國 立 交 通 大 學

電子工程學系　電子研究所碩士班

碩 士 論 文

視點合成器分析與設計

Analysis and Design of a View Synthesis Engine

研究生：洪瑩蓉

指導教授：張添烜

中華民國　九十九年　九月

視點合成器分析與設計

Analysis and Design of a View Synthesis Engine

研 究 生: 洪瑩蓉　　　Student: Ying-Rung Horng
指導教授: 張添烜　博士　　　Advisor: Tian-Sheuan Chang

國　立　交　通　大　學
電子工程學系　電子研究所碩士班
碩　士　論　文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical Engineering and Computer Engineearing
National Chiao Tung University
in Partial Fulfillment of Requirements
for the Degree of Master
In
Electronics Engineering
September 2010
Hsinchu, Taiwan, Republic of China

中華民國　九十九年　九月

視點合成器分析與設計

研究生: 洪瑩蓉　　　　　　　　指導教授: 張添烜博士

國立交通大學
電子工程學系　電子研究所碩士班

摘要

自由視點電視系統（FTV)可以產生自由視點視訊作為新一代的顯示應用。在自由視點電視系統之中，視點合成(view synthesis)是其中重要的一部分，作為接收端的視點內差以及重建。MPEG-FTV 所發展提供的視點合成演算法（view synthesis reference software, VSRS)使用了兩個參考視點的視訊及其深度圖來合成虛擬視點的視訊。為了達到即時處理高解析度視訊的目標，我們根據 VSRS 演算法設計一個高規格視點合成器的積體電路。然而，尤其是高解析度的視訊，在視點之間不規則的對應關係使得硬體設計時遇到資料控制複雜、晶片記憶體需求高以及高頻寬等難題。

在這篇論文中，針對上述的挑戰我們提出一個從像素等級、列等級、到畫框等級的階層式管線化架構。在畫框等級管線化中，以兩個畫框等級分別進行深度貼圖(depth mapping)與圖像貼圖(texture mapping)，並將第一級深度貼圖所產生之虛擬視點的深度圖寫到外部記憶體中來減少內部晶片記憶體的需求。在列等級管線化中，我們利用環形 FIFO(Circular FIFO)以及列層級緩衝的架構來改進對外部記憶體資料傳輸的效率。最後，所有的運算元都以像素等級的管線化來達到每秒 0.5 像素分別處理左、右兩個參考視點的生產效率。

在聯電 90 奈米的製程下，此設計的硬體消耗為 268.5K 個邏輯閘、單埠及雙埠晶片記憶體各需 56.9K 及 12.5K。此外，在實驗結果的客觀評比中，我們提出的硬體化設計相比於原本的 VSRS 演算法可以達到無 PSNR 下降的品質。

Analysis and Design of a View Synthesis Engine

Student: Ying-Rung Horng Advisor: Dr. Tian-Sheuan Chang

Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University

Abstract

The free view-point television (FTV) system is a developing innovative system that can generate free view-point video for the new trend of display application. In the FTV system, the view synthesis is one of the most important components in the receiver-side for view interpolation and reconstruction. MPGE-FTV forum proposed view synthesis reference software (VSRS) that uses two-view videos and the corresponding depth maps to synthesize the virtual-view video. To achieve the demand of synthesizing high quality videos in real-time, we implemented a view synthesis engine in VLSI design based on the VSRS algorithm. However, the irregular mapped positions between views result in complicated data control, high internal storage, and high external bandwidth in hardware implementation, especially for high resolution videos.

To address the design challenges, in this thesis, a hierarchical pipelining architecture from the frame-level, column-level, to pixel-level is proposed. In the frame-level pipelining, two stages are installed for the depth mapping and texture mapping to store temporary warped maps in external memory and reduce internal memory. In the column-level pipelining, the circular FIFO buffering architecture and column-level buffering are proposed to improve the efficiency of external data access. In the pixel-level pipelining, all the computations are parallelized to achieve the throughput of 0.5 pixels per cycle.

With the UMC 90nm CMOS technology, this design consumes 268.5K gate counts, 56.9KB and 12.5KB memory space for on-port and two-ports SRAMs. In the objective evaluation, our hardware implementation can generate the virtual view without quality drop in PSNR, compared to the original results of VSRS.

致謝

感謝我的指導教授，張添烜博士，教導我正確的研究態度和方法，指導我的研究方向並幫助我發現問題、解決問題。

感謝口試委員們，交大資工蔡淳仁教授及台大電機簡韶逸教授。謝謝教授們抽空前來指導並提供許多寶貴意見和想法，讓我的論文更加完整。

感謝實驗室的夥伴們。特別感謝從進實驗室開始到畢業期間引導我的曾宇晟學長，指導我從找資料、寫程式、電路設計到撰寫整理報告的完整研究流程，讓我受益良多。謝謝李國龍學長、王國振學長提供我研究和實驗的經驗和協助。謝謝學弟們吳英佑、邱亮齊、溫孟勳、曹克嘉幫忙做實驗並總是讓實驗室充滿活力氣氛。謝謝許博雄、陳奕均、陳宥辰、廖元歆，強者同儕讓我在修課和研究上都獲得很多幫助。感謝實驗室的所有成員。

最後，感謝我的家人：爸爸阿池、媽媽阿玉、大姐姐、小暘和小淇，謝謝你們包容我這段期間的暗黑爆走陰晴不定喜怒無常，謝謝你們所有的加油打氣和無盡的愛；以及謝謝かめ今年的演唱會，讓我得以抓住學生生涯最後的瘋狂的青春尾巴。

謹將本論文獻給所有愛我以及我愛的人。

Contents

List of Figures

List of tables

Chapter 1

Introduction

1.1 Background

In these years, the 3D-TV system becomes more popular since it can generate stereoscopic views, and raise a drastic revolution in human vision entertainment. Another new developing system, free-viewpoint TV (FTV), can generate arbitrary views chosen by user. These two systems both adopt the depth image-based rendering (DIBR) method [1],[2], which uses depth maps to render novel views, and does not need the real geometry of scene since the depth maps only indicate the mapping relation between views.

In the frameworks of the both systems for broadcast TV, the depth maps are estimated by the sender-side because of its high-complexity effort; after encoding, transmitting and decoding by multi-view video coding (MVC), the novel views are synthesized by the receiver-side.

1.2 Motivation

The recent research of view synthesis focused on algorithm development and implemented by software programming. However, for practical applications, the receiver-side demands a real-time view synthesis engine for displays. Furthermore, for the increasing resolution demands on consumers, the frame size of HD is a trend, so that the challenges on bandwidth or memory usage are increased. Motivated by the real-time speed and high resolution requirement, the thesis proposes a view synthesis engine in application specific integrated circuit (ASIC) design, which supports the real-time demand for HD videos.

1.3 Thesis organization

In this thesis, Chapter 2 concentrates on the basic background of video-plus-depth format and depth-image based rendering (DIBR), which is the most common idea of synthesizing images using depth map. Two different configurations of multi-cameras for DIBR, 3D warping and horizontal shift, are also briefly demonstrated in this chapter.

Chapter 3 introduces the algorithm of view synthesis reference software (VSRS) provided by MPEG-FTV. The VSRS is a two-view based rendering method, using the video-plus depth format for each view. The algorithm uses the 3D warping for general camera mode, and the horizontal shift for 1D camera mode. In addition, the 3D warping needs the homography transform. The main flow of VSRS contains depth mapping, post-filtering, texture mapping, two-view blending and hole-filling. Since the last hole-filling affects the synthesis quality, different hole-filling methods are discussed in this chapter.

Our hardware implementation is for the 3D warping mode of VSRS. Chapter 4 focuses on the architecture design of VSRS, and addresses the problem in 3D warping that the mapped index is random over the whole frame. Bandwidth usage and internal memory size are also the concentrated points in this chapter. Finally, this chapter proposes an efficient architecture with the frame-level pipelining in global view and the hierarchical column-level pipelining in local view.

Chapter 5 describes the detail implementation methods based on the designed architecture. Final implementation results contain gate-count, internal memory usage and the overall performance in peak-signal-to-noise ratio (PSNR).

1.4 Contribution

The major contribution in this thesis includes:

1. We analyzed the hardware cost of VSRS algorithm for the general setting case of cameras, in which the cameras may with rotation such that they are not in the same baseline.
2. We modified the hole-filling method in VSRS by using the simple bi-linear interpolation method and analyzed the bandwidth and memory usage in different hardware design approaches.
3. We implemented the whole VSRS algorithm as ASIC design for the real-time, high-resolution requirements.

Chapter 2

Related work

View synthesis is an image rendering method in the application of depth map. This chapter first introduces the concept of stereo vision and depth estimation method associated with view synthesis. Then the data format of video-plus-depth applied in the 3D-TV and FTV system is described. Finally, the most general view synthesis method, depth-image based rendering (DIBR) is presented.

2.1 Stereo vision and video-plus-depth concept

Human feels 3D visual perception because the scenes seen by left eye and right eye are with horizontal difference. The difference is called screen parallax values or the disparity that brain can interpret it as 3D visual perception as shown in Fig. 2-1. This disparity X_R-X_L can further be transformed to depth Z based on the inverse proportional relationship considering the baseline B and focal length f,

$$Z = \frac{Bf}{X_R - X_L}$$

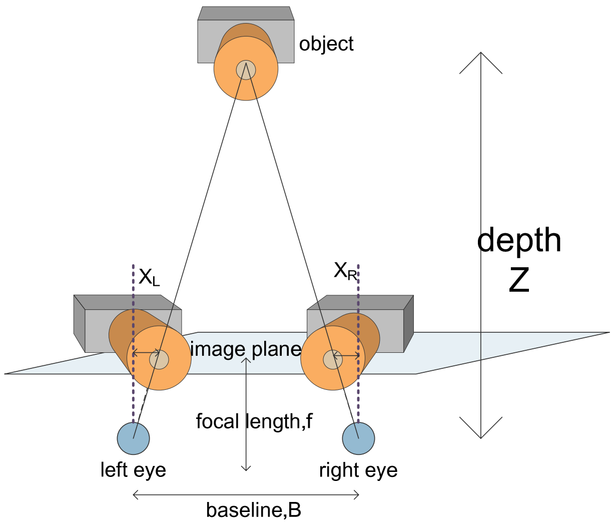


Fig. 2-1 Relationship of depth and disparity for 3D visual perception

The depth map stores the depth value sampled in 8 bits and has variety of applications, such as 3D display, 3D interactive system, or multi-view video etc. In fact, depth map can be obtained by many methods such as using time-of-flight camera (TOF camera), structure from motion algorithm or stereo matching algorithm etc. Nowadays, in both 3D-TV system [1] and FTV system [2], the video and its corresponding depth maps are encoded and transmitted by the sender-side, and they are decoded by the receiver-side to generate novel views or stereoscopic views. The data format in these systems is called video-plus-depth format. In the video-plus-depth format, the sender-side is mainly for video capturing, rectifying and depth estimating, and the receiver-side can do view synthesis flexibly and adaptively for different display needs by the reference of the depth maps [3].

2.2 Depth-image-based rendering

2.2.1 3D warping

The depth-image-based rendering (DIBR) is an image-rendering technique using depth maps for virtual view synthesis. This rendering is by 3D warping model considering the provided known by-pixel depth. In 3D world-coordinate as shown in Fig. 2-2 (a), a point *(X, Y, Z)* can be mapped to a 2D point *(u, v)* in the image plane of a camera. In Eq. (2-1), this projection is performed by the projection matrix **P**, which can be decomposed to the camera intrinsic parameters **K**, the camera rotation matrix **R**, and the translation matrix **T**.

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{P}\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \mathbf{K}\left[\mathbf{R} \quad | -\mathbf{T}\right]\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2-1}$$

$$s = \frac{Z - T_z}{w''}, \text{by letting} \left(u'', v'', w''\right)^T = \mathbf{R}^{-1}\mathbf{K}^{-1}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{2-2}$$

Note that the term *s* in Eq. (2-1) is a scalar factor, which depends on the depth *Z* as in Eq. (2-2).

Next we consider the back-projection from a 2D point to 3D word-coordinate. In fact, a 2D point would be mapped to a line in the 3D world-coordinate. That means one pixel will not be back-projected to a unique point in the 3D world-coordinate. Nevertheless, if we have the depth *Z*, a 2D point can be back-projected to a unique point in the 3D world-coordinate through this equation,

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \frac{Z - T_z}{w''} \mathbf{R}^{-1} \mathbf{K}^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} + \mathbf{T} \tag{2-3}$$

Based on the projection of 3D to 2D and the back-projection of 2D to 3D, we can further extend this concept to the mapping relation between multi-views. For the convergent cameras as shown in Fig. 2-2 (b), the camera centers of two views are mapped and converged to the same point Z_C in the 3D world-coordinate. And (a) is an example of multi-view projection. Since the point (u_{Src}, v_{Src}) of Cam_{Src} and the point (u_{Dst}, v_{Dst}) of Cam_{Dst} are projected to the same point (X, Y, Z) in the 3D world-coordinate, these two points have the mapping relation in the following equation,

$$\frac{Z - T_{Z,Dst}}{w''_{Dst}} \begin{pmatrix} u_{Dst} \\ v_{Dst} \\ 1 \end{pmatrix} = \frac{Z - T_{Z,Src}}{w''_{Src}} \mathbf{K}_{Dst} \mathbf{R}_{Dst} \mathbf{R}_{Src}^{-1} \mathbf{K}_{src}^{-1} \begin{pmatrix} u_{Src} \\ v_{Src} \\ 1 \end{pmatrix} + \mathbf{K}_{Dst} \mathbf{R}_{Dst} (\mathbf{T}_{Src} - \mathbf{T}_{Dst}). \tag{2-4}$$

where the index of source view can be warped to destination view with the given Z value and the camera parameters of each camera.

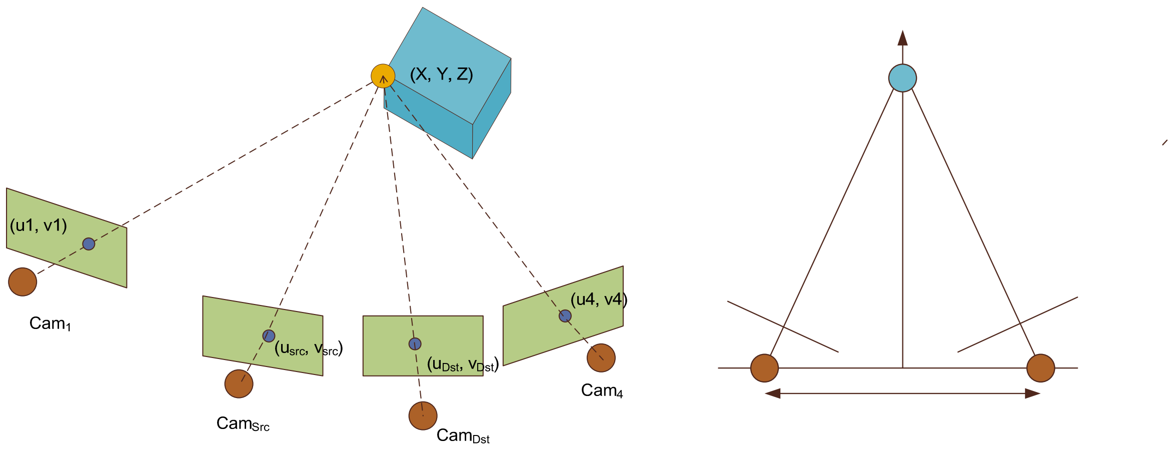


Fig. 2-2 (a) Convergent multi-view cameras; each (u, v) in the image planes can project to the same 3D point(X, Y, Z). (b) Convergent two cameras and the convergence point Z_C. Adapted from [4]

2.2.2 Horizontal shift

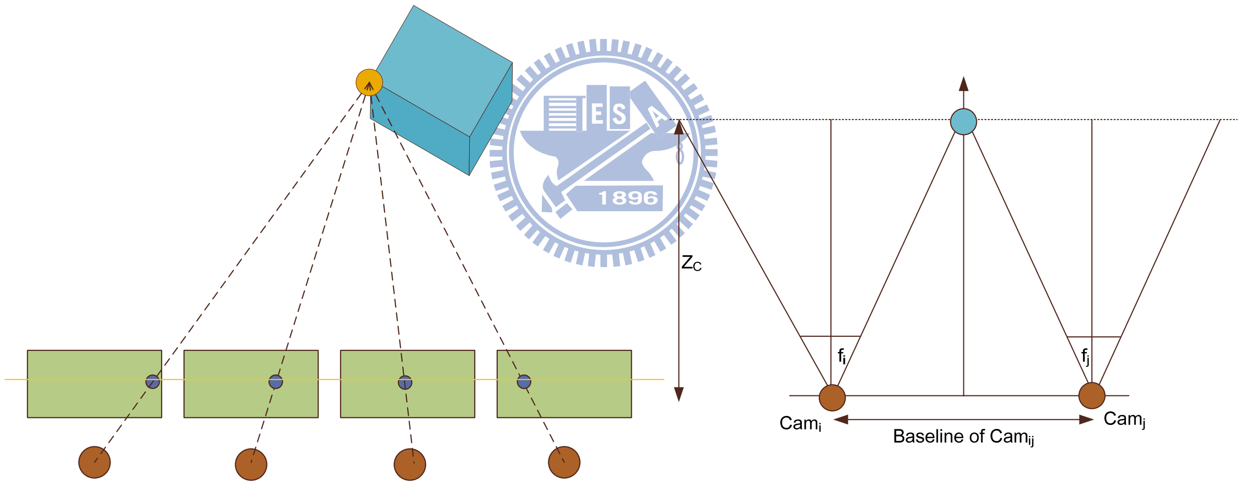


Fig. 2-3 (a) parallel multi-view cameras; each (u, v) in the image planes can project to the same 3D point(X, Y, Z). (b) Parallel two cameras and the convergence distance Z_C. Adapted from [4]

If cameras are all set parallel in a line as shown in Fig. 2-3, the DIBR can be simplified to the mapping method, called horizontal shift [5].

For cameras are in parallel, the camera rotation matrix becomes an identity matrix **I**, and translation vector will only have nonzero element in X dimension as $\mathbf{t}=(T_X, 0, 0)^T$. If the intrinsic matrices of source view and destination view are the same, Eq. (2-4) can be rewrite as

$$u_{Dst} = u_{Src} + \frac{(T_{X,Src} - T_{X,Dst}) f_u}{Z}, \tag{2-5}$$

where f_u is the focal length in the intrinsic matrix. It shows that the mapping index of two views in each image plane only has difference in horizontal direction. In this camera configuration, the process of view synthesis can be performed through a much easier way, which can be horizontal shifting according to the depth value Z.

