Plenoptic Modelling and Rendering of Complex Rigid Scenes

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Jan-Friso Evers-Senne

Kiel

2008

1. Gutachter	: Prof. Dr.-Ing. Reinhard Koch
2. Gutachter	: Prof. Dr. Andreas Kolb
Datum der mündlichen Prüfung	: 17.6.2008

Danksagung

An dieser Stelle möchte ich denen danken, die mich bei der Anfertigung dieser Arbeit direkt oder indirekt unterstützt haben. Als erstes gilt mein Dank Prof. Dr.-Ing. Reinhard Koch, zum einen für die überlassung dieses interessanten Themas, vor allem aber für die Jahre der tollen Zusammenarbeit in seinem Team.

Meinen ehemaligen Kollegen, den wissenschaftlichen Mitarbeitern der Arbeitsgruppe MIP, namentlich Kevin Koeser, Ingo Schiller, Dr. Daniel Grest, Dr. Christian Beder, Bogumil Bartczak, Dr. Jan-Michael Frahm, Dr. Felix Woelk und Jan Woetzel, danke ich für ihren Teamgeist und die stets spannenden fachlichen Diskussionen sowie die nicht-fachlichen Exkurse.

Renate Staeker danke ich für die stetige Hilfsbereitschaft bei der Bewältigung organisatorischer Dinge und Torge Storm danke ich für die vielen Basteleien, ohne die kein Projekt hätte durchgeführt werden können.

Meine Eltern gilt besonderer Dank dafür, dass sie mir diese wissenschaftliche Ausbildung ermöglicht haben. Ebenso möchte ich mich bei meiner Großmutter und meinen Schwiegereltern für die Unterstützung bei der Umsetzung meiner Vorhaben bedanken.

Bei meiner Frau Eka und meinen Kindern Maarit, Phyllis und Kjell möchte ich mich für die dauernde Unterstützung, Geduld und Leidensfähigkeit während der Entstehung dieser Arbeit bedanken.

Contents

Chapter 1

Introduction

The virtual representations of scenes from reality belongs to the most fascinating aspects in Computer Graphics. They can be used in many different applications reaching from mixed-reality film productions to interactive virtual reality applications and games. In the last few years, the visual quality of such virtual scenes and models has increased dramatically. The driving force behind this effect has been the developement in computational power. While the evolution of main memory and CPUs has followed Moore's Law quite precisely, the performance growth of GPUs (Graphics Processing Units) and graphics cards has been even quicker. The number of elementary functions on a GPU has doubled every 10-12 months. Increased performance and graphics memory allow the user to render increasingly complex scenes. A major drawback of every handcrafted virtual model is that it looks realistic only on the first glance. On closer examination, the level of detail does not yet suffice. Most natural scenes are far too complex to be modelled manually.

Image-Based Rendering or IBR tries to close the gap between near-realistic scenes and reality. The basic idea behind IBR is to utilise images of real-world scenes or objects for the generation of new views with the computer.

1.1 Motivation

The main motivation for this thesis was to develop techniques for visual geometric reconstruction and the rendering of arbitrary scenes. Such techniques may be used in applications which need virtual representations of scenes from reality. Modelling complete scenes including background, not isolated objects, is of primary interest.

In virtual film productions, for example, it is a typical task to place a purely virtual object into a real scenery and provide correct shadows, lighting and oc-

clusion. Other mixed-reality applications like virtual studios have similar requirements. The ability to generate images of a scene from viewpoints wich are not identical with the original position of the cameras can be used for free-viewpoint applications like 3D-Games or interactive free-viewpoint television.

The creation of virtual models from single objects is related to the creation of scene models but it has to meet some special requirements. Object modelling has been studied in great detail for decades and has reached a high level of maturity while background- or environment modelling is still at an early developement stage. All the approaches presented in this thesis are focused on modelling scenes, but they are not limited to this purpose and can also be used to model objects.

The development of the new Image-Based Rendering Methods, which will be presented in this thesis, is guided by the following requirements:

- automatic modelling and rendering from image sequences taken by standard digital cameras,
- no user interaction besides defining the new viewpoint,
- highly scalable with respect to the number of input images,
- scalable between rendering speed and visual quality,
- combine sparse sampling (distance between images), full parallax and non-restricted movement,
- support depth information to allow embedding of virtual objects with correct occlusion.

