2 Forward vs. reverse mapping ...................................... 7
2.3 Artificial depth map generation .................................... 9
2.4 Reconstructing the image with reverse mapping ................. 10

3 Simulation results 12

3.1 Original images ......................................................... 12
3.2 Interpolation ............................................................ 12
3.3 Extrapolation ............................................................ 13
    3.3.1 Virtual viewpoint located right of the right camera .......... 13
    3.3.2 Virtual viewpoint located left of the left camera ............. 14

4 Hardware implementation 16
4.1 Overview .............................................. 16

4.2 Row-based pipeline scheme ................................ 17

   4.2.1 Explanation of the concept .......................... 17

   4.2.2 Beginning and end of the frame ................. 18

   4.2.3 Scheduling ........................................ 19

   4.2.4 Left to right and right to left processing .......... 19

4.3 Notes about the schematics and figures ................. 20

4.4 First approach - Depth map shifting in a single step ... 22

   4.4.1 Shifting and inpainting in one step ............ 22

   4.4.2 Maximum circuit ................................... 24

   4.4.3 Conclusion over the first approach ............. 25

4.5 Second approach - Depth map shifting and inpainting in two steps ... 25

4.6 Image construction and output .......................... 27

5 Future Improvements .................................... 28

   5.1 Improving disparity estimation ......................... 28

      5.1.1 The problem of occlusion ..................... 28

      5.1.2 Improvement by detecting occlusion ........ 29

      5.1.3 Improve with a third camera ................... 30

      5.1.4 Filtering .......................................... 30

   5.2 Three cameras ....................................... 31

   5.3 Rectification ......................................... 32

   5.4 Moving in other directions ............................ 32

      5.4.1 Vertical movement ............................... 32
Abstract

Depth estimation is an algorithmic step in a variety of image processing applications such as autonomous navigation, robot and driving systems, 3D geographic information systems, object detection and tracking, medical imaging, computer games and advanced graphic applications, 3D television, multiview coding for stereoscopic video compression, and disparity-based rendering. These applications require high accuracy and speed performances to achieve real-time depth estimation. The computational complexity of depth map estimation algorithms and the need of large size and bandwidth for the external and internal memory make the real-time processing of depth estimation challenging, especially for high resolution images.

Recently, a hardware-oriented adaptive window size disparity estimation (AWDE) algorithm and its real-time reconfigurable hardware implementation that targets high resolution video with high quality disparity results are presented. The implemented hardware is verified using modelsim with post-place and route simulations. The implemented reconfigurable hardware architecture of the AWDE algorithm enable handling 60 frames per second on a Virtex-5 FPGA at a 1024x768 XGA video resolution for a 128 pixel disparity. Moreover, recently, real-time hardware implementation of stereo camera rectification is presented. The rectification hardware is also verified using modelsim with post-place and route simulation. Finally, the hardware is mapped to Virtex-7 FPGA, the depth map results and the original images are displayed in PC using QT based Graphical User Interface (GUI).

The target of this project is to develop real-time 3D-Rendering application using the high-resolution real-time depth map estimation system. In order to reach this goal, first, several papers about 3D rendering are analysed. Afterwards, low complexity and hardware oriented free viewpoint algorithm is implemented in Matlab to analyse the algorithm performance. Finally, 3D rendering algorithm is implemented in real time hardware to finalize the project goal. The implemented hardware can be used to synthesize interpolation and extrapolation views on a straight line using two parallel cameras. Moreover, the implemented rendering hardware is tested with 3 camera disparity estimation system to provide better visual performance.
Acknowledgements

Special thanks should be given to people who also participated to the project:

• Yusuf Leblebici, director of LSM at EPFL, for allowing this project to be made in LSM lab
• Abdulkadir Akin, PHD student at LSM of EPFL, for supervising this project and a group of other projects closely related to this one
• Thomas Maugey, postdoc from LTS4 of EPFL, for cooperating with us and teaching us how to use and improve rendering algorithms
• Luis Gaemperle for developing the embedded system running the disparity estimation hardware and for supporting this project with his good advice
• Alain Vachoux & Alexandre Schmid for doing system and server maintenance in the LSM
• Sylvain Hauser for building the mechanical setup for holding the cameras
Chapter 1

Introduction

1.1 Introduction

This work was inspired by various research papers and documents mentioned in the references [1-7]. As a basis of this research, a system with two high resolution synchronised cameras is used. The disparity results provided by the real time hardware described in references [8-9], and is used to measure precisely the distance between the cameras and the filmed objects.

The work of this thesis will focus on generating another image for a viewpoint which can, to some extent, be arbitrarily chosen by the user. The whole process, from image acquisition to viewpoint synthesis, is entirely done in hardware in real-time. The block diagram of the system is shown in Figure 1.1.

Several assumptions are made in order to create a virtual view:

- The two cameras are perfectly calibrated, identical and are capturing noise and distortion free images.
- The system is only generating views on the straight line that passes through the two cameras.
- No visible object is closer to the cameras than the minimum distance that can be measured by the disparity estimation system.
- The cameras are not filming a mirror or anything which is subject to diffraction. Any lighting effects and reflections are neglected.
- The disparity estimation system is reliable and accurate.
1.2 Camera model

It is important to define how the unbounded, real 3D world will be filmed by the camera and converted into a bounded, finite 2D image, through an ideal model.