Chapter 3

Overview of view synthesis reference software algorithm

3.1 Overview

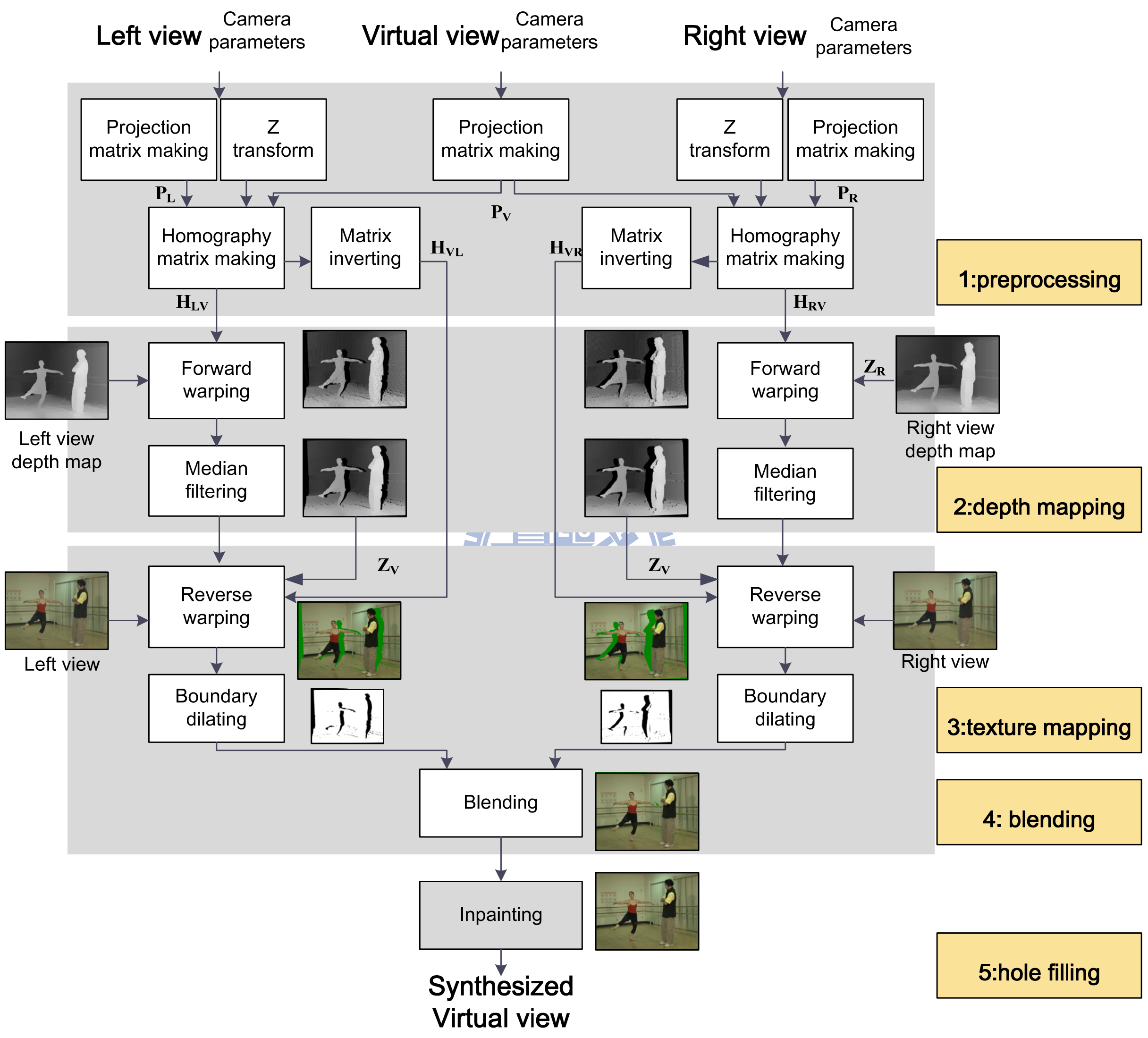


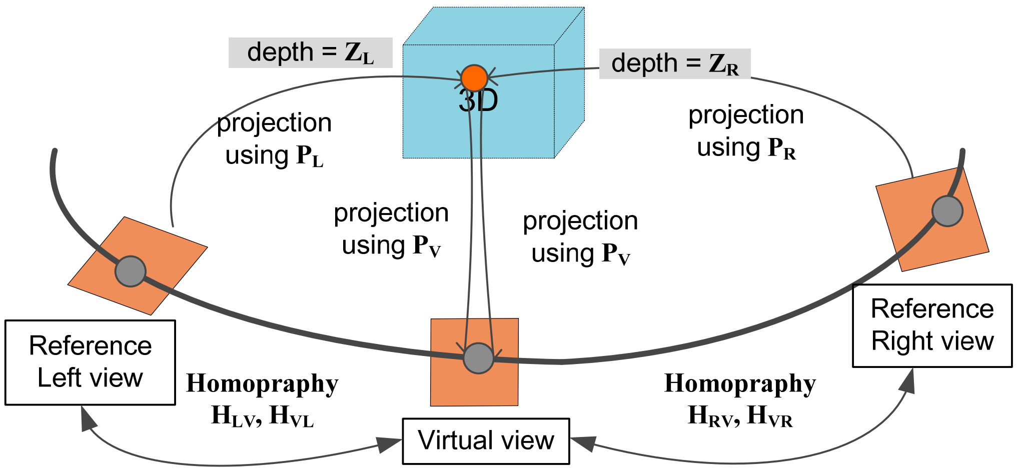
Fig. 3-1 Block diagram of view synthesis algorithm proposed by MPEG-FTV

According to MPEG-FTV, the view synthesis algorithm synthesizes virtual/target/synthesized view based on the two view framework.[6] The algorithm block diagram is Fig. 3-1. First the camera parameters of reference views and synthesized view are used for projection matrix to do 3D image warping, and the projection is further implemented as homography transform will be described in detail in following section. Note that the reference views are usually one in the left and one in the right for high-quality occlusion handling. The depth map at synthesized view is pixel-by-pixel mapped using forward warping from reference view, and then post-filled using median filter. Next, the synthesized depth is reverse warped to reference view for texture mapping. The above processes are adopted for both left view and right view, finally the synthesized texture from left view and right view are blended by occlusion handling and the remaining holes are filled by inpainting method.

3.2 3D image warping

3.2.1 Projection transform and homography transform

This algorithm uses 3D image warping for view synthesis as described in Chapter 2.2 for the video-plus-depth data format. Fig. 3-2 illustrates the process of 3D image warping, where the projection represents the mapping between 3D world coordinate and image coordinate with a a 3x4 matrix. For the example of a point in the left view to its correspondence in the target view, the calculation of warping process contains two projection transform, which are the forward projection using $\mathbf{P}_L$ and the backward projection using $\mathbf{P}_V$. Because of the twice projection and the pixel-by-pixel warping process, the projection transform method has high computational complexity.



depth = $\mathbf{Z_L}$
depth = $\mathbf{Z_R}$
3D
projection using $\mathbf{P_L}$
projection using $\mathbf{P_R}$
projection using $\mathbf{P_V}$
projection using $\mathbf{P_V}$
Reference Left view
Reference Right view
Homopraphy $\mathbf{H_{LV}, H_{VL}}$
Homopraphy $\mathbf{H_{RV}, H_{VR}}$
Virtual view

Fig. 3-2 Warping process using projection or homography transform

On the other hand, the homography method simplifies the warping process, and adopts the homography matrix, which is a 3x3 matrix for the relation between two image planes. For the same example in Fig. 3-2, the calculation of a point from the left view to target view can become only one transform using the homography matrix $\mathbf{H_{LV}}$. Note that one homography matrix is corresponding to one specific depth of Z in the 3D coordinate shown in Fig. 3-3. That is because a homography matrix is deduced from the forward projection and backward projection with fixed depth of Z. Since a depth map is usually represented using a gray-level image, there are 256 homography matrices for 256 depth levels between two views. In addition, because of the efficiency of homograph method, the released software of MPEG-FTV uses homography transform to perform the pixel-by-pixel warping process [5], [8].

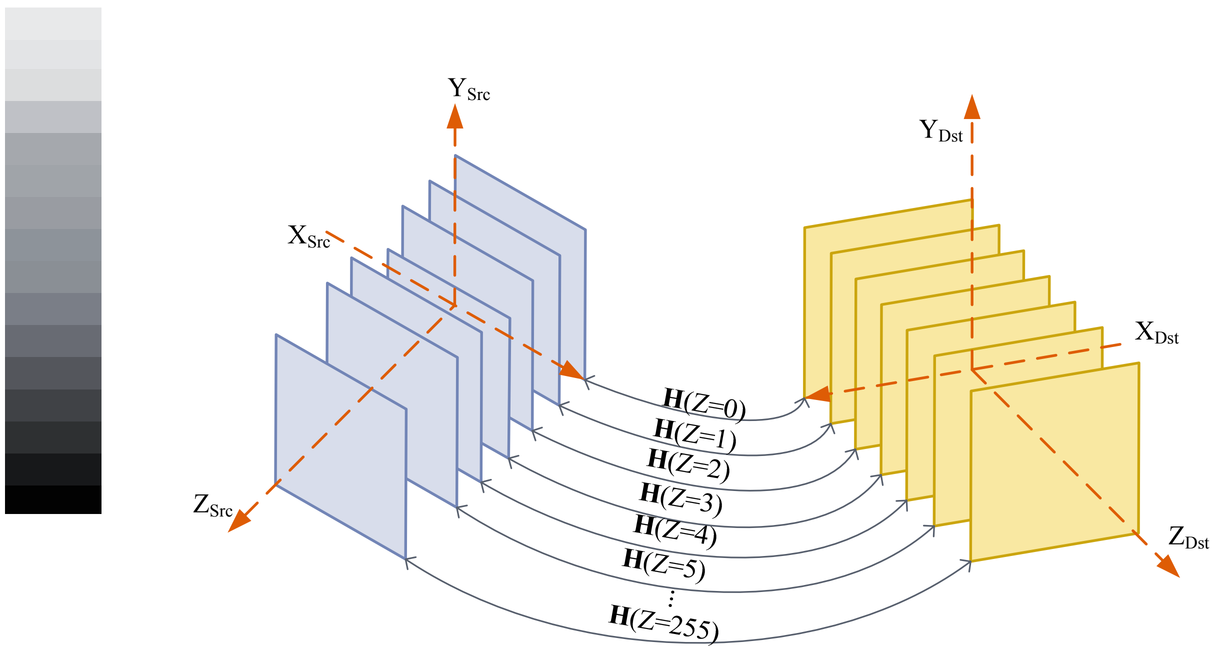


Fig. 3-3 Homography of 256 depth levels between two views

For the homography relation with a specified depth Z, if the two image planes I_{Src} and I_{Dst} have the homography matrix **H**, they should satisfy the transform equation,

$$\mathbf{x}_{Dst} = \mathbf{H}\mathbf{x}_{Src} \tag{3-1}$$

where $\mathbf{x}_{Src}$ is a point with the vector $(u_{Src}, v_{Src}, 1)^T$ on the image plane I_{Src}, and $\mathbf{x}_{Dst}$ is a point with the vector $(u_{Dst}, v_{Dstc}, 1)^T$ on the image plane I_{Dst}. Let **H** be a 3x3 matrix formed by $(h_{00}, h_{01}, h_{02}; h_{10}, h_{11}, h_{12}; h_{20}, h_{21}, h_{22})$. The transform equation Eq. (3-1) can be expanded as Eq. (3-2), and be further rewritten as Eq. (3-3).

$$u_{Dst} = \frac{h_{00}u_{Src} + h_{01}v_{Src} + h_{02}}{h_{20}u_{Src} + h_{21}v_{Src} + h_{22}}, v_{Dst} = \frac{h_{10}u_{Src} + h_{11}v_{Src} + h_{12}}{h_{20}u_{Src} + h_{21}v_{Src} + h_{22}} \tag{3-2}$$

$$\begin{aligned} u_{Src}h_{00} + v_{Src}h_{01} + h_{02} - u_{Dst}u_{Src}h_{20} - u_{Dst}v_{Src}h_{21} - u_{Dst}h_{22} = 0 \\ u_{Src}h_{10} + v_{Src}h_{11} + h_{12} - v_{Dst}u_{Src}h_{20} - v_{Dst}v_{Src}h_{21} - v_{Dst}h_{22} = 0 \end{aligned} \tag{3-3}$$

Note that the two linear equations in Eq. (3-3) are independent for solving **H**. For more clearly explanation, by defining the homography matrix as vector form,

$$\mathbf{h} = (h_{00}, h_{01}, h_{02}, h_{10}, h_{11}, h_{12}, h_{20}, h_{21}, h_{22})^{\mathrm{T}}.$$

Eq. (3-3) can be reformulated into

$$\begin{bmatrix} u_{Src} & v_{Src} & 1 & 0 & 0 & 0 & -u_{Dst}u_{Src} & -u_{Dst}v_{Src} & -u_{Dst} \\ 0 & 0 & 0 & u_{Src} & v_{Src} & 1 & -v_{Dst}u_{Src} & -v_{Dst}v_{Src} & -v_{Dst} \end{bmatrix}\mathbf{h} = \mathbf{0} \tag{3-4}$$

This is a linear equation with 9 unknown values in **h**. We can find that a pair of corresponding points in two image planes can provide two independent equations for solving **h**. That is we have a linear system with 2-by-9. If there are N pairs of points, a linear system with 2N-by-9 is formed. Because the homography transform is a homogeneous vector, the last element h_{22} equals to 1, and it needs at least 4 pairs of points to solve **h** [4] . Then the linear system is formulated as the 8-by-8 square one,

$$\overbrace{\begin{pmatrix} u_{Src,1} & v_{Src,1} & 1 & 0 & 0 & 0 & -u_{Dst,1}u_{Src,1} & -u_{Dst,1}v_{Src,1} \\ 0 & 0 & 0 & u_{Src,1} & v_{Src,1} & 1 & -v_{Dst,1}u_{Src,1} & -v_{Dst,1}v_{Src,1} \\ u_{Src,2} & v_{Src,2} & 1 & 0 & 0 & 0 & -u_{Dst,2}u_{Src,2} & -u_{Dst,2}v_{Src,2} \\ 0 & 0 & 0 & u_{Src,2} & v_{Src,2} & 1 & -v_{Dst,2}u_{Src,2} & -v_{Dst,2}v_{Src,2} \\ u_{Src,3} & v_{Src,3} & 1 & 0 & 0 & 0 & -u_{Dst,3}u_{Src,3} & -u_{Dst,3}v_{Src,3} \\ 0 & 0 & 0 & u_{Src,3} & v_{Src,3} & 1 & -v_{Dst,3}u_{Src,3} & -v_{Dst,3}v_{Src,3} \\ u_{Src,4} & v_{Src,4} & 1 & 0 & 0 & 0 & -u_{Dst,4}u_{Src,4} & -u_{Dst,4}v_{Src,4} \\ 0 & 0 & 0 & u_{Src,4} & v_{Src,4} & 1 & -v_{Dst,4}u_{Src,4} & -v_{Dst,4}v_{Src,4} \end{pmatrix}}^{\mathbf{A}} \overbrace{\begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{pmatrix}}^{\mathbf{h}} = \overbrace{\begin{pmatrix} u_{Dst,1} \\ v_{Dst,1} \\ u_{Dst,2} \\ v_{Dst,2} \\ u_{Dst,3} \\ v_{Dst,3} \\ u_{Dst,4} \\ v_{Dst,4} \end{pmatrix}}^{\mathbf{b}} \tag{3-5}$$

To solve the linear system, many numerical methods could be applied. In MPEG-FTV, the VSRS estimates the homography vector **h** using the function "cvFindHomography" in the open computer vision library (OpenCV). This function solves the above linear system by the singular value decomposition (SVD). For the hardware implementation of SVD, we should consider precision, computational complexity, and hardware cost. The details of implementation method are discussed in Chapter 5.1.2.

3.2.2 Depth mapping using forward warping

In this step, both the depth maps of reference left view and right view are warped to the virtual views. This warping process is called forward warping, which represents the 3D warping from the reference view to the virtual view as in Fig. 3-2. For each pixel in the reference, according to its depth value and the corresponding homography matrix, the warped position in the virtual view can be acquired using the homography transform in Eq. (3-2). With the warped position, the depth value in the reference view is copied to the virtual view. After pixel-by-pixel warping, the whole new depth map is synthesized in the virtual view. Note that there are two new depth maps warped from left view and right view separately.

Since the original depth map in the reference view may have noise, or the warping process may induce sampling alias, the new depth map in the virtual view usually suffers from small noisy holes. To remove them, the new depth map is post-filtered by the median filter [5], [6], [7].

3.2.3 Texture mapping using reverse warping

In this step, the texture in the virtual view can be synthesized by warping the texture in reference view according to the depth map in the virtual view. This warping process is called reverse warping because the warping direction is from the virtual to reference view, instead of the reference to virtual view in the previous forward warping.

The details of the reverse warping are presented as follows. Using depth value in the virtual view, which is the result of previous depth mapping, the position in virtual view can be warped to the position in reference view. With the corresponding positions, the texture from reference view is copied pixel-by-pixel to virtual view. Because of the post-filtered depth map, the synthesized frame has fewer holes, compared to the typical texture mapping by forward warping. Note that the decreased holes are caused mainly by round-off error [7], but the large holes due to occlusion still remains in the result of this step. Note that this step should be performed for left view and right view separately. Thus, two synthesized frame for the virtual view are generated.

3.3 Occlusion handling and blending

In DIBR process, the mapped synthesized images incur large holes which may be seen by the right reference view or left reference view but occluded in the synthesized view. To recover the so called *disocclusion*, the study of one-view synthesis for stereoscopic video generation has adopted various hole-filling methods, such as linear interpolation [1] and horizontal extrapolation [10]. But it suffers from serious texture distortion since the large holes cannot be recovered well. Better filling methods may consider the depth information or gradient character into interpolation [11]. In order to alleviate the difficulty of hole-filling, the depth smoothing method [12]-[14] is adopted before the 3D warping process. The aim of the depth smoothing is to reduce the size of holes by means of lessening the sharp discontinuity in depth maps.