1.2 Graphics vs. Vision

The research field of Image-Based Rendering resides, by its nature, in between Computer Graphics and Computer Vision.

The primary goal of Image-Based Rendering is to generate a new image with the use of mathematical methods. So Image-Based Rendering is a part of Computer Graphics. Tasks such as rendering virtual 3D objects to generate 2D images, computing direction and intensity of rays of light, transforming complex data sets to visually understandable representations (e.g. simulation data) may serve as examples of what Computer Graphics may achieve.

In contrast to Computer Graphics, Computer Vision aims to interpret input data from sensors observing parts of the electromagnetic spectrum. Most often this means 2-dimensional (colour)-images, but X-ray images or 1-dimensional images are also used in Computer Vision.

Finding mathematical methods to transfer visual representations from (real) input images to (virtual) new images obviously requires knowledge from both fields: Computer Vision and Computer Graphics.

Figure 1.1 sketches the combination of graphics and vision. Starting with 2D images, Computer Vision algorithms are used to calibrate the images and to reconstruct the depth information. Methods from Computer Graphics then create geometrical approximations of the scene from calibrated images and depth maps (modelling). Finally, Computer Graphics algorithms are used to project the geometry into new viewpoints (rendering). As mentioned above, Image-Based Render-

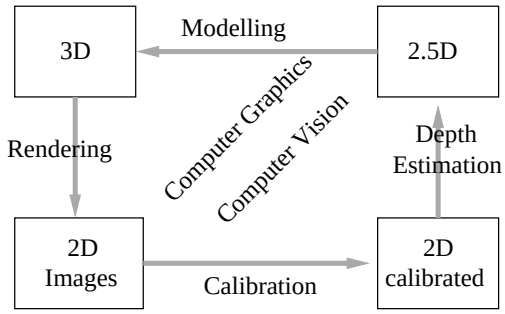


Figure 1.1: Image-Based Rendering uses techniques from Computer Vision and Computer Graphics.

ing uses only some aspects from both fields. Other aspects like object recognition (Computer Vision), ray tracing or sophisticated lighting (Computer Graphics) are not relevant.

Image-Based Rendering is a techniques different from those used in current animation movie productions. For these, all scenery consisting of 3D models, textures, lights, camera positions and definitions of movements are modelled manually. Computer Graphics algorithms, namely ray tracing, are then used to create very detailed highly realistic looking 2D images.

Image-Based Rendering methods compute new images based on a set of input images. The new images show the same scene as the input images, but from different perspectives.

1.3 Goals and Contributions of this Thesis

When examining the known techniques of Image-Based Rendering it becomes clear that most systems are tailored for very special scenarios and setups. The goal of this thesis is to develop a more versatile Image-Based Rendering approach. It should be scalable in various dimensions to fit the specific needs of different applications. The following requirements have to be fulfilled:

- Not only isolated objects, but also complete scenes including the background should be handled. This implies that the results should not depend on the complexity of the scene.
- The image generation should work with interactive frame-rates. If this is not possible for best quality results, the system should be scalable between quality and speed.
- View generation should only be restricted by the scene coverage from the real views. The virtual camera can be moved freely and should not be limited to rotation only or certain special trajectories.
- It should be possible to generate new views from only a few real images of arbitrary distance but also from a a set of several thousand images covering a single scene.
- No direct geometrical information, e.g. a 3D-model, should be required. If geometrical information is required, this has to be computed from the images only.
- It should be possible to add additional 3D-objects into the generated images, preserving correct occlusions. Thus, depth information has to be provided for each new image.

Following these requirements, three different Image-Based Rendering-methods with specific features have been designed. According to these methods, prototypes of Image-Based Rendering-systems have been implemented. Prt of this work was already presented at:

J.-F. Evers-Senne and R. Koch. Image Based Interactive Rendering with View Dependent Geometry . In *Eurographics 2003*, Computer Graphics Forum, pages 573–582. Eurographics Association, 2003.

J.-F. Evers-Senne and R. Koch. Interactive rendering with view-dependent geometry and texture. In *Sketches and Applications SIGGRAPH 2003*, 2003.

J.-F. Evers-Senne and R. Koch. Image Based Rendering from Handheld Cameras using Quad Primitives. In *Vision, Modeling, and Visualization VMV: proceedings*, Nov. 2003.