The camera is modelled as a single pinhole point which throws rays. The total angle of view of the camera is named $\gamma$. Rays are captured by a very small plane surface inside the camera.

Objects which are within the viewing range of $\{-\frac{\gamma}{2} \cdots \frac{\gamma}{2}\}$ are displayed on the captured image. The origin is placed at the centre of the image.

This model is valid for both the X and Y direction, and illustrated in Figure 1.2. The figure only shows displacement in the X direction, for simplicity.

Using the described camera model, the changes on the views by considering movement of the objects in any cardinal directions can be explained as following:

- When moving along a plane perpendicular to the camera beam, it will result in a scrolling object on the 2D image (getting out of the beam implies scrolling out of the image)
- When moving along the straight line between the camera and the object, it will result in a scaling transformation (or zoom) on the 2D representation of the object
- Any 3D movement without any form of rotation can be expressed as a combination of the previous two (scaling + scrolling)
1.3 Disparity and distance

Now that a camera model was established, the direct relation between the disparity values and the distance between the cameras and the object (also known as depth) is going to be shown.

The disparity, which is just a number without any dimension, represents the amount of pixels a particular point shifts between one view and another (usually between the left and right camera’s images). The depth is a physical distance, and can be measured in centimetres for example. The relationship between those two will be shown, and is illustrated in Figure 1.3.

Let $F$ be the focal length of the camera, $d$ be the baseline, the distance between cameras, $D$ be the distance between the line that passes and $\delta$ the disparity, and $\gamma$ the viewing angle of the camera. The
formula between disparity and distance is as following.

\[ \delta = \frac{F \cdot d}{D} \]

The focal length \( F \) can also be expressed in function of the viewing angle \( \gamma \).

\[ F = \frac{X_{\text{res}} \cdot \tan(\frac{\gamma}{2})}{2} \]

Which in turns allows to know the disparity using only the viewing angle \( \gamma \) instead of focal length \( F \).

\[ \delta = \frac{X_{\text{res}} \cdot \tan(\frac{\gamma}{2}) \cdot d}{2 \cdot D} \]

Noting that the disparity between cameras is **inversely proportional** to the distance, with a constant multiplicand factor which depends on the camera, resolution and separation between cameras. If the cameras can be moved arbitrarily by the user, then this factor is not constant anymore.
Chapter 2

Generation of an intermediate image

2.1 Context

As mentioned before, an intermediate image on the axis that connects the two cameras will be generated. This means that objects will only have to be shifted horizontally, by an amount of pixels which depends on their distance.

The displacement of an object, in pixels is linearly proportional to the disparity. Considering the left camera is at position $x = 0$ and the right camera at position $x = d$, then the displacement for a synthesized view for position $x = p$ for an object at distance $d$ is:

$$\text{displacement} = \frac{p}{d} \cdot \text{disparity}[x]$$

It’s possible to do interpolation of the image where the generated image is between the cameras, as drawn on the picture. It is also possible to do extrapolation, where the generated image is on the left or on the right of both cameras, although it will be shown that extrapolation is a bit more challenging.

It’s therefore relatively trivial to build a resulting image by simply applying this principle to every
individual pixel on the image, but two problems appear:

- Multiple pixels from the original image can go to the same location in the generated image.
- A pixel in the resulting image may have no pixel in the source image that maps to it.

The first case is *overlapping*, and is simple to solve: The pixel which is the closest (i.e. biggest disparity) wins.

The second case is called *occlusion*, and is quite complicated to solve, since the original image contains no data of the occluded region, which is behind another object.

An example of this simple method is shown in Figure 2.1. The black regions means that occlusion appeared (i.e. no pixel of the original image mapping to that area).

![Figure 2.1: Using disparity and left image to create a right image](image)

There are two reasons for a region to become occluded. There are shadow zones in the background near foreground objects. These regions are occluded because image data for those regions simply wasn’t here in the original image. The second cause of occlusion is noise in the disparity map, which only concerns “clouds” of individual pixels, as opposed to a whole region. A technique to solve the problem of occlusion will then have to be used.

### 2.2 Forward vs. reverse mapping

The difference between forward and reverse mapping algorithm is very important, although they can be considered as two sides of the same coin.

With a **forward mapping** algorithm, the algorithm takes pixels from any of the original camera images and will place them on the reconstructed image. In other words, for each pixel in the source image, the question “*Where does this pixel of the original image go in the generated image?*” is asked.
Any pixel can potentially go to multiple places. The reasoning for forward mapping is more natural for the human perception, but less natural for hardware.

The simple algorithm described in the previous section is a particular case of forward mapping, where only a single input image is used.

With a reverse mapping algorithm, for each pixel of the reconstructed image is asked: "Where does this pixel of the reconstructed comes from in the original images?" The answer is given by the function, and has the advantage that the process is identical for every pixel. The mapping can then be done in constant time in a single step, as the algorithm is called exactly once per pixel in the reconstructed image. This makes the algorithm more friendly to implement in hardware: There is no "gap filling" to do since there is always defined data for each pixel, however, the defined data might be wrong in the occluded areas.