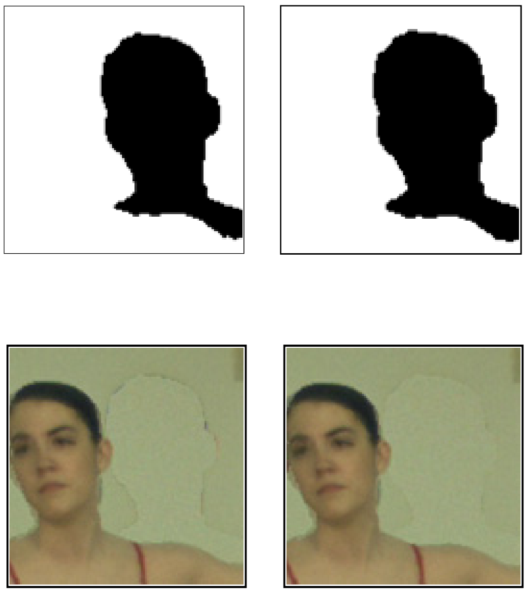
On the other hand, for the two-view synthesis algorithm we adopt, two synthesized views from left reference view and right reference view are produced separately. Hence the holes caused by occlusion can be filled from each synthesized view, which is based on that some scene may be seen by only left-eye or only right-eye. Thus the blending method proposed in [7] is formulated by the equation

$$I(u,v) = \begin{cases} (\mathbf{1}-\alpha)I_L(u,v) + \alpha I_R(u,v), & \text{if } I_L(u,v) \text{and } I_R(u,v) \text{are not holes} \\ I_L(u,v), & \text{if } I_R(u,v) \text{is hole} \\ I_R(u,v), & \text{if } I_L(u,v) \text{ is hole} \\ 0, & \text{else} \end{cases} \tag{3-6}$$

$$\alpha = \frac{|\mathbf{t}-\mathbf{t}_L|}{|\mathbf{t}-\mathbf{t}_L|+|\mathbf{t}-\mathbf{t}_R|} \tag{3-7}$$

In Eq. (3-6), there are four blending modes, and the hole-pixel can be detected in the previous depth mapping step. For the first mode, a pixel is not a hole if it is mapped during the warping process. When both synthesized views are not holes, the blending is done by weighted addition with the distance factor in Eq. (3-7), where **t** is the translation vector in extrinsic camera parameters. For the second and third modes, if only one synthesized view can be obtained, the blending will only the corresponding reference view. For the last mode, if the pixel is a hole both from reference left and right views, it will be marked as the "final-hole", and should be filled by other image interpolation method. The other method is introduced in the next section.

However, boundary noise appears after blending due to mismatch between depth map and texture, especially at disparity discontinuity region. To eliminate these artifacts, the hole-maps are dilated one or two pixels, so that holes borders are extended and will be filled with background. Fig. 3-4 shows the synthesized image with and without dilating hole-map. Because the hole-map after dilating has the wider borders of the holes, the two hole-maps are different at the boundary area. A new truth table of blending mode can be adopted as in Table 3-1. We define the different area of the two hole-maps as "Boundary Special", because it is the border area of depth discontinuity, and may induce boundary noise. Note that its blend mode is different from the general case.



(a) (b)

Fig. 3-4 Synthesized image (a) without dilating the hole-map and (b) with dilating the hole-map

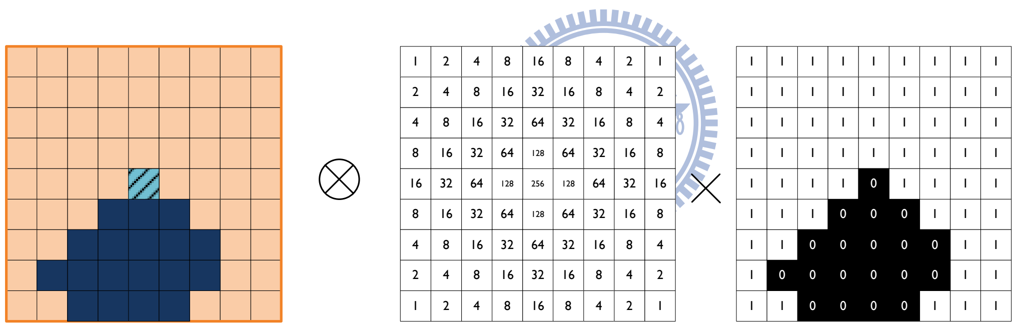
Table 3-1 Blend mode truth table

0 if hole, 1 if not hole				Blend Mode	Boundary Special
reference to L	reference to R	After dilation reference to L	After dilation reference to R		
0	0	-	-	Final-hole	0
0	1	0	-	R only	0
1	0	-	0	L only	0
1	1	0	1	R only	1
1	1	1	0	L only	1
1	1	0	0	Weighted add	0
1	1	1	1	Weighted add	0

3.4 Hole-filling

Remaining holes flagged as "final-hole" after blending can be handled by different methods in view synthesis. FTV uses the advanced inpainting method [15] for the general warping mode, and the simple linear interpolation method for the 1D horizontal shift mode. Müller et al. [9] extrapolates only background color on holes by examining depth value on the two sides of hole-border, because foreground has larger depth and background has smaller depth. Oh et al [16] proposed a depth-based inpainting method which also fills holes with only background color. No matter what methods are, because these remaining final-holes cannot be seen from any reference views but only can be filled reference to surrounding pixels, it is enough for holes to be filled *naturally* but not *exactly*.

However, the inpainting is a frame-based image processing and are more complex in hardware implementation. Thus, we apply a simple bi-linear interpolation, which performs a 2D low-pass filter with geometric distance weighting on the final-hole flag. In this thesis, we implemented this simple bi-linear interpolation by block-based as shown in Fig. 3-5.



1	2	4	8	16	8	4	2	1
2	4	8	16	32	16	8	4	2
4	8	16	32	64	32	16	8	4
8	16	32	64	128	64	32	16	8
16	32	64	128	256	128	64	32	16
8	16	32	64	128	64	32	16	8
4	8	16	32	64	32	16	8	4
2	4	8	16	32	16	8	4	2
1	2	4	8	16	8	4	2	1

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1
1	1	1	0	0	0	1	1	1
1	1	0	0	0	0	0	1	1
1	0	0	0	0	0	0	1	1
1	1	0	0	0	0	1	1	1

Fig. 3-5 Bi-linear interpolation of hole-filling

However, the block size is related to hole-size and is a key factor in internal memory size as well as in performance. When block is too small, the larger holes in frame border may not be filled; if block is too big, the buffer size becomes large, and the interpolated texture would be noised. Table 3-2 shows the performance of some sequences under different block sizes. The sequence “Ballet” has larger holes, so that its performance is better when block size increases. The sequences “BookArrival,” “LoveBird1” and “Kendo” have smaller holes, so that when block size is increased, the performance is degraded. For general good performance in average, we choose the block size of 9x5 in our design.

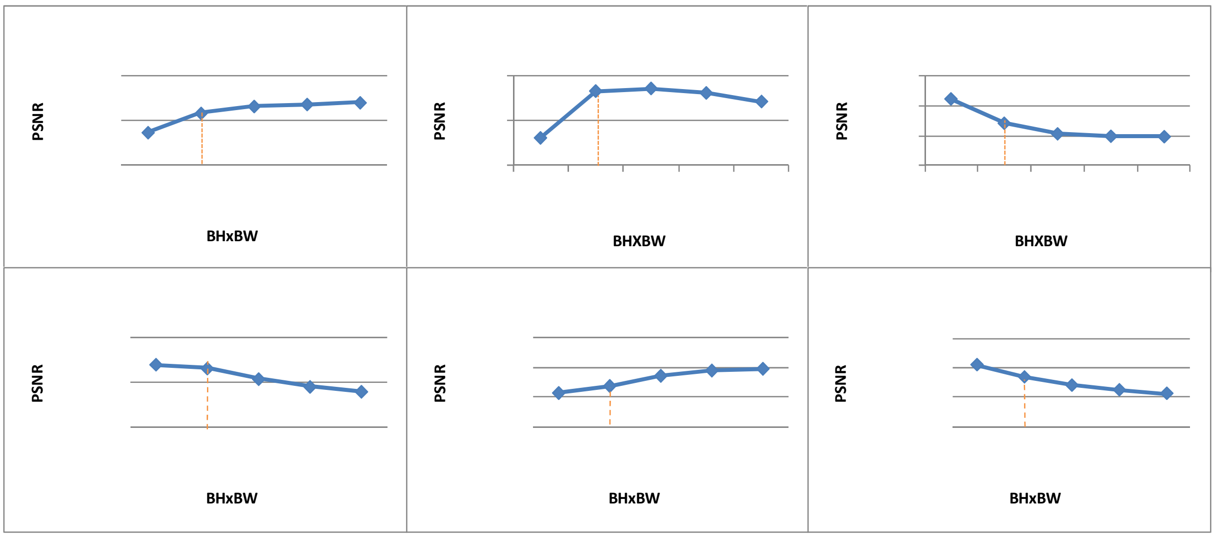


Fig. 3-6 Performance of bi-linear interpolation for hole-filling in different block size

Table 3-2 Performance of hole-filling by using bi-linear interpolation in different block size

Y-PSNR Performance (dB)

Block size	**Ballet**	**Breakdancers**	**BookArrival**	**Lovebird1**	**Newspaper**	**Kendo**
5x3	33.18638	33.06250	36.41172	31.80200	30.67576	33.00001
9x5	33.20828	33.16606	36.37078	31.80157	30.67691	32.99998
13x7	33.21609	33.17187	36.35280	31.80039	30.67858	32.99997
17x9	33.21837	33.16193	36.34878	31.79952	30.67945	32.99996
21x11	33.22026	33.14299	36.34814	31.79897	30.67974	32.99996

Chapter 4

Proposed architecture

Our objective is to implement a real-time view synthesis (VS) engine corresponding to the VSRS algorithm for the frame size of HD1080p (1920x1080). There are three main challenges in implementing the VSRS algorithm. First, for general application, the 3D warping requires much more hardware complexity, especially in storage cost, than the horizontal shift method. That results from the cameras with rotation, so that the disparities between each view are not only in horizontal direction as shown in Fig. 2-2. Hence data storage is increased from 1D to 2D, and its data control becomes complicated.

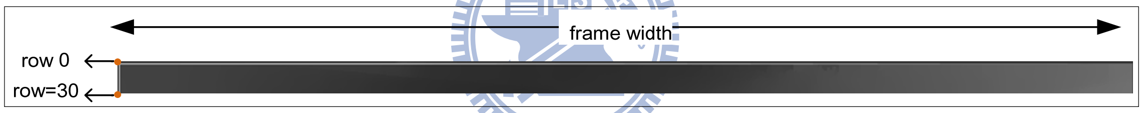
Second is that the VSRS algorithm uses two steps of 3D warping, one for depth mapping and the other for texture mapping. The main advantage in two steps warping is that warped depth map can be post-filled for better texture mapping. In addition, because the reverse warping processes in the index of target view, two synthesized views from different reference views can be processed at blending and hole-filling steps in parallel. However, the data storage and access are increased for the additional synthesized depth map, and therefore internal memory and bandwidth utilization become critical in architecture design.

Third challenge is in the hole-filling. As Chapter 3.4 described, we choose the simple bi-linear interpolation with the block size of 9x5 in this step. But the data storage and access is still a challenge because of the remaining holes at irregular and discontinuous positions.

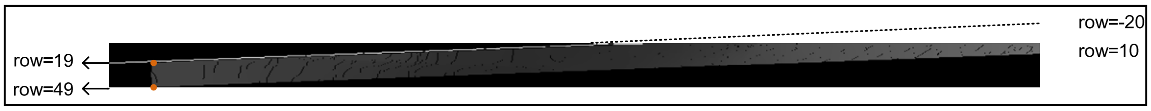
Our architecture design is focus on solving the above three challenges. Finally the architecture adopts the two frame-level pipeline stages, and each with hierarchical column-level pipeline stages.

4.1 Two frame-level pipelining stages

Because the depth mapping using forward warping and the texture mapping using reverse warping are performed at different positions, the former depth mapping should stores the warped depth of virtual view in a reorder buffer for the latter texture mapping. This size of reorder buffer will be disparity level if videos are rectified with no rotation. On the other hand, its size is up to multiple rows if videos are with rotation. For example of "Ballet" in Fig. 4-1, the region of depth map from row 0 to row 30 in reference view are forward warped to the target view with out-of-order position. The previous 20 rows in reference view are warped out to frame range, and this means that the first whole row of the virtual view is collected after the warping process of 20 rows. We need a buffer size of frame width by 20, which is 40.96KB to buffer the previous mapped depth, and is up to 108KB for HD1080P.



(a)



(b)

Fig. 4-1 Warped depth map row0 to row30 of "Ballet" (a) is the reference view and (b) is the virtual view.

To eliminate this reorder buffer, we propose the architecture of two frame-level pipeline stages, which performs the depth mapping process and the texture mapping process in different stages.. Fig. 4-2 shows the schedule of the proposed two frame-level architecture. The warped depth is stored in the external memory at 1st stage and read at 2nd stage for texture mapping.

With the proposed two-level architecture, Table 4-1 shows that the total bandwidth is increased for the additional access of warped depth map. By using 64-bits bus with the working frequency of 200MHz, the bus utilization is 39.375% for the video throughput of 30 frames per second (fps). In addition, for the specific analysis, we use the bus width of 64-bits in our design.

Note that in Fig. 4-2, the warped depth maps are written and read simultaneously by 1st stage and 2nd stage. This means that there are ping-pong-like external memories for the warped depth maps. One is written the warped depth value of frame *i*, and the other is read the warped depth value of frame *i-1*.

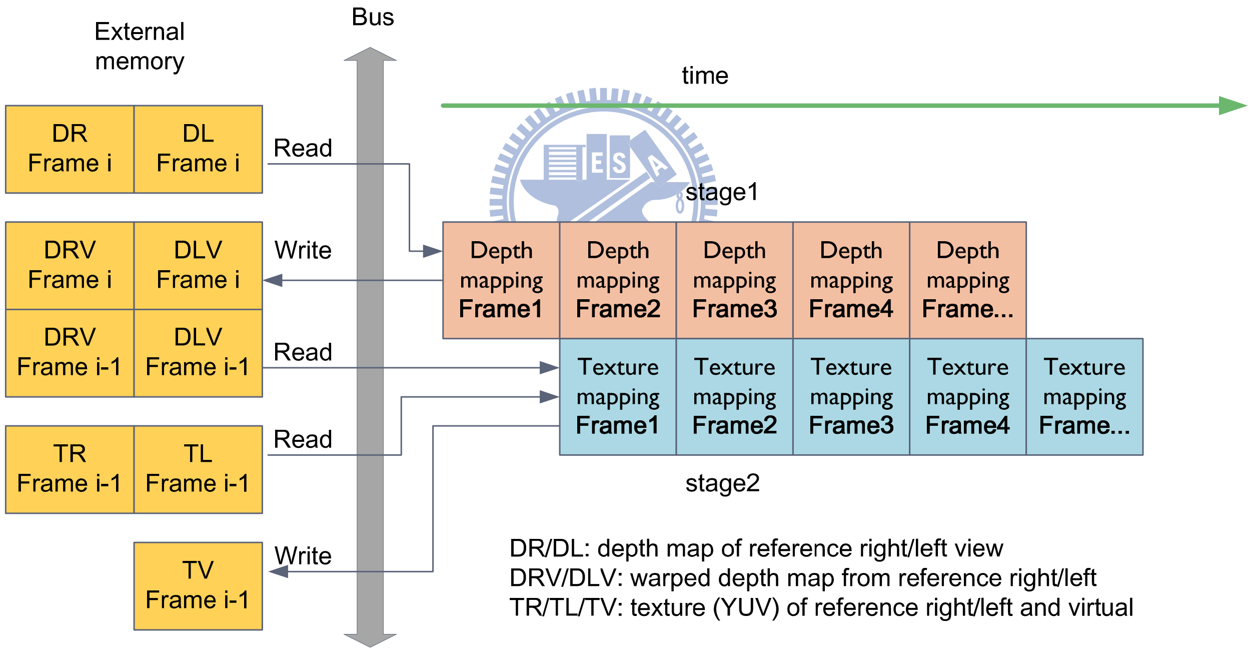


Fig. 4-2 Two frame-level pipeline and the access between external memory

Table 4-1 Total bandwidth of two frame-level stages

	Architecture	
Data	One frame-level stage	Two frame-level stage
Depth map (Left, Right)	2Frame(Read)	2Frame(Read)
Depth map (Left to virtual Right to virtual)	-	2Fram(Read, Write)
Texture (YUV, L, R)	3Frame(Read)	3Frame(Read)
Texture (YUV, virtual)	1.5Frame(Write)	1.5Frame(Write)
Total bandwidth (2MB/frame)	13MB	21MB

4.2 Scan-column warping order

Usually a Z-buffer/depth buffer of frame-size is needed in depth mapping [8]. In 3D world coordinate, if foreground objects and background objects are projected to the same position in the image plane, the foreground objects will occlude the background objects. The Z-buffer should store all warped depth value in the depth mapping process for depth comparison to handle the occlusion problem.

For on-the-fly warping processing, the size of a shift window is proportional to the horizontal search range (SR_H) and the vertical search range (SR_V). Furthermore, the search range is different among scenes and is increased when frame-size is larger. For the example, the frame size of “Ballet” , is 1024x768, the SR_H is 55 and SR_V is 197 for camera 5 and camera 4. Hence the total Z-buffer is at least 21.67KB and is up to 57.13KB for HD1080P.

To eliminate the Z-buffer usage, when cameras are configured in a straight line, the foreground will occlude background in the same scan-line correctly if we scan from left to right for right-view warping and scan from right to left for left-view warping. This warping method is called depth-compatible order method, whose necessary constraint is the epipolar lines are parallel to scan-line. This is the *scan-line order* under the cameras with precisely parallel configuration, and the Z-buffer can be omitted in this case.

When cameras are with rotation, Morvan [4] has derived the occlusion-compatible scanning order for non-rectified images according to the epipolar geometry as shown in Fig. 4-3. For C and C' are camera locations for virtual view and reference view; Pb and Pf are both projected to p in the virtual view and the epipole e' is the point of C projected to the reference plane. The scanning order in the reference view should be from the frame border to epipole e' in the epipolar line so that foreground Pf can occlude background Pb correctly in the vitual view.