J.-F. Evers-Senne, J. Woetzel, and R. Koch. Modelling and Rendering of Complex Scenes with a Multi-Camera Rig. In *1st European Conference on Visual Media Production (CVMP 2004), London, United Kingdom*, pages 11–19, March 2004.

J.-F. Evers-Senne, A. Niemann, and R. Koch. Visual reconstruction using geometry guided photo consistency. In *Vision, Modeling, and Visualization VMV: proceedings*, Aachen, Germany, November 2006.

One part of this thesis (chapter 3) has previously been published as chapter "3D View Synthesis" in:

R. Koch and J.-F. Evers-Senne. View synthesis and rendering methods. In *3D Video Communications*. Wiley, 2005.

1.4 Structure of this Thesis

This thesis is structured in 7 chapters as follows:

In chapter 2 mathematical and technical background knowledge of imaging devices and the theory of plenoptic sampling are introduced. Different image blending strategies and blending weight computations are described. In addition, a short introduction into programmable graphics hardware is also provided.

In chapter 3 related work in the field of Image-Based Rendering and View Generation is reviewed. A taxonomy of the different approaches and systems is introduced which allows to classify systems based on common criteria.

Chapter 4 discusses the design goals and requirements for the development of Image-Based Rendering systems. The necessity of depth information is examined and after the description of calibration and depth estimation, typical sources of errors in these preprocessing steps are analysed.

Chapter 5 presents the three major Image-Based Rendering methods, which have been implemented as prototypes in the context of this work. The tasks which are common for all methods, namely camera selection and blending, are also described.

In Chapter 6 all implemented prototypes are analysed. Different error categories are introduced and metrics are developed to evaluate the quality of view generation. The modelling and rendering systems are tested on various input data ranging from fully synthetic to good-quality real footage and medium-quality data from high-speed capturing and processing systems. This procedure allows to compare the theoretical limits and to demonstrate the usability of the methods with realistic data.

Finally, chapter 7 will summarise this thesis. The particular positive and negative characteristics off all methods will be reviewed briefly. The discussion tries to find answers to the following questions: Which method is best suited for what purpose? What are the requirements for the input data? What is the tradeoff between means of quality, performance and robustness? The conclusion will point

out, which problems have been solved and what questions remain open for future research.

1.5 Symbols

In this thesis, mathematical entities from 2D-, 3D-, Euclidean and Homogenous spaces are used. The nomenclature is similar to that used by Hartley and Zissermann [2004]. In general, math bold face like $\mathbf{M}, \mathbf{m}$ indicates a vector or point, lower case indicates (Homogeneous) 2D, upper case indicates (Homogeneous) 3D. foobarbarfoofoobar The upper-case character 'C' is used for two different purposes. C is used to name a specific camera as in C_0, C_1, in math-bold face $\mathbf{C}$ it denotes the projection center of a camera.

$\mathbf{M} = (x,y,z)^T$	Euclidean 3D point
$\mathbf{m} = (x,y)^T$	Euclidean 2D point
$\tilde{\mathbf{M}} = (x,y,z,w)^T$	Homogeneous 3D point
$\tilde{\mathbf{m}} = (x,y,w)^T$	Homogeneous 2D point
H	Projective transformation/homography
π	Plane
π_∞	Plane at infinity
K	Camera calibration matrix
$\mathbf{C}$	Camera projection centre
R	Camera rotation matrix
P, $P = K[R^T \mid -R^T\mathbf{C}]$	Camera projection matrix
$\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_N$	Corresponding 2D points
C_i	Camera number i

Table 1.1: List of used symbols.

Chapter 2

Theoretical Background

In this chapter, the mathematical background of image formation is described. The process of taking 2D images from the 3D scene with a camera is explained as sampling the Plenoptic Function. Disparity, depth and their relationship are introduced and image blending strategies are presented. The basic principles of programmable graphics hardware are introduced here, too, because Image-Based Rendering as part of Computer Graphics can make use of the graphics hardware available today.