As shown in these graphics, in order to do reverse mapping, a depth map that is aligned with the synthesized view is also needed. However the real depth map is aligned with one of the cameras, so the depth map for the virtual view will have to be first synthesised, using one of the mapping techniques that was already used.
2.3 Artificial depth map generation

As stated in previous section, it’s necessary to synthesize the depth map for this point of view before applying the reverse mapping algorithm. The original disparity map for the left camera is used for this process. Forward mapping is used for the generation of artificial disparity map.

Two major problems arose while constructing depth images, namely:

- Multiple pixels can potentially land to the same place
- Some areas can stay “empty” because of occlusion

![Figure 2.4: Exemple of shifting disparity map](image)

Figure 2.4 shows an example of shifting disparity map. Assuming an object with disparity 3 is in front of a background of disparity 1. For simplification, the position is shifted by \( \frac{d}{2} = 1 \), which means a depth map for the right camera’s position is synthesized, and that all pixels have to be shifted by their disparity multiplied by 1.

The two issues mentioned above can clearly be seen: Overlap (on the left of the object) and occlusion (on the right of the object).

The overlap problem is solved simply by taking the maximum value of all candidates (i.e. the closest object to the camera is chosen to be display).

The occlusion problem is solved by assuming the occluded region is always in the background, so the last defined pixel from the direction the camera is scrolling (i.e. to the direction to which the visible object is scrolling on the screen, so that the occluded area behind them appears to be the background) is simply repeated. This process is called inpainting.
Usually this means going from right to left, as interpolation will be done, and the depth map is aligned with the left camera. However, if extrapolation (generating a view on the left of the left camera) needs to be generated, then inpainting from left to right is required.

![Translated depth map before and after inpainting](image)

It’s also possible to do a “gradient fill”, that is, to fill continuing the 1st order derivative from the last 2 defined pixels. But it was decided that the added complexity wasn’t worth it, as the results could be very erroneous because the noise that is inherently here in the depth map would make such a calculation difficult.

### 2.4 Reconstructing the image with reverse mapping

Now that the depth map has been translated to the new viewpoint, a reverse mapping is going to be used to reconstruct the image itself. The concept is simple, the depth map is converted back into two disparity maps:

- Disparity map between the synthesized image and the original left image
- Disparity map between the synthesized image and the original right image

To create those the fact that the scrolling of objects is a linear operation is used, and the translated depth map is multiplied by a constant, which depends on the considered position:

- Disparity towards left is multiplied by \( \frac{p}{d} \)
- Disparity towards right is multiplied by \( 1 - \frac{p}{d} \)

Note that it’s also possible to create extrapolated views from this, where the generated view is not in between the two cameras (i.e. \( p < 0 \) or \( p > d \)).
Thanks to the known disparity for every point, retrieving the image data is simple. The corresponding pixel just needs to be read in the original left or right image. However, it is important to determine which image is picked to retrieve the data. Four cases can happen:

1. The data is defined (not occluded) in both images → It does not matter if left image or right image is picked
2. The data is defined only in the left image → Left image has to be picked
3. The data is defined only in the right image → Right image has to be picked
4. The data is defined in neither image → Since the correct value is not defined, erroneous values will appear on the screen in both cases

What is done is that data from left image is picked if it was defined (i.e. if the inpaining algorithm didn’t need to be used when translating the depth map), and the right image is used otherwise (in cases 3 and 4).

Note that with this algorithm, even when $p = d$, the reconstructed image is slightly different from the original right image, as it is made mostly of pixels taken directly from the left image.

Because of a limitation in the disparity map estimation, the disparity can’t be computed towards the edges of the image. Therefore the virtual viewpoint cannot be synthesized for these regions. However the image data close to the edges is used to generate the synthesized view.
Chapter 3

Simulation results

In this chapter the Matlab results of the algorithm are presented. It is verified that the hardware is outputting exactly equivalent results.

3.1 Original images

![Left Image](image1.png) ![Right Image](image2.png)

3.2 Interpolation

Reconstructed image with respectively position $p/d = \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$. Reconstruction with $p/d = 0$ is the same as original left image.
It can be noticed that the best quality images are when the synthesis is close to the existing images, that is when position is close to 0 or $d$. Some wrong pixels are seen in the occluded areas, in backgrounds that are just right of a foreground object, such as the chair. The worst quality is for position $d/2$.

### 3.3 Extrapolation

#### 3.3.1 Virtual viewpoint located right of the right camera

Reconstructed images for position $p/d = \frac{5}{4}, 3/2$
It can be noticed that the occluded areas, in the background of foreground objects, starts to contain strongly erroneous data, and this increases as the virtual viewpoint go further right. This is normal, because image data for these areas are absent from both left and right images, thus using the right image data for it is not enough. Displaying correct data would require a guessing algorithm that uses heuristics to attempt a reconstruction of image data.

### 3.3.2 Virtual viewpoint located left of the left camera

Reconstructed images for position $p/d = -\frac{1}{4}, -\frac{1}{2}$

This time the errors are quite different. Zones with incorrect data are more fuzzy, and look more noisy. Parts of the image can be seen appearing several times consecutively, such as this person’s right ear.
This is normal as background that should be shown is captured by neither cameras. This problem can be solved by using a disparity estimation system featuring three cameras, or again, by using a heuristic algorithm that tries to reconstruct inexistent image data.
Chapter 4

Hardware implementation

4.1 Overview

The algorithm for free-view point synthesis is designed to result in an hardware with low resource consumption while maintaining real-time. In this chapter, the details of the real-time hardware implementation are presented.