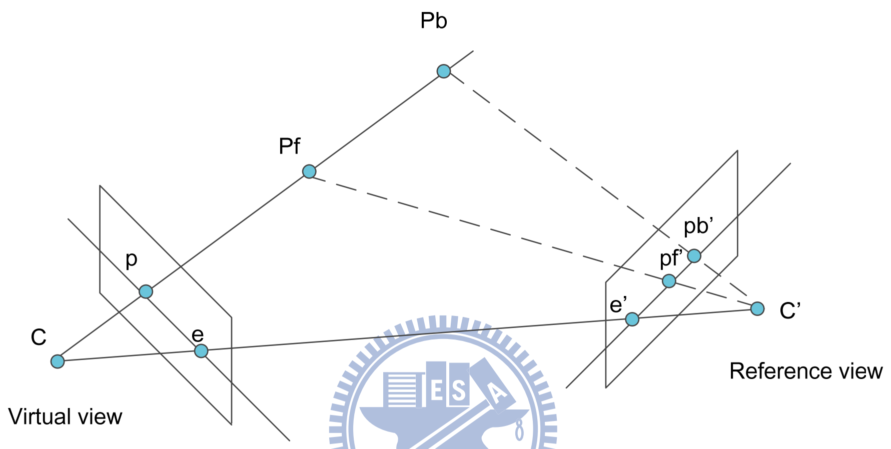


Fig. 4-3 Occlusion-compatible scanning order revised from [4]

However, the calculation of epipolar line consumes additional hardware computational cost, followings we analyze different scan-order approaches to eliminate the computational cost. If the epipolar lines lie in the reference right view as in Fig. 4-4(a), the original *scan-line order* will fail since an epipolar line will be warped in different scan-line and the correct scan order in the epipolar line will be ruined as shown in Fig. 4-4(b). The similar situation occurred in Fig. 4-4(d), (e) for the reference left view. An example of *scan-line order* error is Fig. 4-5(a).

We find that the warping order can be transferred to *scan-column order* that will not induce occlusion error even if without the accurate epipolar lines because the order is adapted from the range of a line to the range of the frame as shown in Fig. 4-4(c)(f). An example of *scan-column order* is Fig. 4-5(a).

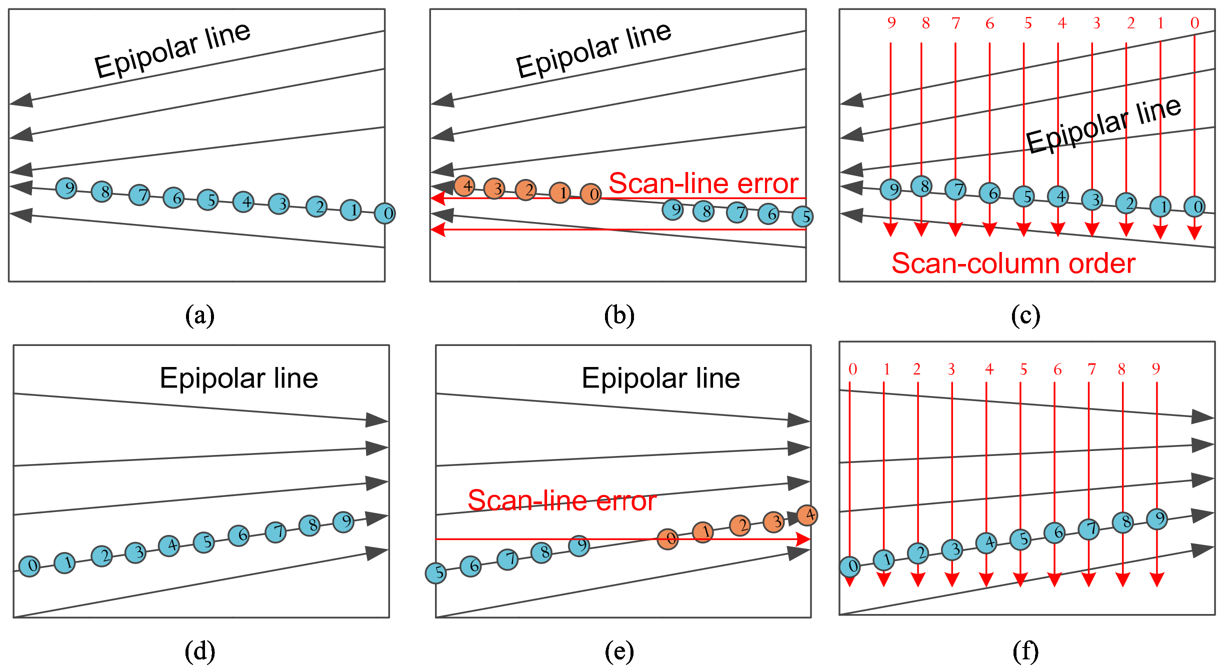


Fig. 4-4 Warping order analysis.

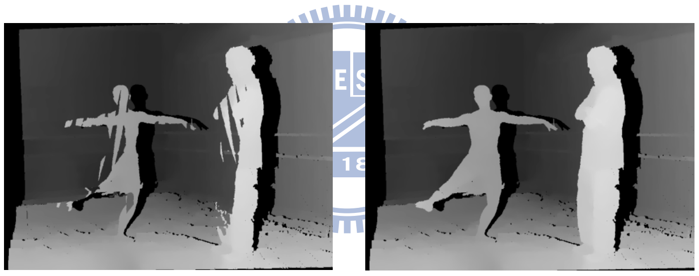


Fig. 4-5 Warped depth maps without Z-buffer with (a) scan-line order and (b) scan-column order

However, the location of epipole determines the scan order in epipolar line and hence for cases that epipole *lies inside* the visible frame as in Fig. 4-6, our *scan-column order* must be modified according to the epipole position.

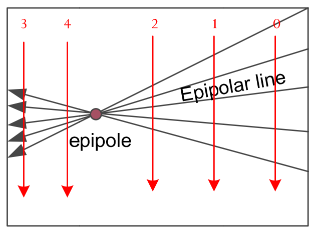


Fig. 4-6 Epipole lies inside frame

4.3 Analysis of bus efficiency and bandwidth in warping process

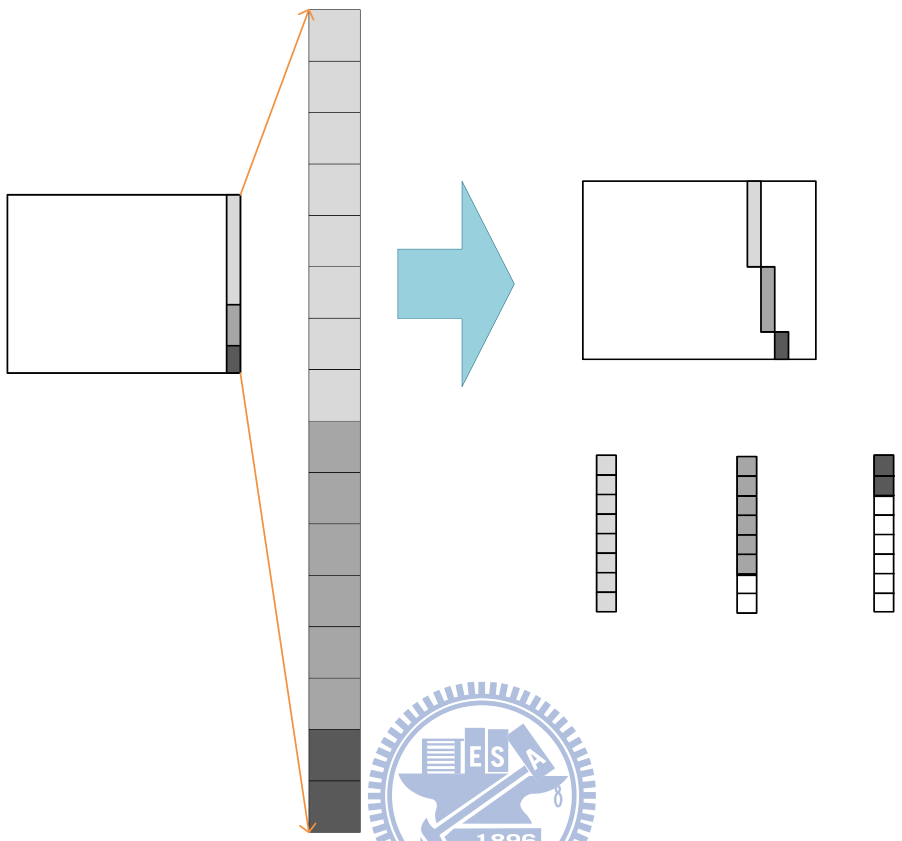


Fig. 4-7 Forward warping example of a column in the reference view

For the proposed frame-level pipelining architecture, the out-of-order warping also increases the request-times between the core and bus due to writing warped depth in the 1st frame-level stage and reading warped texture in the 2nd frame-level stage. Because the scan-column order is adopted in this design, both the depth map and texture are arranged into a column to be stored in a row of the external memory. With this data arrangement, Fig. 4-7 shows a column of reference view is forward warped to the synthesized view. The warped positions are continuous in the synthesized view if their depth values are the same. But pixels with different depths are mapped to different columns, so that they are stored in different rows of the external memory. That results from increasing request times to bus. In addition, if the data size of a continuous segment with identical depth is less than the bus width in byte unit, the bus transfer is not efficiency since only partial data in a transfer are available. It further makes the required bus cycle be increased. Although the total bandwidth is enough under the setting of 64-bit bus, the inefficient transferring results in increasing bus utilization and degrades the overall performance.

The transition efficiency is related to depth continuity, which depends on sequences. Moreover, if we attempt to promote the efficiency of data transmission, the input and output (I/O) buffers would be increased to collect more data for reordering. Fig. 4-8 and Table 4-3 shows the analysis of the bus efficiency with different I/O buffer sizes for the sequences "Breakdancers," "Ballet," "BookArrival," and "Lovebird1". The detail data of "Breakdancers" is shown in Table 4-2. Note that we set the bus width as 64-bits, and these sequences are run for a frame. In Table 4-2, for the bus transmission mode, the single mode means that data are transmitted for less than 8 bytes; while the burst mode means data are transmitted for more than 8 bytes. The request times should be accumulated one if the depth discontinuity happened. The transmit times should be accumulated one for the single mode, and the burst length for the burst mode. In addition, the maximum length is the maximum continuity in depth. The average bus efficiency is calculated as dividing the frame size by the average transmission times multiplying bus width in bytes..

Tabel 4-2 shows that when buffer size is increased, the average bus efficiency is increased. But the efficiency cannot reach to the maximum value, 100%, except for the whole frame is with the same depth value which does not occur. Therefore, with the scan-column order warping, we choose the I/O buffer as the frame height for higher bus efficiency. Table 4-3 shows that the average efficiency reaches to 88.7%. This means the data are ready when a column access is complete. This concept could be further extended to the column-level pipelining architecture, and is described in Chapter 4.5.

Table 4-2 “Breakcancer,” Cam5 to Came4, analysis of bus efficiency for different I/O buffer size

“Breakdancers” Camera#5 to Camera#4						
Buffer Size (Byte)	Single mode (Byte)	Burst mode (Byte)	Request times	Transmit times	Max length (Byte)	Average bus efficiency
8	151712	0	151712	151712	8	0.648
16	94899	27664	108731	122563	16	0.802
32	72799	48140	87488	120939	32	0.813
64	61048	57206	76440	118254	64	0.831
128	55581	60750	71009	116331	128	0.845
256	52790	62245	68269	115035	137	0.855
768	51097	63395	66412	114492	187	0.859

Table 4-3 Bus efficiency in buffer size of 8byte to frame width for different sequences

Buffer Size (Byte)	Breakdancers C5toC4	Breakdancers C3toC4	Ballet C5toC4	Ballet C3toC4	BookArrival C10toC8	BookArrival C7toC8	Lovebird C5toC6	Lovebird C10toC6	Average
8	0.648	0.662	0.623	0.628	0.774	0.848	0.800	0.783	0.721
16	0.802	0.807	0.774	0.775	0.906	0.972	0.916	0.896	0.856
32	0.813	0.810	0.779	0.791	0.917	0.955	0.901	0.887	0.857
64	0.831	0.826	0.793	0.812	0.925	0.957	0.904	0.887	0.867
128	0.845	0.838	0.803	0.825	0.935	0.961	0.909	0.892	0.876
256	0.855	0.846	0.807	0.834	0.941	0.967	0.915	0.899	0.883
768	0.859	0.850	0.809	0.838	0.946	0.973	0.919	0.904	0.887

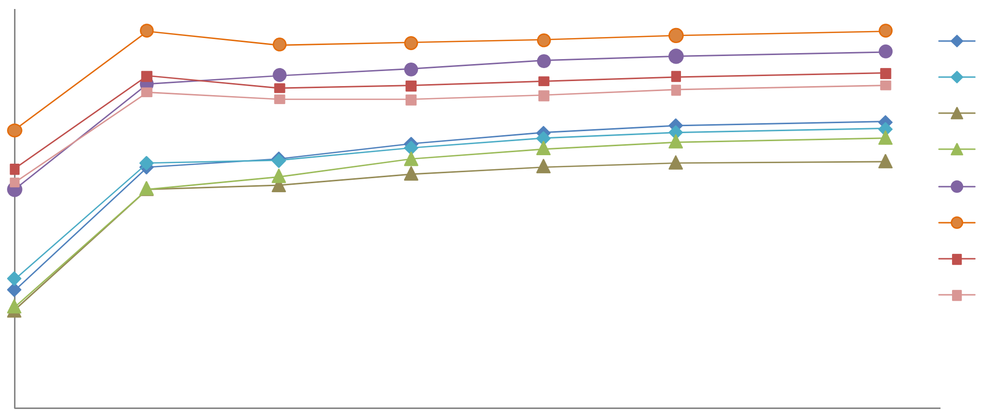


Fig. 4-8 Bus efficiency in different I/O buffer size

4.4 Analysis of bandwidth and memory size in hole-filling process

As discussion in Chapter 3.4, we select bi-linear interpolation with a window of 9x5 to do hole-filling. The position of final-hole is determined according to the depth map processed by the median filtering and the two synthesized hole-maps processed by the dilation as shown in the blend mode of Table 3-1. However, the locations of holes are irregular and different in among frames and scenes.   For this random position characteristic, the run-time synthesized output has to be stored in the internal memory for hole-filling. To minimize the internal memory usage, the data of whole frame can be stored in the external memory. But to lessen the overall bandwidth, the data should be stored in the internal buffer with the size of several columns. Thus, that is a trade-off issue between the external memory bandwidth and the internal memory usage, and the two approaches are proposed in the following sub-chapters.

4.4.1 Frame-level buffering with vertically dynamic reuse

To have a smaller volume in internal memory, the bi-linear interpolation in hole-filling can be processed at another frame-level stage. But the bandwidth utilization will be raised if doing interpolation by fetching every pixel in block size. For example, a 9x5 bi-linear interpolation needs the additional bandwidth of 45x2MB for a HD1080p frame. If only fetch pixels are flagged as final hole, a hole table recording hole position is needed. As shown in Table 4-4, the holes counts are up to about 1% of a frame. With this percentage, the bandwidth will be up to 0.9MB and the hole-buffer needs 55KB memory for an HD1080p frame.

The bandwidth can be saved by using the data reuse technique. For the random-positioned characteristic of hole, in the blending step, the hole-position (HP) and the hole-height (HH) are pre-calculated and stored into a hole-index table. In the hole-filling step, according to the stored HP and HH, texture data are fetched into the filter kernel, and then the center pixel can be filtered as shown in Fig. 4-9. This process is performed row by row until the holes are completely filled. For the holes are belong to the same index, their data for filtering can be reused, so that the bandwidth could be saved by 53.903%% in average as shown in Table 4-4.

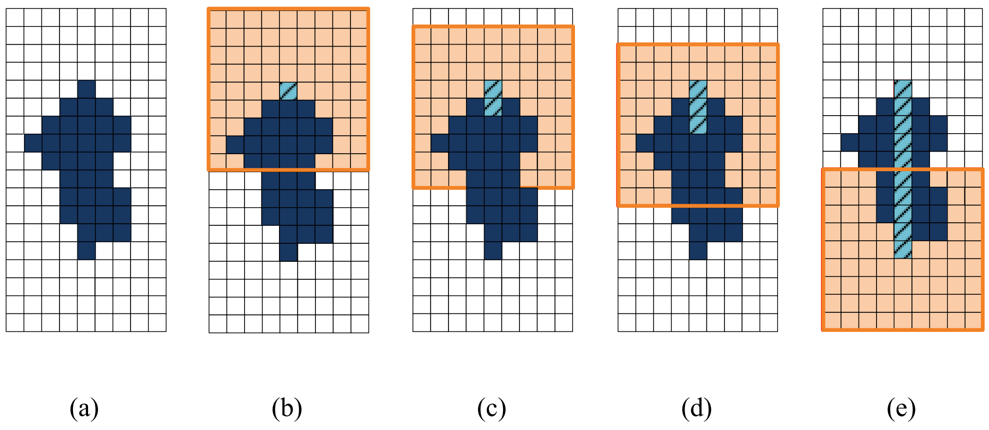


Fig. 4-9 Filling process with dynamic vertically reuse for a 9x9 bi-linear filter.

However, the bandwidth is low but the request-time is high. Because the kernel is fetched row by row but the texture data in external memory is stored by column, filling a hole needs request the bus for the block-size times.

4.4.2 Column-level buffering

The other method is run-time storing data in the internal memory. In the *scan-column-order* warping process, the blended texture is stored in the column buffers. Hence we take this strategy as *column-level buffering*. After all buffers are full, the interpolation kernel is full and the interpolation starts immediately. It needs no additional bandwidth. However, there is the number of column buffers equal to block-width, and both the texture and flagged final-holes need be buffered at the same time.

4.4.3 Comparison

Table 4-4 and Table 4-5 list overall bandwidth usage and internal memory size for frame-level buffering and column-level buffering. Note the hole-index table recording holes position and holes length. Because holes length is less than 128 in the test sequences, we choose the bits of representing hole-length as 7 bits. Furthermore, we need 22bits to represent frame indexes for HD 1080P video. Therefore the total bits in hole-index table are 29bits as written in Table 4-5.

Although the frame-level buffering approach is with the smallest internal buffer, it suffers from additional bandwidth. Finally, we choose *column-level buffering* as the interpolation method for it contributes no bandwidth usage and the 56.16Kbit of internal memory is also smaller than the use of *vertically dynamic reuse* approach.