2.1 The Plenoptic Function

The term *Plenoptic Function* (*PF*) of a 3D scene, introduced by Adelson and Bergen [1991], is derived from the Latin *plenus* = full and *optic* which relates to vision. It describes the intensity of all irradiation observed at every point in the 3D scene coming from every direction; for an arbitrary dynamic scene the plenoptic function is of dimension 7.

$$\text{Plen}_{\text{full}} : \mathbb{R}^3 \times [0, 2\pi)^2 \times \mathbb{R}^+ \times \mathbb{R} \rightarrow \mathbb{R} \text{ , } \text{Plen}_{\text{full}}(x, y, z, \phi, \theta, \lambda, t) = I \qquad (2.1)$$

I is the light intensity of the incoming light rays at a spatial 3D-point $(x, y, z)^T$ from a direction given by spherical coordinates (ϕ, θ) for a wavelength λ at a time t. If the *PF* is known to its full extent, then the visual scene appearance can be reproduced precisely from any viewpoint at any time. Obviously, it is technically not possible to record an arbitrary *PF* of full dimensionality. This would make it necessary to simultaneously place light probes to fully cover the space permanently.

2.1.1 Sampling the Plenoptic Function

To record an arbitrary scene we need to sample the high-dimensional *PF*. The sampling problem can be simplified if we separate the 4 spatio-temporal dimensions (x,y,z,t) from the dimensions (ϕ,θ,λ) for viewing direction and colour sensing by using spherical colour image sensors. As a *plenoptic sample (PS)* we define the three-dimensional subspace that forms a spherical image $I_{ps}(\phi,\theta,\lambda)$ at a particular spatio-temporal position (x_t,y_t,z_t)

$$I_{ps} : [0,2\pi)^2 \times \mathbb{R}^+ \rightarrow \mathbb{R} \text{ , } I_{ps}(\phi,\theta,\lambda)|_{x,y,z,t} = I \tag{2.2}$$

The set of all plenoptic samples $I_{ps}(\phi,\theta,\lambda)$ for all spatial dimensions (x,y,z) and for all times t form the full *PF* (2.1).

In a stationary scene, the dependency on time t can be eliminated and we can move a single light probe over time to different spatial positions (x_t,y_t,z_t) and record the stationary plenoptic field sequentially:

$$\text{Plen}_{\text{stationary}} : \mathbb{R}^3 \times [0,2\pi)^2 \times \mathbb{R}^+ \rightarrow \mathbb{R} \text{ , } \text{Plen}_{\text{stationary}}(x_t,y_t,z_t,\phi,\theta,\lambda) = I \tag{2.3}$$

Now we can define the spatial *PF* as collection of all *PS* in space:

$$\text{Plen}_{\text{spatial}} : \mathbb{R}^3 \rightarrow I_{ps} : [0,2\pi)^2 \times \mathbb{R}^+ \rightarrow \mathbb{R} \text{ , } \text{Plen}_{\text{spatial}}(x_t,y_t,z_t) = I_{ps}(\phi,\theta,\lambda) \tag{2.4}$$

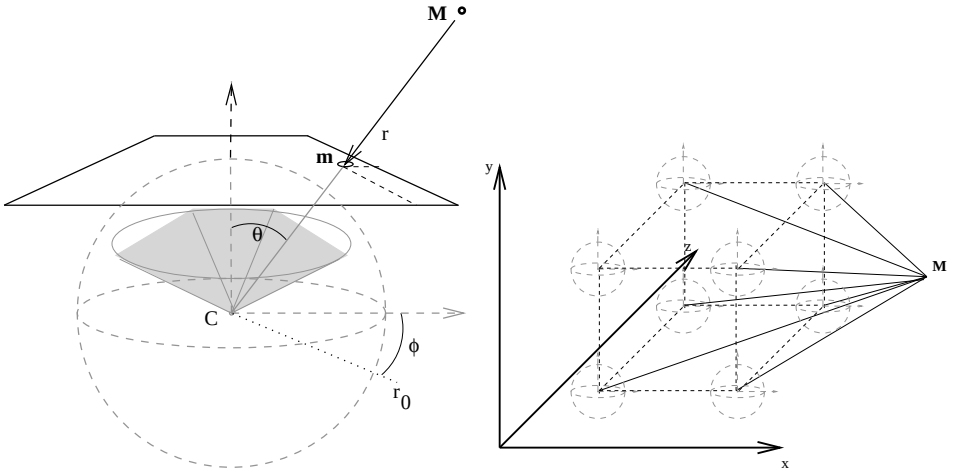
A particular plenoptic sample is obtained by placing a spherical imaging sensor in space and recording the light intensity for each incoming ray. Direction angles ϕ and θ are discretised with pixel positions on the sphere. Colour perception is obtained by recording three separate images for red, green and blue according to the human tristimulus perception [Hunt, 1998]. The smoothed λ is finally discretised with three samples.