In order to do so, video memory will not be used to store entire frames of the rendered image (also known as "frame buffers"), as there is no need to do so. Instead, combinational and sequential logic will be used in order to produce the output on the fly. Left image, right image and disparity data are received from an input port, and the synthesized image is output on the Out_Image port.

The top-level input and outputs of the rendering hardware are presented in Figure 4.1: Left image, right image and disparity data are received from respective input ports, and the synthesized image is output on the Out_Image port. Therefore, rendering hardware works with a stream input stream output feature, without using external memory, which makes its portability to different systems easier and its possible ASIC implementation more efficient.

The inputs In_X and In_Y define the pixel position of left image, right image and disparity image. The assumption is that, disparity estimation hardware synchronizes left image pixels, right image pixels and disparity values to send all of them concurrently and regularly, using standard left-to-right and top-to-bottom order for In_X and In_Y values.

Since disparity data originally come with a delay compared to image data but rendering hardware requires concurrent read of the left image, right image and depth image, a delay block is added in front of rendering hardware. This synchronization is made using FIFOs and reading FIFOs with left camera clock. Therefore, rendering hardware is using a single clock, and this is the clock of left camera. The presence of "enable" signal as input and output allows for horizontal and vertical blanking time, as well as arbitrary pauses while sending sequential data.
The output port is not subject to strict timing constraints since the synthesis hardware is not interfaced directly to a display device. The only timing constraint on the input stream is that the vertical blanking is longer than 2 lines, for reasons that will be presented in the next section. The output of the system is written to an output buffer which handles all synchronisation issues. However in this case the timing (including the clock, but also horizontal and vertical blanking times) is orchestrated by the input, and the output follows the input.

This system is colorspace agnostic. Thus the image data can be input and output in any colour space, i.e. RGB, YCbCr, or anything else. For this particular implementation, RGB (8-8-8 bits, 24-bits total) representation is used, but this could be changed without any problems if needed.

4.2 Row-based pipeline scheme

4.2.1 Explanation of the concept

The hardware is working on a concept that it will be called a "row-based pipeline". The data is received on channels of left camera pixels, right camera pixels, and disparity values. Several independent transformations are then applied to the data in a pipeline presented in Figure 4.2. Once the data is ready, it is sent to output, which gives a total delay of N lines between the input and output.

Block RAM (BRAM) components are used for storing image data between the steps, in a double buffered fashion. When one buffer is read by a pipeline stage, it is written to by the other stage. As presented in Figure 4.2, the BRAM components can store data for exactly one row, thus 2 BRAMs per row and per channel are used where applicable (currently left, right and disparity channels).

This concept could be extended to any number of simultaneous rows, but in our case a 2 stage pipeline, as shown in Figure 4.2 was enough. In both cases, the first stage has storage for 3 channels (left image, right image and disparity), while the 2nd stage has storage for only one channel (output data).

1 The channels referred to here are not to be confused with colorspace channels such as R, G, B or Y, Cb, Cr
Note that the processing stage of data within a row can also be itself pipelined to increase the maximum possible working frequency. This pipeline scheme is called "column-based pipeline", since it allows to process multiple pixels in same row in parallel in a pipeline scheme. Details of row-based pipeline and column-based pipeline are given in following chapters.

4.2.2 Beginning and end of the frame

The main idea of the row-based pipeline is processing of a row while receiving next rows. For the rendering process of the last two rows, usual expectation from the pipeline is to process them while receiving first 2 rows of next image as an input. However, the camera has vertical blanking time between sending consecutive images. Therefore, completing the rendering process at the beginning of next image causes a discontinuity in the regular timing of the image.

This is solved by processing the last two lines of the image during the camera’s vertical blanking, and by not processing any data during the first 2 lines of an image. During camera’s blanking time, the
input X and Y coordinates are not valid, and should not be used. Instead of using the input X and Y coordinates during the process, our own set of X and Y counters are added. These counters are independent, but “follow” the input X and Y values by obeying their timings.

While the input is orchestrating the first pipeline stage, X and Y counters are orchestrating the following pipeline stages. Whenever a full line has been received on the input side (meaning that the 1st pipeline stage has finished processing that line), our counters starts to count, processing the same line in the 2nd pipeline stage. The output stage is using the same counter, with an offset added to the Y counter to compensate for the delay the pipeline induces.

At the end of the frame, when the input stage stops to process any data as vertical blanking has started for it, our counters will continue to count until the whole pipeline is empty (in this case for 2 additional lines). The already-empty pipeline stages are processing dummy data but this is not a problem, as long as the dummy data is not sent to the output with the enable signal.