Table 4-4 Bandwidth of different bi-linear interpolation approaches

	Frame-level buffering			**Frame-level buffering with vertically dynamic reuse**			**Column-level buffering**
Sequence name	**Number of holes**	**Number of holes /frame (%)**	**Bandwidth (MB /frame)**	**Number of holes index**	**Bandwidth (MB /frame)**	**Save (%)**	**Bandwidth (MB/frame)**
Ballet	7499	0.954	0.506	1475	0.145	71.405	0
Breakdancers	7019	0.893	0.474	1243	0.127	73.147	0
BookArrival	948	0.121	0.064	79	0.012	81.481	0
Lovebird1	1197	0.152	0.081	1067	0.073	9.654	0
Newspaper	901	0.115	0.061	185	0.018	70.243	0
Champagne	574	0.047	0.039	113	0.011	71.390	0
Kendo	183	0.023	0.012	183	0.012	0.000	0
average						**53.903**	**0**

Table 4-5 Internal memory size of different bi-linear interpolation approaches

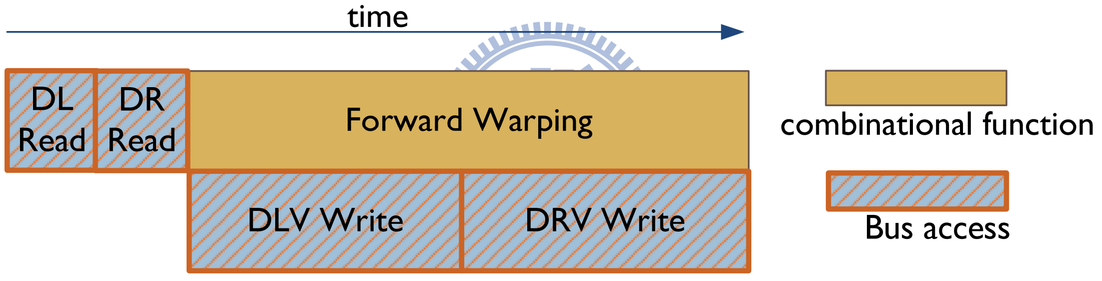
Internal memory size	**Frame-level buffering Original access**	**Frame-level buffering with vertically dynamic reuse**	**Column-level buffering**
Kernel	9x5=45 bytes	9x5=45 bytes	9x5+2x5x3=75 bytes
Hole-table	20Kx22bit=55KB	-	-
Hole-index table	-	2Kx29bit=7.25KB	-
Y buffer	-	-	4.32Kbytes
Final hole buffer	-	-	4.32Kbits
U buffer	-	-	1.08Kbytes
V buffer	-	-	1.08Kbytes
Total	**55.04KB**	**7.25KB**	**7.02KB**

4.5 Column-level pipeline and bus scheduling

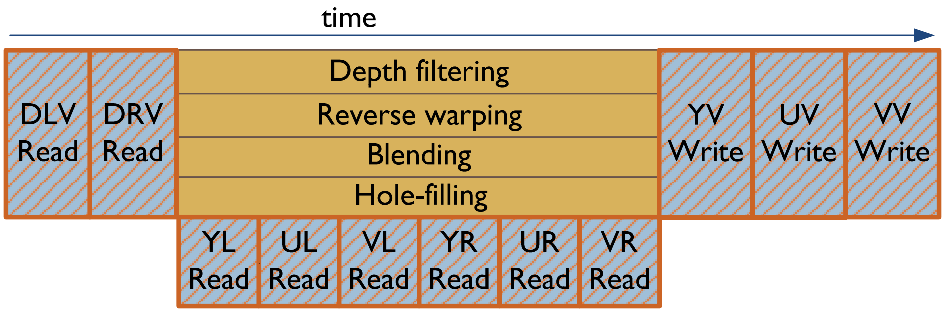
From Chapter 4.3, we choose the data access by the column order. More specifically, the two critical data accesses, the warped depth writing and the reference texture reading, are column-pipelining for the forward warping and the reverse warping, respectively. From Chapter 4.1, we divide the depth mapping and texture mapping into two frame-level pipeline stages. In the first stage, the depth mapping is done by forward warping; in the second stage, the texture mapping contains depth filtering, reverser warping, blending, and hole-filling. The overall column-level processing is shown in Fig. 4-10, for the first stage and the second stage.

To let a HD1080P video be processed for the speed of 30fps, and if this design runs at 200MHz, the maximum processing cycle is 6.667M for a frame. For the most critical combinational logic are the warping and depth filtering, because they are the by-pixel process, a pixel can be processed for 3.215 cycles in average. In fact, we can do warping as well as depth filtering in pixel-pipeline hence the throughput is 1 pixel/cycle for left view and right view separately, and the total throughput achieves 0.5 pixels per cycle in the virtual view.

Therefore, the data rate for combinational design is enough, and the total bandwidth as well, by choosing bus width as 64-bit as described in Chapter 4.1. However, as discussed in Chapter 4.3, the random position of warping makes data access overhead increase. Moreover, the pipeline structure of two frame-level stages means that the bus has to deal with the accesses of two stages simultaneously, and the critical data access of two stages need be processed by sharing bus bandwidth in a certain cycles. Therefore, to ensure the critical access working correctly, we make the regular access of I/O in sequential with the column-level pipelining. The bus first deals the regular data read before column-level process starting; then it deals the random read/write during the column-level pipelining period; finally it does the regular texture write-back access as shown in Fig. 4-10.



(a)



(b)

Fig. 4-10 Scheduling of BUS transmission in processing a column (a) is for depth mapping and (b) is for texture mapping. In fact that bus access is another column-level pipeline stage.

Chapter 5

Hardware implementation

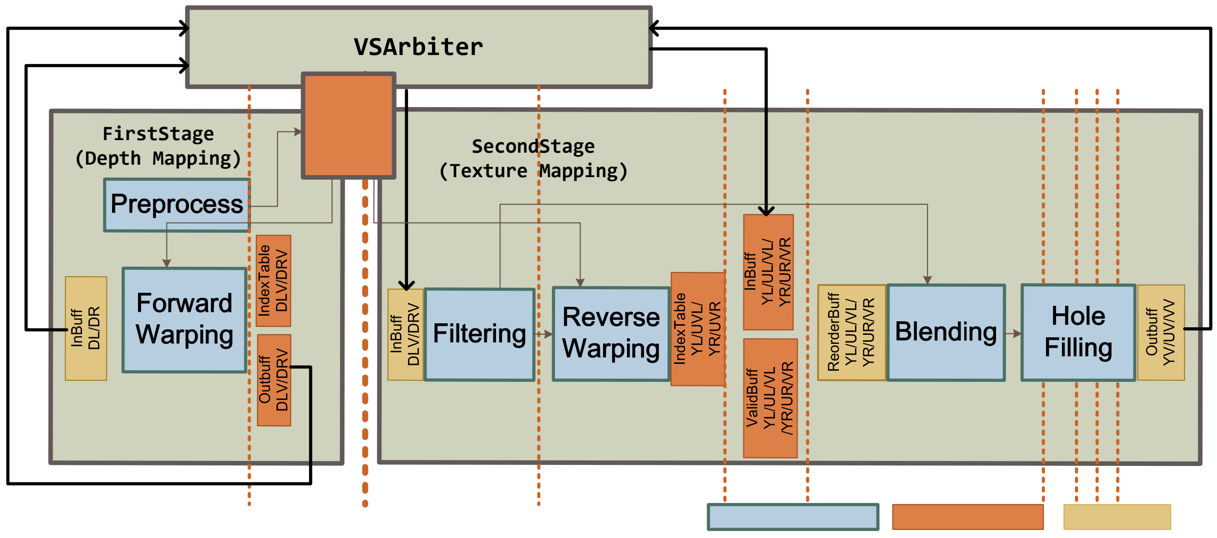


Fig. 5-1 Overall architecture of VS engine

Fig. 5-1 shows the overall hardware architecture which has three main parts: "Depth Mapping," "Texture Mapping," and "VSArbiter." The "Depth mapping" module has two main function blocks: one is the preprocess for homography estimating, and the other is the forward warping for depth mapping. The "Texture mapping" module has two processes with complicated data flow. One is the hole-map processing to decide blend mode, and the other processing contains the depth filtering, reverse warping, blending and hole-filing. Finally, the "VSArbiter" arbitrates the bus requests from the above two modules.

In addition, there are data buffers between the processes for data reuse and reordering. Note that for the notation of buffers, "D" means depth map, and "Y," "U," "V" are the 3-channel texture data. "L," "R," "V" mean reference left view, reference right view, and the virtual view, respectively. "LV" and "RV" mean the warped virtual view form reference left view and reference right view, respectively.

5.1 Preprocess

The preprocess module is mainly for making homography matrices that will be used in the warping process, and does all calculation regarding to camera parameters. Since camera parameters are the same among one video, the module only have to process once over the all frames. For the proposed frame-level pipelining architecture, this preprocess module is installed into the first stage because the second stage of texture mapping module has less timing budget. The detailed architecture of the preprocess module is shown in Fig. 5-2.

The “Ztrans” performs Z-transform for mapping 8-bit sampled *depth level* to *depth value*. The “MakeProjeciton” loads the camera parameters **K**, **R**, **T** to do matrix multiplication for the projection matrix **P**, as formulated in Eq. (2-1). With the depth value and projection matrix **P**, the “Projection transform” calculates the two-side projection in Eq. (2-4). All the input points *src* and the output points *dst* are feed into the “Make Homography” module for computing a homography matrix. Finally, all the coefficients of 256 homography matrices are stored in the internal memory “Homography table.”

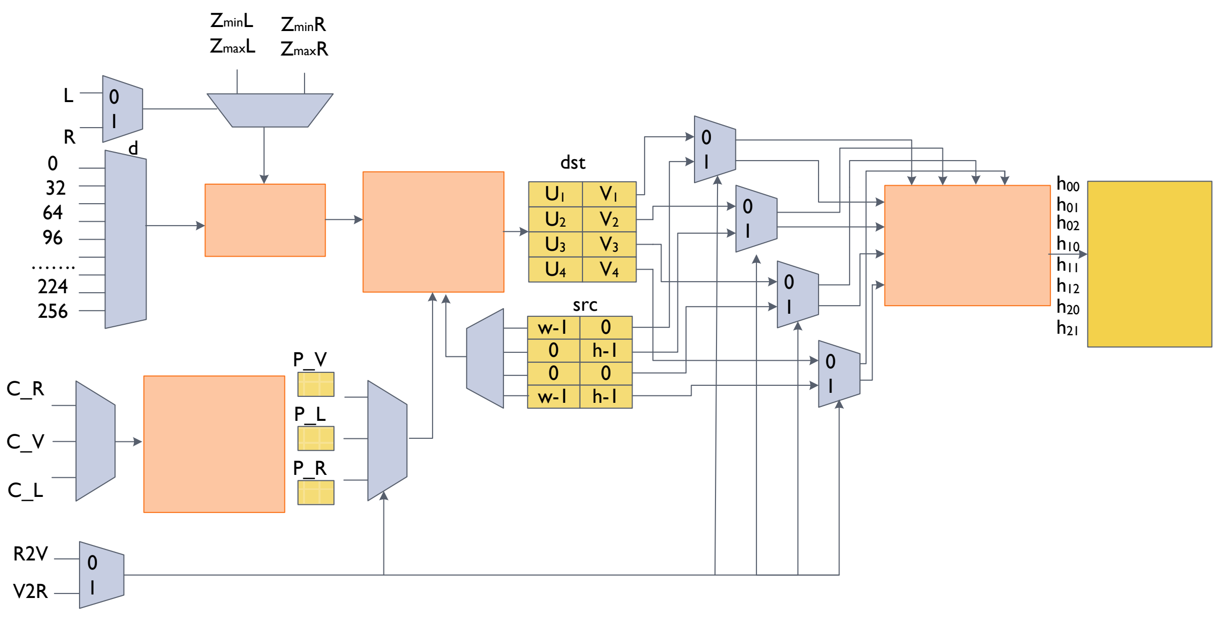


Fig. 5-2 sub-modules of the preprocess module

According to the description in Chapter 3.2.1, a homography matrix can be solved through an 8x8 linear system. With the homography matrix, we can reduce the computational complexity in the warping process from two 3x4 matrix multiplication of projection transform to one 3x3 matrix multiplication of homography transform. However, the 4 pairs of correspondence for solving the linear system should be obtained by projection transform, so that its hardware cost (i.e. MakingProjection and ProjectionTransform) cannot be avoided.

Besides, the homography matrices need be stored in internal memory. There are 256 sets of homgraphy matrices for 256 depth values, and each set consists of 4 homography matrices for 4 different mapping relationships. The mapping relationships are reference left view to virtual view, reference right view to virtual view, virtual view to reference left view, and virtual view to reference right view. The former two is for the depth mapping, and the latter two is for texture mapping. As a result, the internal memory should store 256x4 homography matrices. Since each homography matrix is a 3x3 matrix with 8 floating elements, the buffer size will be of 524,288 bits.

To solve this large internal memory problem, Lin et al. [17] proposed a linear-interpolated approximation (LIA) method. They found that the homography matrices have the linear relationship between successive depth values. Thus, only N+1 sets of homography matrices are stored in memory, and the other depth values are interpolated using the stored matrices with inverse of depth distance as interpolation weighting, as shown in Eq. (5-1).

$$\mathbf{H}_N(Z) = \mathbf{H}_{Base,i} + \frac{Z - i \times 256/N}{256/N}\mathbf{H}_{Inc,i}\ , \text{for } Z = 0, \frac{256}{N}, 2\frac{256}{N}, \ldots 255$$
$$\mathrm{i} = \left\lfloor \frac{Z}{256/N} \right\rfloor, \mathbf{H}_{Base,i} = \mathbf{H}(\min(i \times 256/N, 255)), \mathbf{H}_{Inc,i} = \mathbf{H}_{Base,i+1} - \mathbf{H}_{Base,i} \qquad (5\text{-}1)$$

We adopt this method and choose N as 8 for implementation. Therefore we will need 9 homography matrices of depth value equaling to 0, 32, 64, 96, 128, 160, 192, 224, and 255. The overall flow chart for estimating homography is shown in Fig. 5-3.

In the following parts, we describe the detailed architecture of the modules ProjectionTransform and MakeHomography.

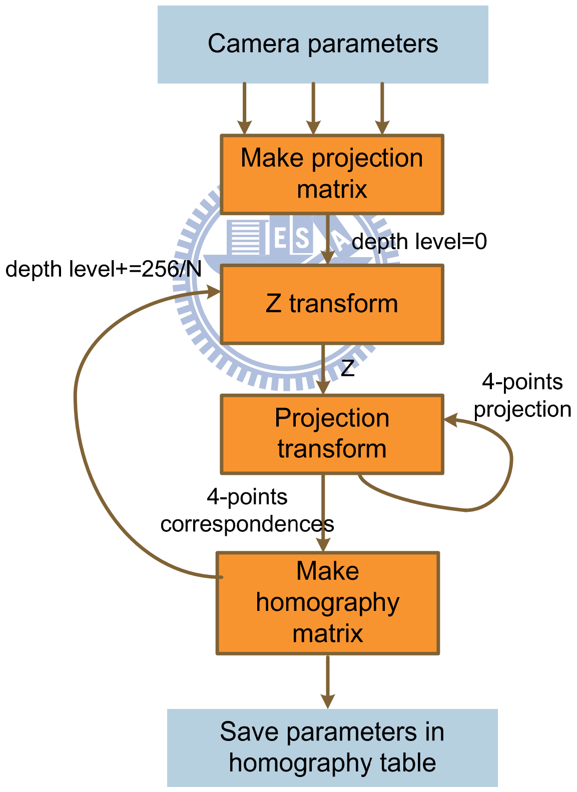


Fig. 5-3 Flow chart of the preprocess module

5.1.1 Projection transform

The projection transform will do 4-points transform for homography estimation. Suppose $\mathbf{P}_{Src}$=$\mathbf{K}_{Src}$ [$\mathbf{R}_{Src}$; $\mathbf{T}_{Src}$] projects $(u_{Src}, v_{Src})^T$ to $(X, Y, Z)^T$ and $\mathbf{P}_{Dst}$=$\mathbf{K}_{Dst}$ [$\mathbf{R}_{Dst}$; $\mathbf{T}_{Dst}$] projects $(u_{Dst},$

$v_{Dst})^T$ to $(X, Y, Z)^T$. The Eq. (2-4) shows the relation of $(u_{Src}, v_{Src})^T$ and $(u_{Dst}, v_{Dst})^T$ obtained by the back-projection and projection. For the projection process, we need adders, multipliers and dividers in hardware implementation. In addition, the required bits of these operators regard to the Z value as well as the camera parameters, which depends on the content of scenes. Because the hardware cost will be increased when the operator bits are large, in the following, we discuss several methods to decrease the operator bits, especially for dividers.

Table 5-1 Z value and the scaled Z of some sequences

	Before scaling		Scale	After scaling	
Sequence	Zmax	Zmin	factor	Zmax	Zmin
Ballet	42	130	1	42	130
Breakdancers	44	120	1	44	120
BookArrival	23.345	54.471	1	23.345	54.471
Lovebird1	1560.122	156012.2	1/1024	1.523	152.355
Akko&Kayo	2342.249	12491.99	1/64	36.597	195.187
Newspaper	2715.182	9050.605	1/64	42.424	141.415
Champagne	2281.358	7045.261	1/32	71.292	220.164
Kendo	448.2512	11206.28	1/64	7.003	175.098

Table 5-1 shows the maximum and minimum depth of Z among different scenes. If the range of Z is a large dynamic range, the depth Z must be represented by wide-bit in hardware implementation. Hence the divider is very large. Nevertheless, the depth Z is sampled as 8-bit disparity in the depth map, so that we can scale $(X, Y, Z)^T$ to $(sX, sY, sZ)^T$, where s is a base-2 scalar and sZ is a 8-bit value, while keeping the original performance well. From Eq. (2-4), the relation of $(u_{Src}, v_{Src})^T$ and $(u_{Dst}, v_{Dst})^T$ remains when we also scale **T** to s**T**. By doing the step, the integer parts of Z and **T** can be sampled to only 8-bit numbers, and the divider can cut for maximum 10 bits in integer part. To further decrease the divider complexity, we transform the fixed-point division to IEEE 754 floating-point division, and the gate count can reduced to 3.258% at UMC 90nm process. The associated comparison is list in Table 5-2.

Table 5-2 Comparison of divider sizes for projection

	Divider size (integer.fractional)	Gate count(K) @90nmUMC	Reduction
Original	48.11/29	80.079	100%
Z scaling	39.11/20	44.682	55.797%
Floating-point	9.23/9.23	2.870	3.258%

5.1.2 Homograhy matrix estimation

As mentioned in Chapter 3.2.1, the homography matrix is estimated by solving an 8x8 linear system, and there are many methods for solving a linear system. The general methods are using matrix decomposition, such as the well-known LU decomposition or singular value decomposition (SVD). However, they suffer from high computational complexity in software implementation. The SVD is accelerated in VLSI design for many years. By using two-sided rotation, it can be implemented with parallel operators, such as systolic array [19] for efficiency. The operating processors in the systolic array containing rotation and angle computation to do two-sided rotation, and are further implemented as coordinate rotation digital computer (CORDIC) [20] for ease the hardware complexity. The CORDIC can replace trigonometry computation by iterative shift and addition [21].