A conventional perspective camera can be modelled as a tangent plane to the sphere with a limited field of view as shown in figure 2.1. Thus, the sampling of the *PF* is reduced to recording colour images in each spatial position. If the scene is non-stationary, an image sequence can be taken to capture the time variation of a plenoptic sample. Sampling of equation 2.4 with real sensors introduces discretisation on three levels:

1. Angular sampling (ϕ,θ) of a single plenoptic sample due to the finite pixel resolution of the imaging sensor,

2. spatial sampling (x_t,y_t,z_t) due to the finite sampling density between plenoptic samples,

3. colour quantisation by using red, green and blue band-pass filters to sample the visible spectrum (λ)



(a) Spherical plenoptic sample. (b) Spatial distribution of PS.

Figure 2.1: a) An ideal spherical sensor at position $C = (x,y,z)^T$ sampling a ray r from **M** with (ϕ,θ). The image plane of a perspective sensor is tangent to the sphere giving an image point **m**. b) Eight Plenoptic Samples organised in a regular grid observing a point **M**.

It is therefore necessary to obey the sampling theorem [Unbehauen, 2002] to avoid aliasing. Angular sampling is usually not a problem. The discretisation is implied by the pixel raster of the photo sensitive device which gives one luminance value for each pixel position. But each photo sensitive cell occupies a given area over which incident light rays are integrated and averaged. As an assumption that the cells are packed closely together, this ensures that no aliasing can occur. In addition, every lens has a finite impulse response, which means that thin structures like points are spread out a bit. This so-called *point spread function* functions as a Gaussian filter removing high frequencies.

Spatial sampling may pose a problem if a large viewing volume needs to be sampled. Therefore, the IBR systems have to distinguish between dense and sparse sampling of the *PF* in the following. Dense sampling eliminates parallax effects but may cause (highly redundant) oversampling. Sparse sampling will violate the sampling theorem, hence additional information for correct interpolation, scene depth for example, will be necessary. Chai et al. [2000] have evaluated the effects of sampling density and parallax influences on the rendering for the light field problem in detail and point out limits for the sampling density as a function of the depth variations in the scene.

2.1.2 Recording of Plenoptic Samples

As shown above, plenoptic samples are recorded by capturing images at specified spatial positions. Recently, a wealth of spherical and hemispherical imaging sensors have become available, e.g. catadioptic (mirror-optic) devices or wide-angular fish-eye lenses that directly capture $I_{(\phi,\theta,\lambda)}$ for a hemisphere. Furthermore, the theory of the image formation of such sensing devices is now well understood, see Baker and Nayar [1998], Geyer and Daniilidis [2003] or Bakstein and Pajdla [2003a]. These hemispherical sensors allow direct recording of the *PS* with low angular resolution [1]. If high resolution samples are needed, the spherical sample can be reconstructed by mosaicing of multiple rotated images into a spherical panorama. Many systems have been developed for direct image mosaicing, called also rotational mosaics [Shum and Szeliski, 1997], or with the use of motorised camera heads, as used for Quicktime VR systems [Chen, 1995].Very high-resolution cylindrical panoramas can be recorded with rotating high-resolution line-sensors that generate sections of up to 30.000 x 100.000 Pixel for a single plenoptic sample [Klette et al., 2003]. The spherical image can be mapped onto a perspective view in order to display the panorama with a conventional screen.

Plenoptic samples taken from the scene are denoted as *Real Views* in the following, to distinguish them from a *Virtual View*, which is synthesised by reconstructing the *PF*. The word *Camera* is used in this context to refer to the internal and external parameters (the projection) which have been used for a view. Thus, a *Real Camera*(denoted as C_0, C_1,..) is associated with a specific *Real View* and a *Virtual Camera* (denoted as C_v) is associated with every *Virtual View*.