### 4.2.3 Scheduling

As stated before and as shown in Figure 4.3, the process of synthesising an image is made of two fundamental steps, the first step is split in two smaller steps:

1. Generating translated depth map
   
   (a) Translate the pixels by their own disparity (multiplied by the position $p$)
   
   (b) Fill the holes with the neighbour disparity (following the direction of the translation)

2. Generating image data using the translated depth map

### 4.2.4 Left to right and right to left processing

When $p \geq 0$ (which is normally the case), the inpainting step on disparity map is happening from right to left. For this reason inpainting have to work “backwards” in the scanline (which is sent left-to-right by the input device). The row-based pipeline is used in order to allow the processing to go backwards within a line by decreasing the processed column value from X resolution downto 0. The row processing based pipeline stages are given below:

1. Receive disparity and image data (normal order)

2. Reconstruct shifted disparity, synthesise image (backwards)

3. Send data to output (normal order)

In fact, only stage 2 is doing main processing of the rendering, but stages 1 and 3 are there for reordering purposes. In order to count from right to left by decreasing the column values, all the bits of the horizontal counter are simply toggled.
How those steps are translated in a row-based pipeline and how BRAMs is used to retain data between is shown with Figure 4.4. A pipeline of three stages can be seen. The first stage simply records data from the input. The second stage, working in reverse order, uses disparity data and picks which column of the images is to be used for this pixel, and also decides between left and right image to pick the pixel. Finally the last stage sends data in the conventional left-to-right order.

### 4.3 Notes about the schematics and figures

For simplification reasons, all schematics and figures drawn in this chapter show the idea of the circuit, and not the real circuit as it is synthesized in the FPGA. The exact circuit is described in the Verilog hardware description language, and drawing an exact equivalent in the form of a schematic would be harder to understand rather than a simplified schematic that just shows the concept.

Many operations are actually pipelined (horizontally), so that a calculation’s result is ready only several
clock cycles after it has started. For this reason, in the real circuit registered (delayed) version of the signals are used when appropriate. Because the calculations on each pixel are identical and independent, the column-based pipeline introduces almost no problems, as opposed as, for example, a processor’s pipeline.

During the calculations of various steps, whenever precision lower than integer was required, fixed-point representation is used in order to use only integer hardware. Two’s complement signed representation is also used whenever applicable. Lower (sub-integer precision) bits are dropped as soon as they are not needed anymore in the computation path.

For instance, the position $p_d$ is typically between 0 and 1, although it can also overflow slightly from this range without problem in some cases. The number of bits used for the fixed point representation of position is parametrizable. The default is 3 bits for the fractional part and 2 bits for the integer part, allowing to represent numbers between -2 and almost +2 with steps of $\frac{1}{8}$, because representing all positions between -1 and +1 inclusive was needed, to easily adapt the explained two camera based rendering hardware to a 3 camera disparity estimation hardware.

The existing processing scheme between the BRAM stages in Figure 4.4 is implemented in two different methods. Those two methods provides the same output results. Their architectures are explained in section 4.4 and section 4.5 Moreover, comparison between the approaches is presented.
4.4 First approach - Depth map shifting in a single step

At first, in order to reduce the size of the row-based pipeline to the strict minimum, a hardware implementation that merges the shifting and inpainting steps in a single bigger step was implemented.

4.4.1 Shifting and inpainting in one step

In order to create this "single bigger step", the work is done on each pixel individually. Let \( x \) be the current X position in the image, and \( t \) be the displacement between any other pixel on the considered line.

When building depth data for a particular pixel, all pixels of the line need to be looked at, in order to determine which ones satisfy the following equation for all values of \( t \) (i.e. which pixels would "map" to the considered pixel).

\[
\left| \frac{p}{d} \cdot \text{disparity}[x + t] - t \right| \leq \frac{1}{2}
\]

In order to help how this is possible some examples are drawn in Figure 4.5. Examples are given for \( \frac{p}{d} = 1 \) (a depth map is generated for the position of the right camera from a depth map aligned with the left camera), and \( \frac{p}{d} = \frac{1}{2} \) (a depth map is generated halfway between cameras).

The boxes represent pixel locations. The numbers inside the boxes represent disparity values. Arrows represents, where the disparity values moves during the translation process.

![Figure 4.5: Examples of generating translated depth in a single step](image-url)
In the case $\frac{D}{d} = 1$, all pixels moves their disparity values to the right (which is logical, because it’s the definition of the disparity itself). Every pixel which is $t$ pixels on the right of the current pixel has to be looked at if the disparity is equal to $t$.

In the case $\frac{D}{d} = \frac{1}{2}$, all pixels moves half of their disparity values to the right (rounded to the nearest integer). Every pixel which is $t$ pixels on the right has to be looked for if the disparity is equal to $t/2$.

When none of the considered pixels maps to the determined pixel, inpainting with a neighbour known value is done. This is the main reason for using reordering scheme (going right-to-left instead of the more usual left-to-right). Now the disparity value of the last pixel just need to be stored in a register, and this value needs to be used when its value is not updated which is the case where inpainting is needed.

There can be more than one disparity values which satisfy the equation above. In this case, the pixel of maximum disparity (that is the one which is closer to the camera) is taken. If no pixel satisfy the equation, then the inpainting should be performed.

One major problem with this approach is that, for a resolution of 1024 horizontal pixels, if the disparity range is 256, 256 values should be checked in parallel, or if the disparity range is 512, 512 values should be checked in parallel. This high parallel processing increases hardware resource consumption. Moreover, if there are not close objects, hardware performs unnecessary comparisons for high disparity values.

Since the range of viewing is limited, this fact can be used to limit the search area within a single scan line to a much smaller value. In the case where $|\frac{D}{d}| \leq 1$, it is known the multiplied value is smaller or equal than 128, as the multiply factor cannot be greater than 1. This means the history buffer can be restricted to 129 values, as the disparity ranges from 0 to 128 with the disparity estimation circuit. If the disparity range of the disparity estimation circuit is above 128 and below 256, then rendering hardware should be re-synthesized to support 256 disparity range.