The other easier method excepts for matrix decomposition in solving linear system is the iterative methods such as Gaussian-Seidel method as formulated in Eq. (5-2). For solving the above linear system in Eq. (3-5), the a_{ii}, h_i, and b_i are elements of **A**, **h**, and **b**, respectively; k is the iteration times, and n equals to 8 because this is an 8-by-8 system.

$$h_i^{(k)} = -\sum_{j=1, j<i}^{n} (a_{ij}/a_{ii})h_j^{(k)} - \sum_{j=1, j>i}^{n} (a_{ij}/a_{ii})h_j^{(k-1)} + (b_i/a_{ii}) \tag{5-2}$$

Solving the linear system using iteration method has the following advantages. First is that the computation precision can be controlled by the iteration times. Next, because the homography estimation is independent of frame-size, it is not the timing-critical function in the VS engine.

Therefore, the iteration is a sequential computation and it can be with less logic in hardware implementation.

However, the key fact of iteration computation is the convergence condition. In fact, Gaussian-Seidel methods will converge if the matrix A has the property of diagonal dominance [22],

$$\left|a_{ii}\right| > \sum_{j=1, j\neq i}^{n} \left|a_{ij}\right| \tag{5-3}$$

This is a sufficient condition, and this condition will be *nearly fit* if we rearrange **A** to let bigger elements in the matrix diagonal. Usually the matrix **A** are built uses source points in the frame corner, (width-1, 0), (0, height-1), (0, 0), and (width-1, height-1), which are indexed as source point 1, 2, 3, and 4 respectively. However, in order to let bigger elements in the diagonal, we rearrange Eq. (3-5) to the following equation,

$$\begin{pmatrix} u_{Src,1} & v_{Src,1} & 1 & 0 & 0 & 0 & -u_{Dst,1}u_{Src,1} & -u_{Dst,1}v_{Src,1} \\ u_{Src,2} & v_{Src,2} & 1 & 0 & 0 & 0 & -u_{Dst,2}u_{Src,2} & -u_{Dst,2}v_{Src,2} \\ u_{Src,3} & v_{Src,3} & 1 & 0 & 0 & 0 & -u_{Dst,3}u_{Src,3} & -u_{Dst,3}v_{Src,3} \\ 0 & 0 & 0 & u_{Src,1} & v_{Src,1} & 1 & -v_{Dst,1}u_{Src,1} & -v_{Dst,1}v_{Src,1} \\ 0 & 0 & 0 & u_{Src,2} & v_{Src,2} & 1 & -v_{Dst,2}u_{Src,2} & -v_{Dst,2}v_{Src,2} \\ 0 & 0 & 0 & u_{Src,3} & v_{Src,3} & 1 & -v_{Dst,3}u_{Src,3} & -v_{Dst,3}v_{Src,3} \\ 0 & 0 & 0 & u_{Src,4} & v_{Src,4} & 1 & -v_{Dst,4}u_{Src,4} & -v_{Dst,4}v_{Src,4} \\ u_{Src,4} & v_{Src,4} & 1 & 0 & 0 & 0 & -u_{Dst,4}u_{Src,4} & -u_{Dst,4}v_{Src,4} \end{pmatrix} \begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{pmatrix} = \begin{pmatrix} u_{Dst,1} \\ u_{Dst,2} \\ u_{Dst,3} \\ v_{Dst,1} \\ v_{Dst,2} \\ v_{Dst,3} \\ v_{Dst,4} \\ u_{Dst,4} \end{pmatrix} \tag{5-4}$$

By using Gaussian-Seidel method in Eq. (5-2), to solve the rearranged system Eq. (5-4) , we find that the homography matrix can be estimated for certain iteration times, which is related to calculation precision. In the constraint of the precision requirement of homography coefficients list in Table 5-3, the iteration times for the system to converge are less than 20 as shown in Fig. 5-4.

Table 5-3 Precision of homography coefficients

coefficient	h_{00}	h_{01}	h_{02}	h_{10}	h_{11}	h_{12}	h_{20}	h_{21}
(integer.fractional)	2.16	2.16	8.5	2.16	2.16	8.5	1.27	1.27

.

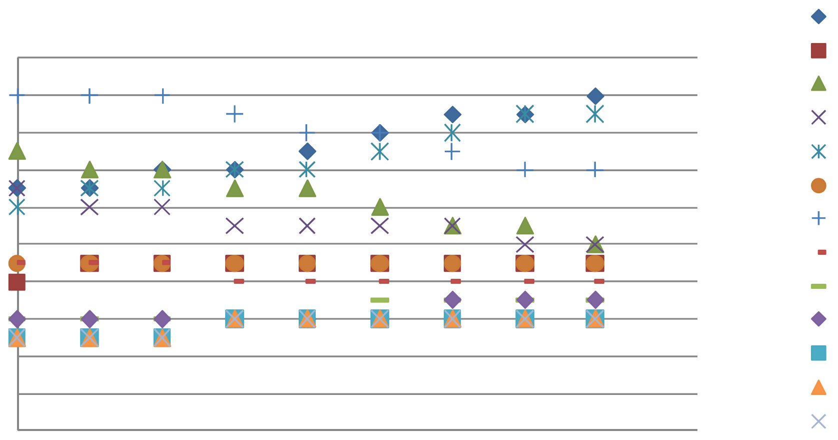


Fig. 5-4 Iteration times of homography estimation

When storing the estimated homography matrix, the homogrphy coefficients will be divided into $\mathbf{H}_{\mathbf{Base}}$ and $\mathbf{H}_{\mathbf{Inc}}$ as shown in Eq. (5-1). In addition, according to the bits setting in Table 5-3, the total bits of a homography matrix are 154 bits. Therefore we need 308 bits to store a homography matrix, and totally 8x308 bits for setting N as 8 in the LIA method. Because there are four mapping relations, the total homography storage in the VS engine should be 32x308 bits. However, due to the proposed two-stage frame-level pipelining architecture, the homography estimation is finished at the first stage while the reverse warping in the second stage also requires the homography matrices. Therefore, the homography table should be implemented with a ping-pong buffer for the reverse warping. In summary, the homography table is 48x308 bits in our design.

5.2 Warping

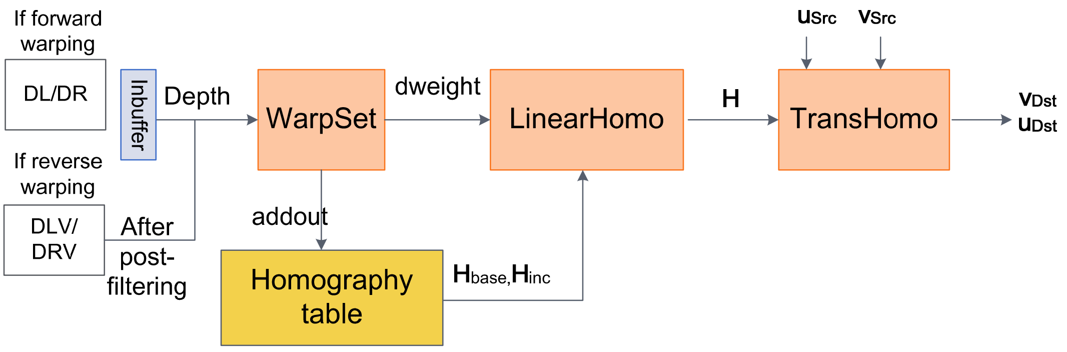


Fig. 5-5 Sub-modules of warping

The architecture for warping is shown in Fig. 5-5. It is the same for both forward warping and reverse warping but only the input is different; one is the depth map of reference views, and the other is the warped depth map of virtual view. The “WarpSet” module controls the input data, and refers to homography table according to the input depth. The “LinearHomo” module linearly interpolates the homography parameters as in Eq. (5-1). The “TransHomo” module performs the homography transform in Eq. (3-2), with 18- stage pipelining to achieve the throughput of 1 pixel/cycle as in Fig. 5-6.

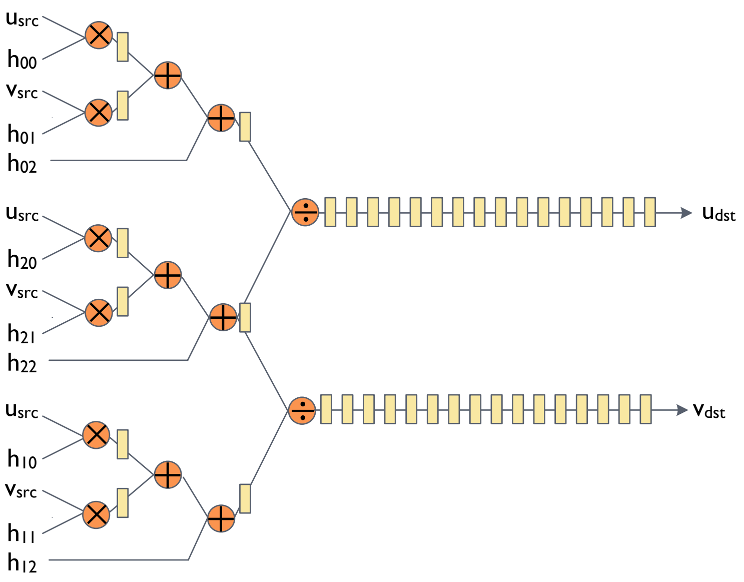


Fig. 5-6 Combinational logic of the “TransHomo” module; the two dividers are implemented using Synopsys DesignWare block IP “DW_div_pipe” of 16 pipeline stages

5.2.1 Forward warping

In the VSRS software, a hole-map of virtual view labeled with 0 for hole and 1 for non-hole could be simultaneously produced in the forward warping process. To avoid storing the hole-map in internal memory, the information of hole-map could parasitize in the warped depth map using the depth of 0 as hole. Thus, the warped depth map in external memory are initially set as 0, and after forward warping process, the pixel with zero depth represents a hole. But for some cases, the depth value is 0 at the reference view are not the real-hole although they are still 0 after forward warping. We avoid this condition by leveling up the depth 0 as depth 1 when forward warping.

Furthermore, because of the initialization of warped depth maps in external memory, the bus bandwidth will be increased to 25Mbytes per frame, and the bus utilization will increased to 43.75% when using 64bits bus, as demonstrated in Chapter 4.1. In fact, the initialization process can be done by data memory access (DMA) controller, so that this process will not affect the timing plan as shown in Fig. 4-10. But since the preprocess module is part of the first-stage frame-level pipelining, and this module do not have to access data in external memory, we can overlap the schedule of the external memory initialization and the original preprocess as shown in in Fig. 5-7.

However, the additional bus access will slow down the 2nd frame-level stage because the column-level scheduling plan for the 2nd frame-level stage is started with another bus access. The result of throughput drop due to the conflicts in external memory access will be detail discussed in Chapter 5.6.1.

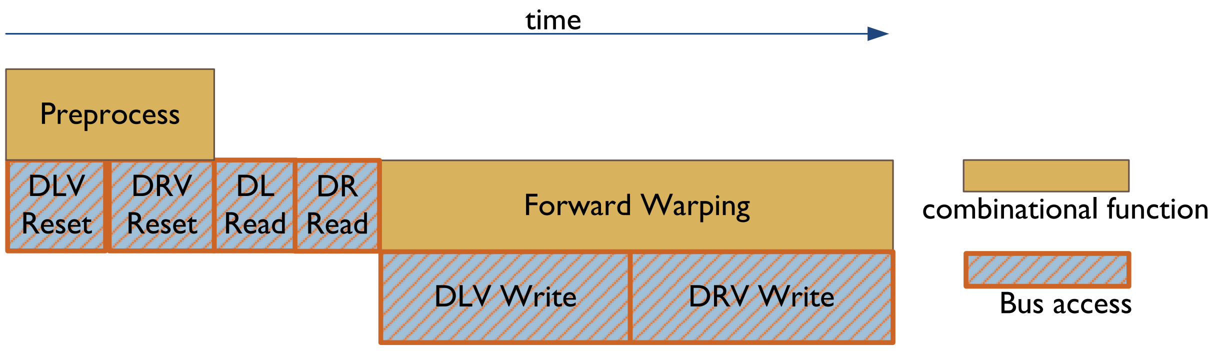


Fig. 5-7 Refined scheduling of BUS transmission in processing the first column for the 1st frame-level stage

5.2.2 Reverse warping

In the second stage in the frame-level pipelining, the reverse warping is one of the column-level stages as shown in Fig. 4-10. After the reverse warping, the texture data are read in next stage. But as discussed in Chapter 4.3, the warped position is random, so in warping process, we record the warped index (u, v) and the length of depth continuity in a ping-pong *index table*. The index table can provide the next blending process to access pixel data from external memory more efficiently. Note that the accessed data may be 1 byte to 8 bytes, depending on the continuity length. To make this data in regular order for the use of blending in the next pipeline stage, we set a ping-pong *valid table* to flag the valid byte of input in the data accessing stage as shown in Fig. 5-8. Then the data can be reordered according to the *valid table* by only grabbing data with valid flag 1 into the reorder buffer.

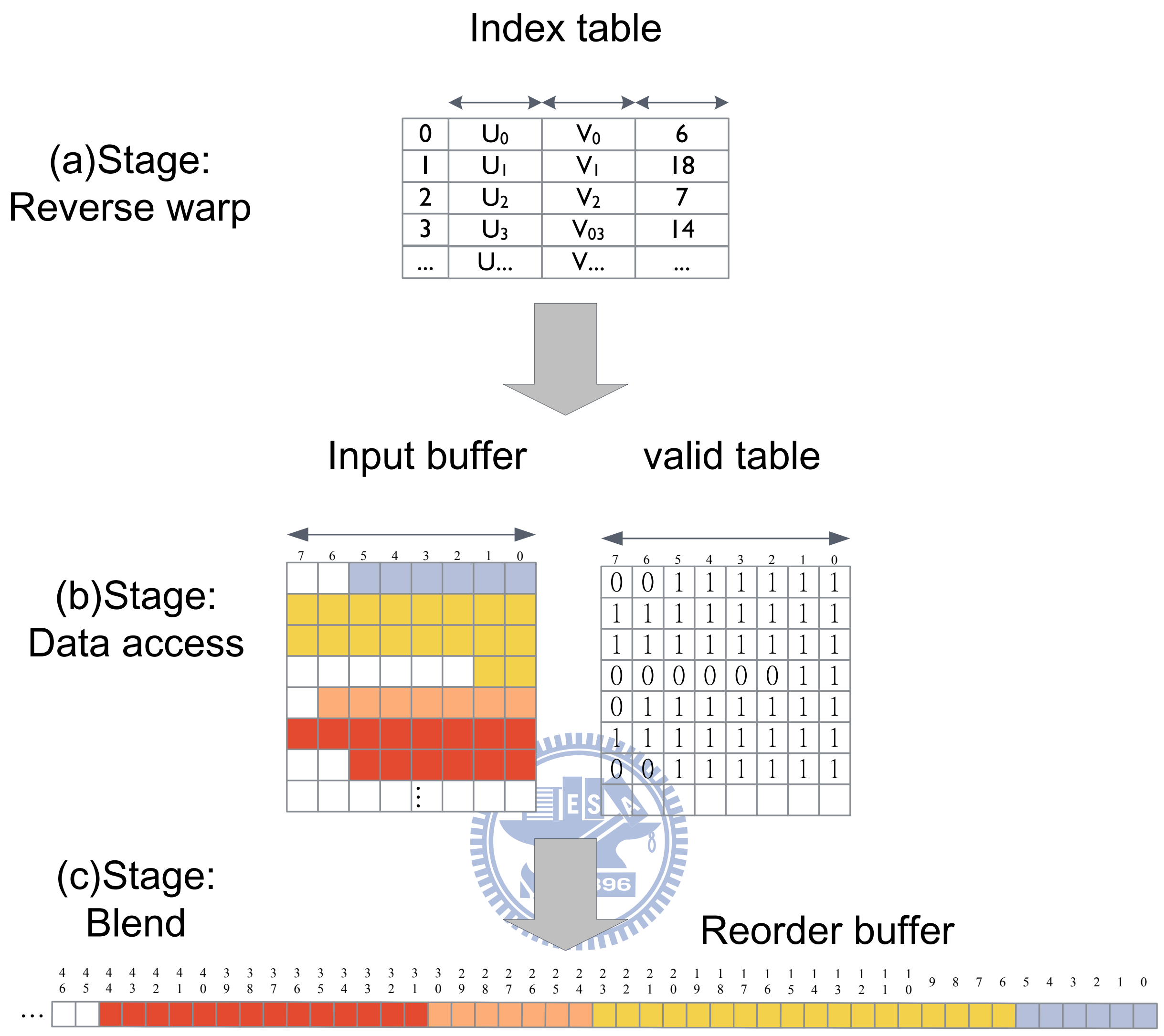


Fig. 5-8 Example of the reordering process using the index table and valid table

5.3 Median filtering, dilation, and bi-linear interpolation

This section demonstrates the block-based median filtering, dilation and bi-linear interpolation. These are all sub-functions of the second stage in frame-level pipelining. For the column-level pipelining architecture as described in Chapter 4.5, these filtering and interpolation need buffers data with the size of several columns. In addition, to achieve the throughput of 1 pixel/cycle, the column buffers are all controlled using FIFOs, which will update data in the first-in-first-out order at every cycle.

5.3.1 Circular FIFO control for 3x3 median filtering and 3x3 dilation

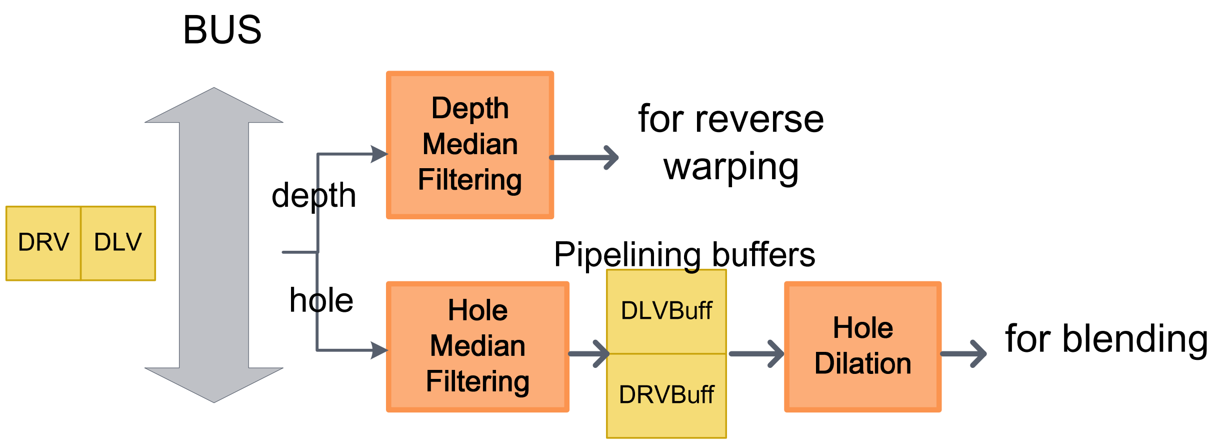


Fig. 5-9 Sub-modules of filter

The 3x3 median filtering is applied both in warped hole-map and depth map as shown in Fig. 5-9. For the warped hole-map, the median concept can be changed to “summation larger than 5.” In other words, if the number of hole in a 3x3 window is larger than 5, the center pixel will be labeled as 1. Also note that the hole-map is built by judging if depth is zero, as described in Chapter 5.2.1. On the other hand, for the warped depth map, 5 minimum selectors are allocated to select the median value among 9 inputs as in Fig. 5-10. The critical path of the 5 minimum selectors equals to the delay of 20 comparators.