2.1.3 Re-Sampling

Due to the given pixel quantisation of real cameras, view generation is a re-sampling process. Problems can occur if the resolution of input and output images differ. For example, if a new view shows a magnification of the scene (figure 2.2), forward-mapping of all pixels of a real view does not fill all pixels in the novel view. In-between pixels have to be interpolated which implies that details not visible in the original image cannot appear in the new image. If we assume that the target resolution is lower than the resolution of the input data, appropriate filtering has to be performed before sub-sampling the image. Otherwise the sampling theorem will be violated resulting in aliasing artefacts.

[1] Projecting a complete hemisphere onto a given sensor yields a lower angular resolution than projecing with a perspective lens with smaller field of view.

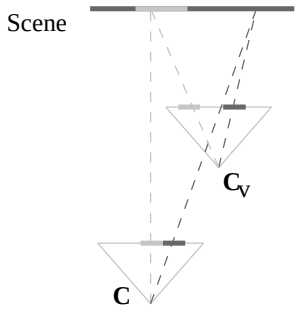


Figure 2.2: The new image (C_v) shows a close-up of the original image (C). Without any additional information, interpolation has to be used to fill the gaps between pixels from C.

2.2 Projection Models

Real (physical) cameras project 3D scenes into 2D images. The next section now describes the modelling of imaging devices starting with the perspective projection based on the pinhole camera model. Then the projection of hemispherical (so called fish-eye) lenses is discussed.

2.2.1 The Perspective Projection

For perspective cameras the projection of points from 3D into 2D images can be modelled with the *pinhole camera model*. In this model, rays of light from the object passing through the projection centre intersect the image plane. This projection centre is located in the origin of the coordinate frame and the image plane is located in $z = -1$. This is an idealised representation of the "camera obscura", where all light comes through a single hole of a box. The idealisation is that the hole is of infinitely small size and that it is always located in zero. The image of a camera obscura is flipped horizontally and vertically. In the pinhole camera model, the image plane can be located at $z = 1$ instead, which results in an up-right image.

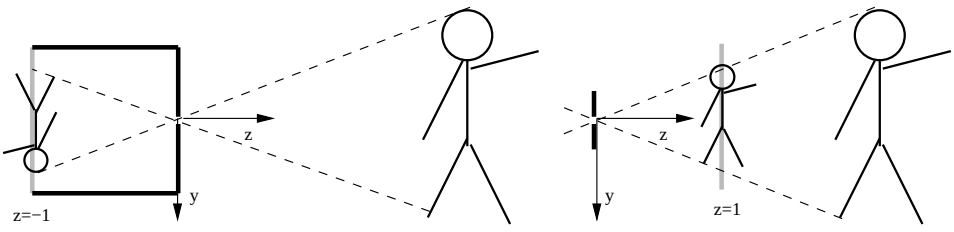


Figure 2.3: Camera-Obscura and Pinhole Camera Model

Since all data which are to be processed or generated by computers are discretised, 2D images are discretised, too. This means that the image plane of a camera is divided into a regular grid of cells called *pixel* (from *Pi*cture *El*ement). Typically those are addressed in row-major order from the top-left of the image.

The pinhole projection onto a discretising 2D sensor is described by a *Camera Matrix*. The projection of one euclidean scene point $\mathbf{M} = (x,y,z)^T$ is given by:

$$\mathbf{m} = K\mathbf{M} = \begin{pmatrix} f & s & c_x \\ 0 & a \cdot f & c_y \\ 0 & 0 & 1 \end{pmatrix} \mathbf{M}. \tag{2.5}$$

$\mathbf{m} = (\tilde{x},\tilde{y},\tilde{z})^T$ is a three dimensional vector describing the ray from the center of projection (the origin) to $\mathbf{M}$. Because the image plane of the pinhole camera model is located in $z = 1$, $\mathbf{m}$ has to be divided by $\tilde{z}$ to get the pixel coordinate (x',y'):

$$\mathbf{m} = \lambda \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} \tilde{x}/\tilde{z} \\ \tilde{y}/\tilde{z} \\ 1 \end{pmatrix}, \text{with: } \lambda = \tilde{z} \tag{2.6}$$

This is called the *perspective division* and λ is the *projective depth*, the distance of $\mathbf{M}$ to the centre of projection. Using image sensors measuring only the 2D pixel position $(x',y',1)^T$, this perspective depth λ gets lost. This reduces the dimension from 3 ($\mathbf{M}$) to 2 for $\mathbf{m}$. $\mathbf{m}$ can also be interpreted as being a homogeneous 2D point $(x',y',1)^T$.