The hardware to realise this processing scheme is shown in Figure 4.6. This is handled with a shift register (“history buffer”) of the size of the comparison window. It will automatically remember the disparity for the last matched pixels (regardless if going left to right or right to left) which is needed for inpainting.

The values are pre-multiplied with left and right position difference (respectively $\frac{D}{d}$ for left and $1 - \frac{D}{d}$ for right) before entering in the history buffer, so that they can be compared directly. If any value in pre-multiplied disparity history is equal to it’s position in the history buffer, it should be used.

If more than one value has to be used, its maximum one should be used, which is done by a special large-scale maximum-finder circuit. When no value can be used, then this is equivalent to a hole that has to be filled, a register is simply used on output that repeats the last known value.

Once the disparity value of a pixel for the synthesized image is known, the result is used to reconstruct YCbCr or RGB based synthesized image. This process is explained in Section 4.6.
4.4.2 Maximum circuit

The maximum disparity value out of matching values needs to be found relatively quickly. Indeed, most of them will be zero, because they don’t satisfy the equation to be displayed, but this doesn’t affect the circuit. In order to make a reasonable critical path, a tree like structure is used, which finds the maximum of $2^n$ values, as seen in Figure 4.7. In practice, the maximum circuit is pipelined in order to get a shorter critical path.

![Tree structure for finding the maximum of many values](image)

**Figure 4.7:** Tree structure for finding the maximum of many values
4.4.3 Conclusion over the first approach

Going this direction for shifting the disparity map turned to be an inefficient decision. The hardware doing so is bloated and complex. Effectively, the history buffer is long, and has to become even longer if range of position \( p \) or the range of disparity values is increased. The history buffer has to be doubled for left and right channels. A maximum circuit for hundreds of values has to be used, even though typically only 2 or 3 values will be non-zero at a given time.

The idea of the presented first approach came before the idea of a more proper row-based pipeline idea which is presented in Figure 4.2. By using the row-based pipeline in a smart way, shifting the disparity data using multiple separate steps is possible, in a way that is much closer to the original algorithm. This is not only more intuitive, making the hardware easier to debug and maintain, but also reduces hardware resources in general.

4.5 Second approach - Depth map shifting and inpainting in two steps

The second approach that is used to generate a shifted disparity map is much simpler, and it is more expandable. Moreover, it made extrapolation synthesis relatively easy, since image synthesis for the left of the left camera \((p < 0)\) or right of the right camera \((p > 0)\) does not have overflow problem for this approach.

The approach is simpler because it is closer to the original algorithm that was done with a software point of view, there is no need for a maximum circuit, and therefore the “horizontal pipelines” are shorter and easier to debug.

The idea of the second approach is to re-order the 3 steps of the row-based pipeline as follows:

1. Receive image data, build shifted disparity
2. Read back disparity backwards, apply inpainting, synthesize image
3. Send data to output (normal order)

The last part of the second stage and the third stage are absolutely identical to the previous approach, and will be covered in the next section. The schematic of the simpler disparity shift hardware is shown in Figure 4.8.

In order to create the shifted disparity map, the original disparity map is simply never recorded as it is. Instead they are shift directly in the 1st row-based pipeline stage before writing to BRAM. This creates a shifted disparity map without using the complex hardware mentioned above. However two problems arise: First occluded pixels will contain invalid data previously written here instead of valid data. Second, multiple writes to the same place should be handled in order to make sure the maximum value prevails.
To fix the first problem with holes, validity bits are used, handled by two arrays of flip flops. Each array is associated with a BRAM value, in a double-buffered fashion as already seen. When input hardware starts to send any scanline, all the flip-flops of the corresponding BRAM are cleared. That means all values becomes invalid. Since a whole line of BRAM cannot be physically changed at the same time, but only a single byte at a time, this solution can be considered as an efficient method to handle this problem.

Each time a valid value is written to BRAM, the corresponding valid bit is set. Then, during the next pipeline stage, it is known if a read back value is a valid shifted-disparity value for the current scanline. If it is invalid, the previously encountered valid data, remembered with the help of a register, is used to fill it, just like before.

In order to enforce that only the maximum value is written to a pixel, the cases of going left-to-right \((p > 0)\) or going right-to-left \((p < 0)\) should be separated.

If \(p > 0\), the shift is always made towards the left (negative offset), and since the shift is proportional to the value itself, a greater value will always come later, overwriting a smaller value. Therefore, at the end of the line, the values in valid disparity pixels will be the maximum written one automatically, there is no need to add any hardware.

However if \(p < 0\), the situation is the opposite. Pixels are shifted towards the right, and it is known that a second write is always smaller than a previous write. Only the first write should be effective, and all further writes should be ignored. In order to have this feature, the validity bit can again be used. If this bit is set, BRAM writes are disabled, only the first (the highest) value is written effectively.
Finally, there is no uncertainty for the length of the "history buffer" like seen before for the first approach, since the range of shift for individual disparity pixels is simply determined by the # of bits in the adder on the address lines. The resulting shifted disparity map is identical to the result of the first approach.

### 4.6 Image construction and output

The last pipeline stage uses the shifted disparity data in order to construct the synthesized image. Afterwards, the output synthesized image is transferred in real time. The schematic of the image construction hardware is shown in Figure 4.9.