The 3x3 dilation can be implemented with a simple Boolean function. And it has the same data control as 3x3 median filtering.

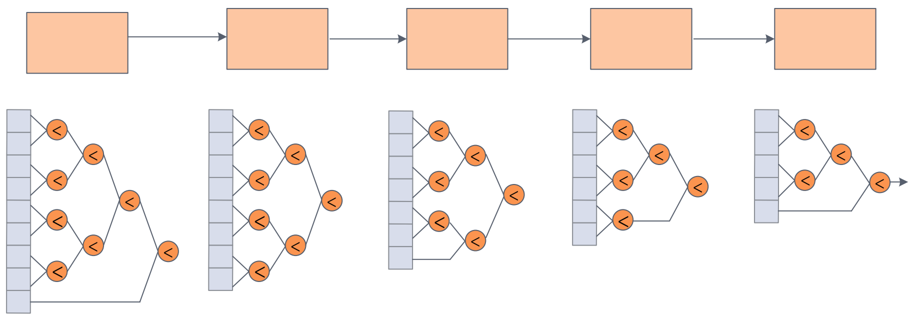


Fig. 5-10 Model of 3x3 Median filter

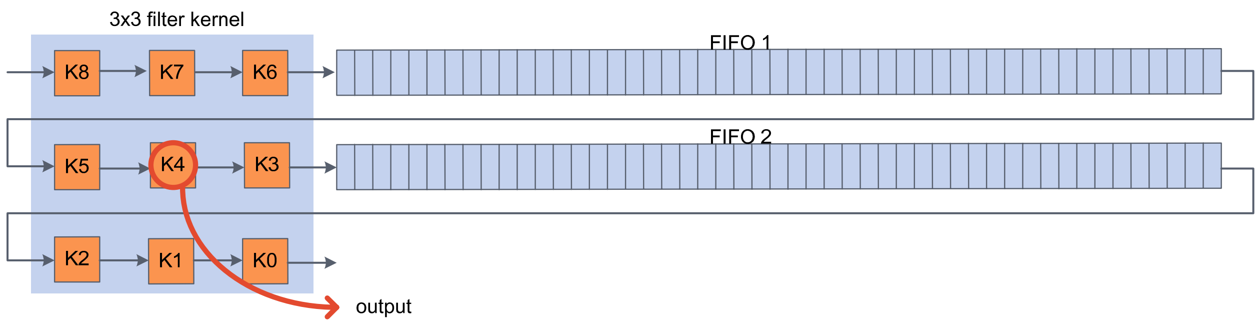


Fig. 5-11 Circular FIFO for 3x3 median filter and dilation

The data control is shown in Fig. 5-11. There are 2 circular FIFOs for this 3x3 median filter. Data is stored into FIFO1 at the first column stage and will be pop out for median filtering in the second column stage, and then is re-stored into FIFO2. It will be pop-out again as the bottom line of filter kernel in the third column stage. The size of this circular FIFO is column height. It pushes in the newest input data and simultaneously popes out the data stored at previous column stage.

By using the circular FIFOs, the input data can be reused for 9 times in 3x3 block-based filtering, and the hence bandwidth usage is small. The bandwidth equals to the frame size but not 9 times of frame size, which is the bandwidth usage without internal buffer.

Note that this median filter is non-accumulated. The filtered data will not be stored back into FIFO, but is passed to next processing. The filtered depth is the input of reverse warping module and the filtered hole-map is stored in another pipelining buffer waiting for dilation as shown in Fig. 5-9..

5.3.2 Column-level accumulated bi-linear interpolation

The bi-linear interpolation is used for hole-filling. As discussed previously, we implemented it as column-level buffering, for the advantage of data reuse and low bandwidth. The block-based bi-linear interpolation has a similar control as 3x3 filtering. For the block size of 9x5, there are 5 stages in column-level pipeline and 4 circular FIFOs for data reuse. Because this interpolation is guided by final-hole flag, the hole is also buffered in another 4 circular FIFOs. Moreover, the texture data is of 3 channels, Y, U, V, for the format of YUV4:2:0, the 9x5 block can be adjusted to 5x3 block for U and V. So there are another two sets of 2 smaller FIFOs for U, V data.

Because we want all holes to be filled successfully, the interpolated data must be capable of being referenced for filling process of following flagged holes. Therefore the interpolated output must be fed back into the circular buffer FIFIO3 as shown in Fig. 5-12.

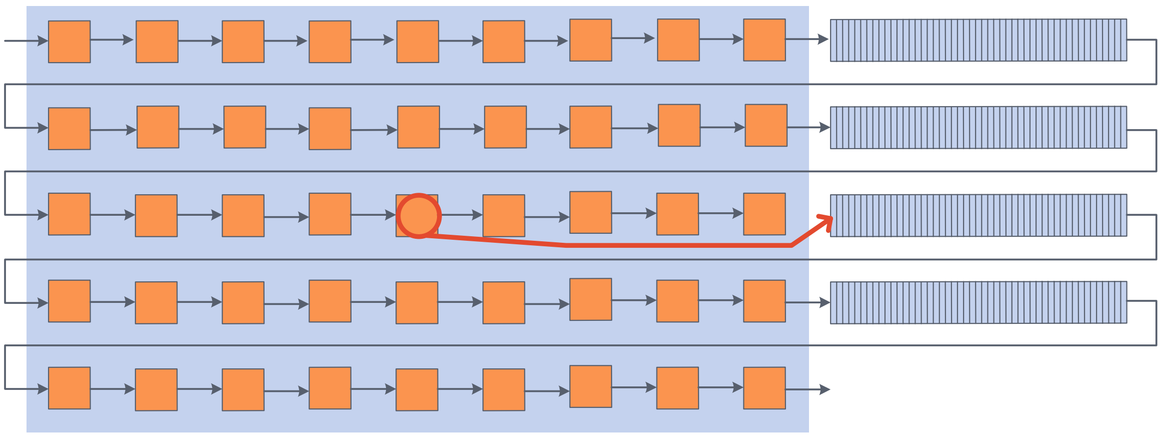


Fig. 5-12 Circular FIFO for 9x5 bi-linear interpolation in column-level accumulation

In fact, because the interpolation has a latency of 4 cycles, this is only a column-level accumulation. The fed back data will be pop-out and reused after a column stage.

5.4 Blending

The “Blending” module calculates Eq. (3-6), and it blends the two synthesized textures from two reference view. As mentioned in Chapter 5.2.2, because the warped data is out of order and is placed different positions between the two synthesized textures, we use the “reorder buffer” to reorder this warped texture data.

The control of reorder buffer is by blend mode list in Table 3-1. For its data accessing step, all data will be fetched if it is non-hole. But the non-hole data may be not used in blending stage if it is in the boundary noise area, and this is the “Boundary Special” case defined in Chapter 3.3. The data in reorder buffer of reference left view will be popped out when the blend mode is “L only”, “Weighted add,” and “Boundary Special” with “R only”. “L only” means only reference left view has reference texture; “Weighted add” means that both left view and right view have reference texture and they will be blended by weighted addition. The “R only” with “Boundary Special” flag means both left view and right view have mapped texture but the left view’s is with boundary noise. In the final case, although there is texture from the reference left view, the blending will not include it for that it is boundary noise. And thus this data in the reorder buffer need to be popped out but in fact, it will not be used in the blending stage.

Note that the reorder buffer for reference right view has a similar control.

5.5 Arbitration

To implement the timing plan in bus access as shown in Fig. 4-10, those accesses units are divided into two groups for different arbitration type as in Fig. 5-13.

Group A is for regular access and the arbitration is by round-robin. In round-robin arbitration, when one's transmission with bus completes, the next grant will give other requests except for no others request. In fact, the Group A can request only once in a column process if bus transmits with the burst length equal to the frame height. That is because the transmitted data are placed regularly in the external memory, where one column data is placed in one row of memory.

On the other hand, group B will request data continuously during the warping process because the transmitted data are in fragment and the access location in memory is random. Furthermore, the access time of Group B is critical in overall processing cycle because access of Group B is one of column-level pipeline stages as discussed in Chapter 4.5. If the access time is increased for the reason of grant lost with bus, the overall pipeline stages will also increase. Therefore the arbitration is designed as that bus will always grant one if it requests continuously.

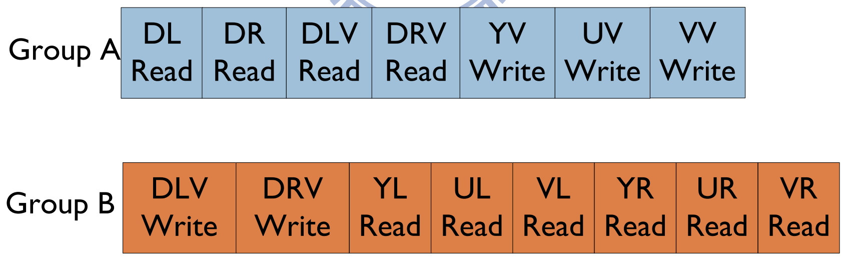


Fig. 5-13 Two different groups in bus arbitration design

5.6 Implementation Result

5.6.1 Overall schedule

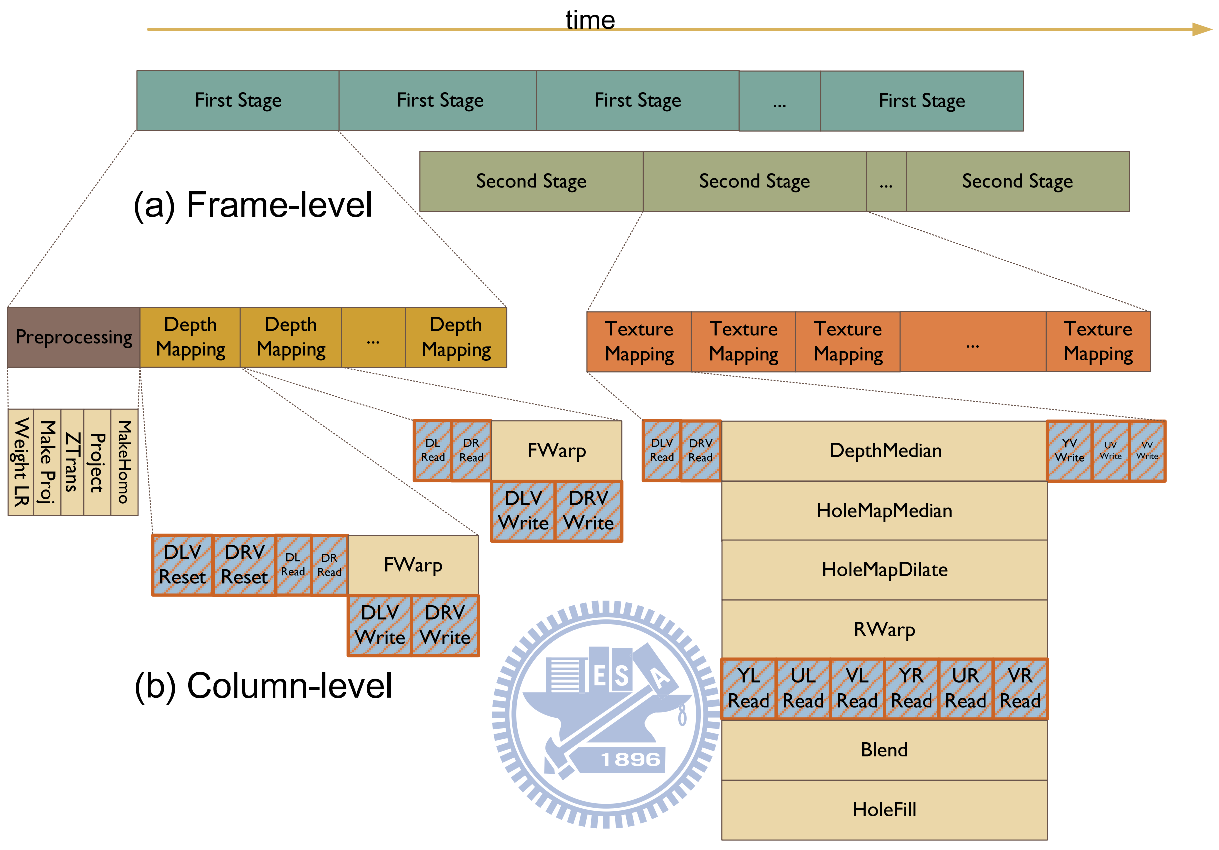


Fig. 5-14 Overall schedule of VS engine

Above shows the overall schedule of our design. This is a hierarchical pipeline from the frame-level, column-level, to pixel-level. In the frame-level pipelining, two stages are the depth mapping and texture mapping. In the column-level pipelining, sub-modules and random bus access both use column-level buffering for efficiency. In the pixel-level pipelining, all the computations are parallelized to achieve the throughput of 0.5 pixels per cycle. Note that the "Preprocess" will only start at initial of a frame; it is not in the column-level pipelining stage. And the reset of external memory for DLV and DRV will be processed only in the first column of a frame.

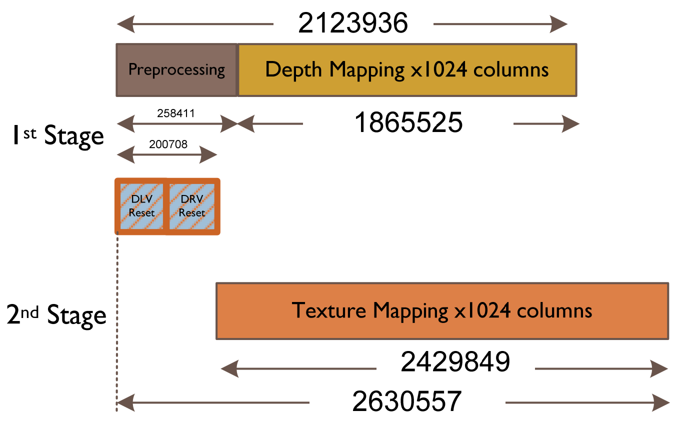


Fig. 5-15 Accurate cycle counts for the sequence “Breakdancers”

Fig. 5-15 shows the accurate cycle counts for the test sequence “Breakdancers” at the working frequency of 200MHz. Note that the frame size of this sequence is 1024x768. The cycle counts of the first frame-level stage is 2,123,936 cycles, containing 258,411 cycles for the “Preprocess” module and 1,865,525 cycles for 1024 columns in total. Because this stage contains 2 column-level pipelining stages, the latency is a column process. Hence in average, one column stage may cost 1,821 cycles. For the second frame-level stage, the total cycle counts are 2,429,849 for 1024 columns. For this stage contains 6 column-level pipelines, the latency is 5 column processes. Therefore one column process in the second frame-level stage is 2362 cycles in average.

The critical one is the 2nd frame-level stage. If we scale the frame size to HD1080P, the cycle counts become 6,406,829 cycles, and the throughput will be 31 fps. However, because the reset of external memory is adopted at the first column, this makes the bus access unable in the beginning to the 2nd frame-level stage. As a result, the total cycle counts for 2nd frame-level stage are 2,630,557 cycles.

Therefore, it needs 2,630,557 cycles to complete a frame of frame size 1024x768. If we scale the frame size to HD1080P, the cycle counts become 6,935,749 cycles, and the throughput will be decreased to 28 fps. In summary, this design can get the throughput of 58M pixels per second.

5.6.2 Hardware cost

This VS engine is synthesized using the UMC90nm technology process and the clock rate is 200MHz. The detail gate count and internal memory report is shown below.

Table 5-4 Implementation result of area, gate count and internal memory size

	SRAM included		**SRAM excluded**		**Internal memory**	
	Cell Area(K)	**Gate count(K)**	**Cell Area(K)**	**Gate count(K)**	**1-port(KB)**	**2-port(KB)**
FirstStage (Depth mapping)						
Preprocess	174.314	61.769	174.314	61.769	0	0
Fwarp	148.922	52.772	148.922	52.772	0	0
Ctrl	12.540	4.443	12.540	4.443	0	0
I/O	256.331	90.833	0	0	14.592	0
Total	592.108	209.818	335.777	118.985	14.592	0
SecondStage (Texture mapping)						
Filter	342.057	121.210	72.860	25.818	0	5.440
Rwarp	146.440	51.892	146.440	51.892	0	0
Blend	167.941	59.51	80.184	28.414	4.352	0
HoleFill	390.680	138.440	81.444	28.860	0	7.072
Ctrl	40.862	14.479	40.862	14.479	0	0
I/O	479.311	169.848	0	0	23.104	0
Total	1567.294	555.384	421.793	149.466	27.456	12.512
Homography Table	191.808	67.968	0	0	14.784	**0**
Total	**2159.403**	**765.203**	**757.570**	**268.451**	**56.832**	**12.512**

The sub-modules “Preprocess,” “FWarp” and “RWarp” have high complexity for the matrix computation; while “Filter” and “HoleFill” are heavy for the FIFO-based control, which is implemented using 2-port SRAM. The I/O interface of the two top modules “FirstStage” and “SecondStage” are also large for the column-level 1-port SRAM.

5.6.3 Performance result

Table 5-5 Experimental sequences

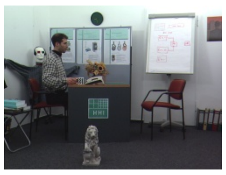
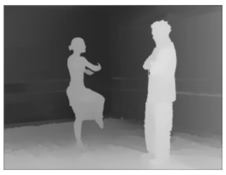
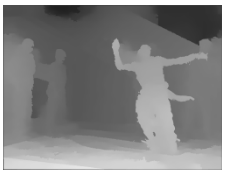
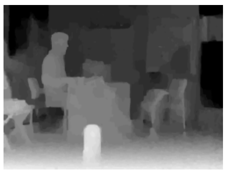
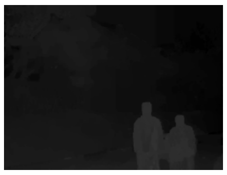
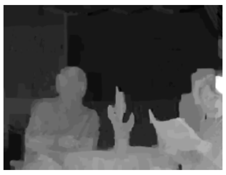
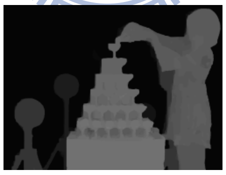
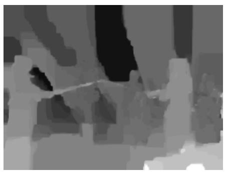
sequence name	Ballet	Breakdancers	BookArrival	Lovebird1
texture				
depth				
WxH	1024x768	1024x768	1024x768	1024x768
provider	MSR	MSR	HHI	ETRI
depth provider	provider	Provider	DERS_4_9	DERS_4_9
sequence name	**Newspaper**	**Champagne Tower**	**Kendo**	
texture				
depth				
WxH	1024x768	1280x960	1024x768	
provider	GIST	MPEG-FTV	MPEG-FTV	
depth provider	DERS_4_9	Provider	Provider	

Table 5-5 lists 7 test sequences used in the experiment. The "Ballet" and "Breakdancers" are from Microsoft research (MSR) [23]; the "BookArrival," "Lovebird1," and "Newspaper" are from Fraunhofer Heinrich-Hertz-Institut (HHI) [24], Electronics and Telecommunications Research Institute (ETRS), and Gwangju Institute of Science and Technology (GIST) respectively, and their depth map are estimated using DERS_4_9, which is provided by MPED-FTV; the "Champagne Tower" and "Kendo" are from MPEG-FTV.

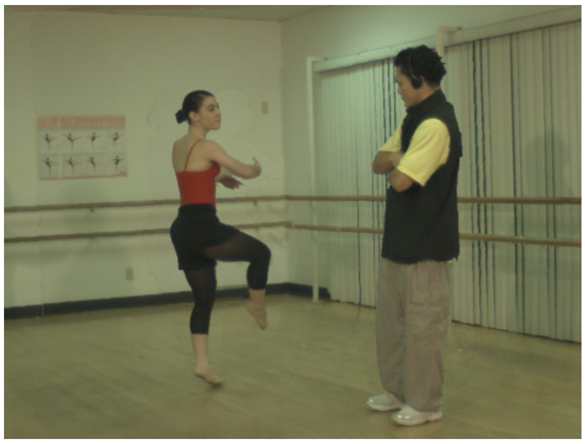
All sequences are run for 10 frames, and are evaluated by averaging the PSNR of each frame. The total performance is shown in Table 5-6. Three VSRS approaches are taken for each sequence of certain view and of certain 10 frames. The first approach is original VSRS_3_5 in the general mode of camera setting. And the second approach is VSRS_3_5 with inpainting replaced by bi-linear interpolation. The final is our hardware implementation design.

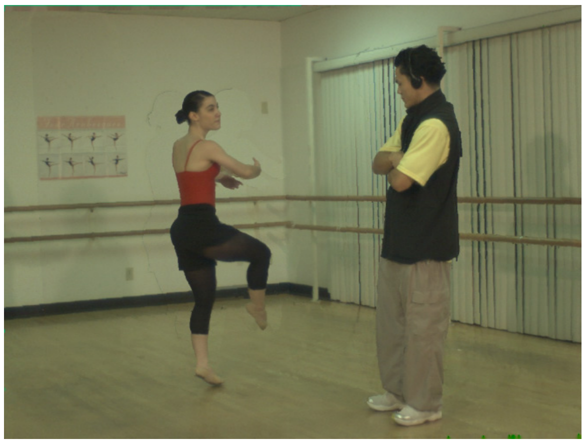
Table 5-6 Comparison of Average PSNR in 10 frames

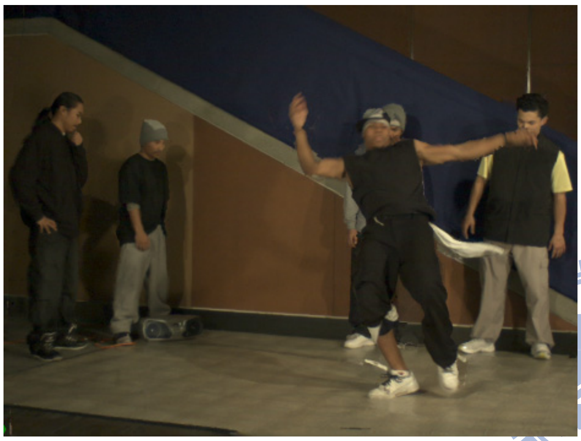
Performance **Average Y-PSNR(dB) in 10 frames**	**Original VSRS_3_5**	**Original VSRS_3_5 9x5 bi-linear interpolation**	**Our design**
Ballet, C5-C4-C3, f90-f99	33.081	33.208	33.372
Breakdancers, C5-C4-C3, f81-f90	32.984	33.166	33.121
BookArrival C10-C8-C7,f0-f9	36.385	36.371	36.499
lovebird1, C5-C6-C8, f0-f9	31.791	31.802	31.800
Newspaper, C3-C5-C6, f0-f9	30.683	30.677	30.778
Champagne, C37-C38-C39, f0-f9	33.367	33.367	33.361
Kendo C1-C2-C3, f0-f9	33.000	33.000	33.250
Average ΔPSNR compare VSRS_3_5	**0.000**	**0.043**	**0.127**

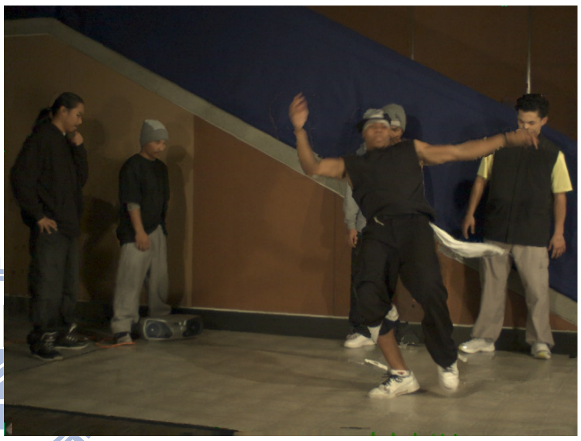
In the comparison between the result of first and the second approach, it indicates that the inpainting method can be substituted by the simple bi-linear interpolation without degrading the overall performance. Even some sequences have worse performance in bi-linear interpolation approach, the degrading PSNR are less than 0.014 dB.

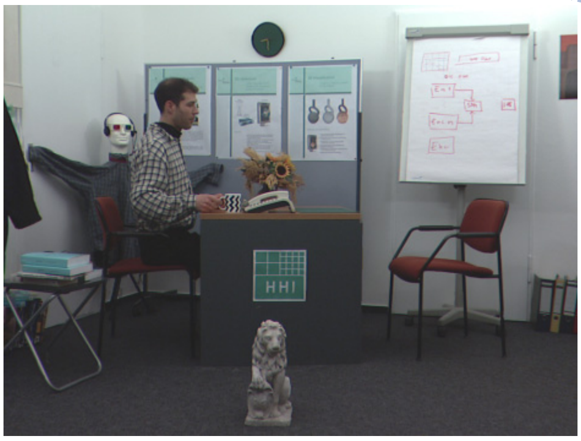
This table also shows that the overall PSNR performance of our hardware implementation approach is better than software approach. This may be because the sampling alias in 3D warping is less in hardware implementation. Nevertheless, if in subjective sense, instead of the objective PSNR evaluation, their performances are in fact nearly the same in Fig. 5-16 because our implementation is based on the software approach.

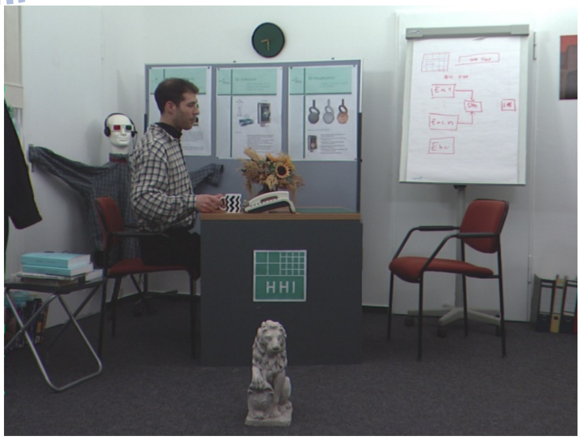

(a-1)Ballet, Our implementation


(a-2)Ballet, VSRS


(b-1)Breakdancers, Our implementation


(b-2)Breakdancers, VSRS


(c-1)BookArrival, Our implementation


(c-2)BookArrival, VSRS

(d-1)Lovebird1, Our implementation

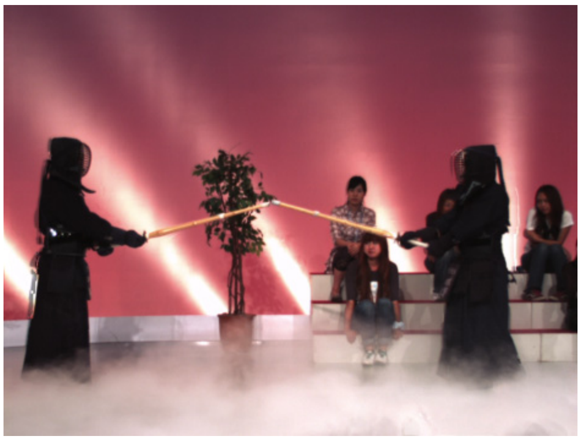
(d-2)Lovebird1, VSRS

(e-1)Newspaper, Our implementation

(e-2)Newspaper, VSRS

(f-1)Champagne Tower, Our implementation

(f-2)Champagne Tower, VSRS


(e-1)Kendo, Our implementation

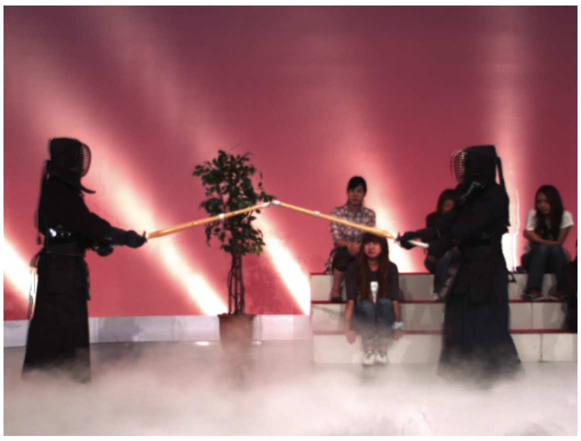

(e-2)Kendo, VSRS

Fig. 5-16 Experimental results in comparison to original VSRS results

Chapter 6

Conclusion

The major contribution in this thesis is the analysis and implementation of view synthesis engine (VS engine) based on the view synthesis reference software (VSRS) algorithm provided by MPEG-FTV.

This implementation includes the whole flow of the VSRS algorithm, from making projection matrix, homography estimation, 3D warping, post-filtering to blending and hole-filling. In global view, we proposed 2-stages frame-level pipelining architecture to implement 3D warping of depth and texture in memory concern; in local view, we used the column-level pipelining to make data access efficient, because the warped index is random among whole frame. We also analyzed the hole-filling interpolation method and proposed a simple bi-linear interpolation that is friendly for hardware implementation.

This design supports the video throughput of 58M pixels per second. In the UMC 90nm technology and at the 200MHz work frequency, the total hardware cost is 268.451K in gate counts and the total internal memory usage is 56.832KB for one-port SRAM and 12.512KB for two-port SRAM. Furthermore, the total bandwidth is 700Mbytes per second and the bus utilization is 43.75% by using 64-bits bus. The performance is good for it is without quality drop in the objective PSNR evaluation comparing to original VSRS algorithm.

Future work

We have implemented the whole VSRS algorithm as ASIC design that supports the general mode of camera settings, but the preprocess part that regarding to projection matrix and homography estimation is heavy, which has gate counts of totally 61.769K in our design. This part can be simplified if the cameras are set in a line, and the 3D warping method can then be substituted by simple horizontal shift approach. In this approach, the hardware cost will be further cost down because the computation is simpler and the large usage in memory will also be saved. This adjustment is acceptable because actually, most videos are fetched in the parallel camera settings in the nowadays MPEG-FTV study.

In addition, about 64.9% hardware cost is from the large usage in internal memory. In our design, we choose the size of I/O buffer as column-level for best bus efficiency. But we can indeed reduce the size to 16bytes or 32 bytes that with bus efficiency 85.6% and 85.7% in average and only decrease 3.1% compared to column size buffer as list in Table 4-3. By decrease the I/O buffer size to 32KB, the size of one-port SRAM can be reduced from 56.832KB to 19.968 KB.

Furthermore, for higher specification in today's visual entertainment, the throughput usually is 60 fps for HD1080p videos. Because our design takes turns to processing left-view video and right-view video in a column-level stage, we can improve our design by doubling the combinational logics to process two views simultaneously so that the throughput could be doubled.

Reference

[1] C. Fehn, “Depth-image-based rendering (DIBR), compression and transmission for a new approach on 3D-TV, ” in Proc. of SPIE conf. on Stereoscopic Displays and Virtual Reality Systems, vol. 5291, pp. 93-104, May 2004.

[2] M. Tanimoto and M. Wildeboer, “Framework for FTV coding,” in Proc. of the 27th conference on Picture Coding Symposium, pp.429-432, 2009.

[3] P. Kauff, N. Atzpadin, C. Fehn, M. Müller, O. Schreer, A. Smolic, and R. Tanger. “Depth map creation and image-based rendering for advanced 3DTV services providing interoperability and scalability,” Signal Processing: Image Communication. Special Issue on Three-Dimensional Television and Applications, 22(2), pp.217-234, Feb. 2007.

[4] Y. Morvan, “Acquisition, compression and rendering of depth and texture for multi-view video,” in ISBN: 978-90-386-1682-7, Apr. 2009.

[5] M. Tanimoto, T. Fujii, K. Suzuki, N. Fukushima, and Y. Mori, “Reference softwares for depth estimation and view synthesis,” MPEG Doc. M15377, 2008.

[6] M. Tanimoto, T. Fujii, and K. Suzuki, “View synthesis algorithm in view synthesis reference software 2.0 (VSRS2.0),” MPEG Doc.M16090, Feb. 2008.

[7] Y. Mori, N. Fukushima, T. Fuji, and M. Tanimoto, “View generation with 3D warping using depth information for FTV,” 3DTV-CON’08, Istanbul, Turkey, May 2008.

[8] M. Gotfryd, Krzysztof Wegner, Marek Domański, “View synthesis software and assessment of its performance, ” MPEG Doc. M15672, Hannover, Germany, July 2008.

[9] K. Müller, A. Smolic, K. Dix, P. Merkle, P. Kauff, and T. Wiegand, “View synthesis for advanced 3D video systems”, EURASIP Journal on Image and Video Processing, Volume 2008.

[10] C. Vázquez, W. J. Tam, F. Speranza, Stereoscopic imaging: Filling disoccluded areas in image-based rendering, Proc. SPIE Three-Dimensional TV, Video, and Display (ITCOM), Vol. 6392, Boston, MA, 2006.

[11] C.M. Cheng, S.J. Lin, S.H. Lai and J.C Yang, “Improved novel view synthesis from depth image with large baseline,” IEEE International Conference on Pattern Recognition, pp.1-4, 2008.

[12] L.Zhang, and W. J. Tam, “Stereoscopic image generation based on depth images for 3D TV,” IEEE Trans. Broadcasting, vol. 51, pp. 191-199, June 2005.

[13] W.-Y. Chen, Y.-L. Chang, S.-F. Lon, L.-F. Ding, and L.-G. Chen, “Efficient depth image based rendering with edge dependent depth filter and interpolation,” in Proc. of IEEE International Conf. on Multimedia and Expo, pp. 1314-1317, July 2005.

[14] Y. K. Park, K. Jung, Y. Oh, S. Lee, J. K. Kim, G. Lee, H. Lee, K. Yun, N. Hur, and J. Kim, “Depth-image-based rendering for 3DTV service over T-DMB, ” in Signal processing: Image communication, vol. 24, pp. 122-36, Jan. 2009.

[15] A. Telea “An image inpainting technique based on the fast marching method,” Journal of Graphics Tools, vol. 9, no. 1, pp.25-36, 2004.

[16] K. J. Oh, Y. Sehoon, Y. S. Ho,, "Hole-Filling Method Using Depth Based In-Painting for View Synthesis in Free Viewpoint Television (FTV) and 3D Video", Picture Coding Symposium (PCS), May 2009.

[17] P. C. Lin, P. K. Tsung, and L. G. Chen, “Low-cost hardware architecture design for 3D warping engine in multiview Video,” in Proc. of IEEE International Symposium on Circuits and Systems, 2010.

[18] L. McMillan. “An image-based approach to three-dimensional computer graphics.” PhD thesis, University of North Carolina, Chapel Hill, USA, Apr.1997.

[19] R. P. Brent, F. T. Luk and C. F. Van Loan, “Computation of the singular value decomposition using meshconnected processors”, Journal of VLSI and Computer Systems 1, 3, pp. 242–270 1983–1985.

[20] J. R. Cavallaro and F. T. Luk, "CORDIC Arithmetic for an SVD Processor," Journal of Parallel and Distributed Computing, vol. 5, no. 3, pp. 271-290, 1988.

[21] R. Andraka, “A Survey of CORDIC Algorithms for FPGAs,” in Proc. of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays (FPGA ’98),Monterey, CA, Feb. 22–24, pp. 191–200, 1998.

[22] W. Cheney and D. Kincaid, "Numerical Mathematics and Computing," sixth edition, Thomson Brooks/Cole Publishing Company, 2008

[23] C.L. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," ACM SIGGRAPH and ACM Trans. on Graphics, Los Angeles, CA, pp. 600-608, Aug. 2004.

[24] I. Feldmann, M. Mueller, F. Zilly, R. Tanger, K. Mueller, A. Smolic, P. Kauff, and T. Wiegand, “HHI Test Material for 3D Video,” MPEG Doc. M15413, 2008.

[25] Tanimoto Laboratory. (2008). MPEG-FTV [Online]. Available: http://www.tanimoto.nuee.nagoya-u.ac.jp/mpeg/mpeg_ftv.html.