![Disparity shift hardware](image)

**Figure 4.9: Schematic of the image construction pipeline stage**

Thanks to the shifted disparity map, which was calculated in the previous stage, this part of the system is quite simple. As seen before, the counter goes either left-to-right or right-to-left depending on the viewpoint, and the disparity data multiplied respectively for left and right viewpoints is added to the counter in order to feed the address lines of BRAM. This way, the corresponding pixel on both left and right is read, and the appropriate one is used depending of whenever the image is occluded or not.
Chapter 5

Future Improvements

Although satisfying results were obtained, there is room for many improvements, as synthesized images are usually quite noisy.

5.1 Improving disparity estimation

5.1.1 The problem of occlusion

Currently, the main source of error in rendered images is errors in the disparity estimation hardware. The key to improve rendered images is to improve disparity estimation itself.

The main problem is that, currently the dual camera disparity estimation system does not detect occlusion on left-edges of objects. The problem is illustrated in Figure 5.1. For this simplified example, a blue object (disparity = 2) is seen in front of a red object (disparity = 1).

As a reminder, the Disparity Estimation System can find the corresponding pixel in the right image for a given pixel in the left image. It is possible doing so for pixels a-d, A-D, J and K. However, it is impossible to find a corresponding pixel in the right image for pixel E, because it is behind the blue object. Therefore, this region is occluded, and the disparity is undefined.

However, the Disparity Estimation system does not give this information and sends invalid data instead. This is problematic, since in the rendering hardware, it is assumed that disparity data is valid. In order to fix this problem, the invalid value (noted XX on Figure 5.1) should somehow be replaced with a ‘1’, because even though disparity is technically undefined, this value is what is needed for a depth map.
5.1.2 Improvement by detecting occlusion

The occlusion problem detected in the previous section is extremely similar to the occlusion problem seen in Figure 2.4 when shifting the disparity map towards the right view. For pixel I of the right image, there is no corresponding pixel in the left image. We used the approach to copy disparity of pixel J by inpainting. Knowing that this pixel only exists in the right image, data from the right image should be used when reconstructing data.

The problem for pixel E is an exact mirror of the same problem, the only difference is that this occlusion already appears before shifting the disparity map. A solution would be to have the Disparity Estimation System detect this case. If it has too much a hard time to find a corresponding pixel in the right image, it should mark the disparity data as "invalid". The invalid data can then be corrected by using inpainting.

Detecting invalid disparity data because of occlusion could be added by either the Disparity Estimation System, or by the synthesis hardware. However, it would probably be beneficial to do this in the Disparity Estimation System, as it would also create a more accurate depth map for other potential applications.
5.1.3 Improve with a third camera

The occlusion problem can be solved by adding a third camera, and research continues on the topic of using a 3rd camera to improve the disparity estimation significantly.

By estimating the disparity from both left and right camera in the reference of a center camera, the right camera will give proper disparity for right edges and the left camera will give proper disparity for left edge. Values from the appropriate disparity map can then always be taken to create a new depth map, with correct edges on both sides.

Adding a third camera allows for other improvements than only the depth map. This is further discussed in Figure 5.2.

5.1.4 Filtering

Another possible improvement is to apply a filter to the depth map. Filtering will not fix anything if the disparity estimation has large errors (such as left-edge occlusion), but it can reduce noise and possibly other kind of small errors.

Digital filters are usually present in the form of :

- **Finite impulse filters (FIR)** which uses data from a window of unfiltered values to generate a sample. They usually require a large number of coefficients.

- **Infinite impulse filters (IIR)** which uses the combination of a window of unfiltered values and a recursive algorithm based on previous filtered samples to generate a filtered sample. They usually require a small number of coefficients.

- **Non-linear filters** which comes in various forms and uses various heuristic techniques to filter out unwanted data in well-defined cases.

Because an image is 2-dimentional data, filters can be applied in horizontal and vertical directions. The amount of coefficients (or the width of the window of unfiltered samples) is determinant for the hardware complexity of the filter.

Filtering horizontally can be easily added and requires little additional hardware. Namely a window of unfiltered samples is implemented using a shift register. However filtering vertically requires additional stages in the row-based pipeline.

If only horizontal filtering is done, then there is almost no limits and can pick almost any kind of filter. For filtering in the vertical direction, however, there is a compromise between the size of the row-based pipeline (BRAM usage) and the filter’s quality.

Non-linear filtering is probably to be preferred for this application, as the goal is to preserve sharp edges in disparity, while removing noise.
5.2 Three cameras

Since it is planned to add a third camera to improve depth estimation, it makes no sense to ignore it’s image data completely. The third camera should be used as well in the image synthesis stage: Instead of selecting a pixel between the left or right camera, selection should be made between left, centre and right. This time the disparity map is based on the centre camera (instead of left), as shown in Figure 5.2.

A simple algorithm is to select centre image when no occlusion appears. When occlusion appears, left camera is used when position \( p < 0 \) and right camera is used when position \( p > 0 \). It is also possible to implement more complicated arbitration between the cameras, but this is likely to be not necessary. The new image synthesis pipeline stage would look like Figure 5.3.

![Figure 5.2: Overview of the system with an additional camera](image)

![Figure 5.3: Schematic of image generation stage with three camera](image)
5.3 Rectification

The idea behind rectification is to correct the imperfect alignment of the cameras and lens distortions. The rectification hardware corrects the images acquired by the cameras so that every pixel is on the same epipolar line in both images. This task is not performed currently, but the research proceeds on the integration of the rectification hardware to the system.

There are 2 main advantages of implementing rectification. First, there is no need to align the cameras manually every time the system is transported, thus the system is more portable. Moreover, without rectification, even a small miscalibration can result in an inaccurate disparity map. Rectification improves disparity estimation significantly, as it removes the effect of misalignment from the synthesized image.

5.4 Moving in other directions

The virtual viewpoint can only move horizontally using the current rendering hardware. It could be also possible to move this point in the vertical or transversal directions. Rotations among the 3 axis could also be implemented. Combinations of those 6 possible transformations makes up all movements that can be done in a 3D space environment, as shown in Figure 5.4.

![Figure 5.4: Orthogonal coordinates system as seen on the screen](image)

5.4.1 Vertical movement

Synthesizing images for vertical viewpoint movement would be in concept very similar to horizontal. The main problem with moving the virtual viewpoint vertically is that data required to synthesize parts of the image is not acquired by the cameras. A solution could be adding another camera in the vertical direction, which complicates the system and makes it even harder to setup. Another solution is to estimate the data required by some heuristic algorithm.

If movement of the free viewpoint can be done both horizontally and vertically, it is possible to combine both to move freely in a plane.
5.4.2  Transversal movement

Moving the virtual viewpoint in the transversal direction would also be interesting. It would be the equivalent of zooming into or out of the image, taking the disparity of the objects into account. Moving in this direction is quite different from moving horizontally. The problem needs to be studied again from a geometrical viewpoint. Moreover, a row-based pipeline is not sufficient to handle the transformation; so a full frame buffer with an external memory should be used.

As long as the movement is restricted to going forward (zooming), and that going backwards (unzooming) is not allowed, then this can be done without having to run any heuristic algorithm to estimate image data. Thanks to this fact, transversal movement is probably a more interesting direction than vertical movement.

Combining transversal and horizontal movements is relatively easier using two cameras. This combination may be also more useful for mimicking human depth perception.

5.4.3  Rotation

The angle of view can be rotated in 3 directions: vertical (around the x-axis), horizontal (around the y-axis) and transversal (around the z-axis).

Horizontal and vertical rotations correspond to moving the eyes or the head up/down and left/right, which are natural movements. Done on an image captured by a camera, they are equivalent to a two-dimensional scrolling of the image. This can be implemented trivially depending on the application, but occluded zones appears immediately as blank rows and columns in the border of the image, limiting the interest of such an effect to very small viewing angles.

In order to increase the viewing angle horizontally, it is possible to rearrange the cameras to be on the edge of a circle to follow the rotation instead of parallel arrangement. However doing so requires major adaptations for the disparity estimation algorithm.

Transversal rotation is equivalent to a 2-dimensional rotation of the image. As this is the least natural rotation for an human being, thus this not an extremely interesting effect to do.

5.5  Virtual reality

A virtual reality head mounted display (HMD), such as the one shown in Figure 5.5 could be used to provide an interesting user experience. The images required for the 3D view are synthesized separately by using viewpoints at 2 finely calculated points. The illusion of transversal movement of the eyes or the head can easily be created by changing the location of both viewpoints.

Simulating vertical and horizontal rotations would also be interesting. However, the viewing angles and positions is of course limited by the position of the cameras that are filming the scene.
5.6 Conclusion

The final goal of the project is to achieve an efficient system as presented in Figure 5.6. At the time of writing this thesis this was not ready yet. Just at the last day of thesis submission, the first real-time hardware implementation results for the 3 camera system is obtained. Therefore, implementation results for 3 camera system is not presented in this thesis. Much better results for rendering is achieved, as all the additional components helps in synthesizing better quality images.

![Figure 5.6: System that will be built in the future](image)

Further improvement can be achieved by working more on three camera system. Still, the hardware design and its results in this thesis present that, efficient real-time rendering hardware can be implemented using two cameras and rendering hardware is scalable to multiple cameras.

Research about image synthesis has been made by many researchers. But we do not rely on any kind of pre-rendering technique. The image acquisition, depth estimation and image synthesis are all done in real time, and with negligible latency. Future work will be able to do more 3D transformation and head tracking, creating impressive immersion systems using 3D technology for multiple applications.

Moreover, implementing and testing this system with 3 cameras is the most important novelty that come up with this work. We can obtain disparity map with ordinary cameras, instead of infra-red cameras.
with true distance measurement. This makes the system affordable financially, as the customer only has to buy two or three ordinary cameras and a hardware chip that combined disparity estimation and rendering. This chip can be attractive for consumer electronics since it does not need any external memory, thus, it can work standalone if the chip is interfaced directly or indirectly with two cameras.
Bibliography


4. "View generation with 3D warping using depth information for FTV", Yuji Mori & al., Department of Information Electronics, Graduate School of Engineering Nagoya University, 2008

5. "Acquisition, Compression and Rendering of Depth and Texture for Multi-View Video", Yannick Morvan, Technische Universiteit Eindhoven, 2009


8. "Dynamically adaptive real-time disparity estimation hardware using iterative refinement", Abdulkadir Akin & al., Integration, the VLSI Journal, EPFL, 2013