The matrix K is an upper triangular matrix where f (*focal length*) is the scale from the world coordinate system to the image coordinate system of the camera expressed in pixel units. The *aspect ratio a* is the ratio between the length of a pixel in x-direction and the length of a pixel in y-direction, in other words, the ratio between the focal length in x and y direction. The *principal point* of the camera (c_x,c_y) describes the offset of the image coordinate system from the origin of the world coordinate system. The *skew s* is a parameter which models the angle between columns and rows of the sensor. The parameters f,a,s,c_x,c_y are the so-called *intrinsic camera parameters*.

2.2.2 Extrinsic Camera Parameters

So far, the camera has been located in the origin of the (local) coordinate frame and has been oriented towards the z-axis. To allow arbitrary camera positions and orientations, the camera model has to be extended by the *extrinsic camera parameters*. These consist of a 3-dimensional vector $\mathbf{C}$ for the location of the camera centre in 3D Euclidean space and the rotation of the camera given by a 3x3 matrix R.

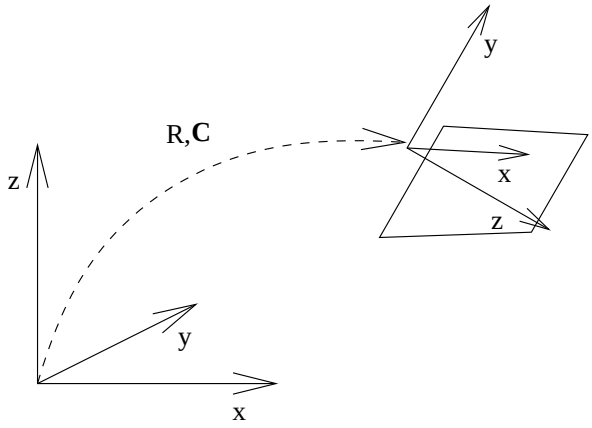


Figure 2.4: The extrinsic parameters **C** and R map from world coordinate system to camera coordinate system.

The rotation matrix R consists of three vectors $\mathbf{H},\mathbf{V},\mathbf{A}$ which describe the axis of the camera coordinate system in relation to the world coordinate system:

$$R = [\mathbf{HVA}] = \begin{pmatrix} h_1 & v_1 & a_1 \\ h_2 & v_2 & a_2 \\ h_3 & v_3 & a_3 \end{pmatrix}. \tag{2.7}$$

H and **V** span the image plane while **A** is the viewing direction of the camera. R is an orthonormal matrix which means that $R^{-1} = R^T$ holds. In 3D space, any rotation can be expressed by one rotation axis and a rotation angle around this axis. Also very common is a representation using three concatenated rotations around three orthogonal axes with three angles. If we choose the coordinate axes as rotation axes, the angles are called *Euler angles*. The matrices for rotation around the $x-$,$y-$,and z-axis are defined by:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix} R_y = \begin{pmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{pmatrix} R_z = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

The concatenation defines the order of rotation and if the rotation axis are rotated or fixed. On the assumption that a camera is first rotated around the pan-axis (y), then around the tilt-axis (x) and finally around the roll-axis(z) and that in each step the remaining axes are also rotated, the rotation is:

$$R = R_z R_x R_y \tag{2.9}$$

Combining R and **C** into one 3×4 matrix $[R|\mathbf{C}] = P_{\text{ext}}$ and extending the euclidean point $\mathbf{M} = (x,y,z)^T$ to a homogeneous 3D point $\tilde{\mathbf{M}} = (x,y,z,1)^T$ allows to formulate an affine transformation from $\tilde{\mathbf{M}}'$ to $\tilde{\mathbf{M}}$:

$$\tilde{\mathbf{M}} = \begin{bmatrix} R & | & \mathbf{C} \\ \mathbf{0^T} & | & 1 \end{bmatrix} \cdot \tilde{\mathbf{M}}', \text{ with: } \tilde{\mathbf{M}} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \tilde{\mathbf{M}}' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} \tag{2.10}$$

While $(R|\mathbf{C})$ defines the transformation from the camera coordinate system to the world coordinate system, the inverse transforms the world coordinate system back into the camera coordinate system. This inverse transformation can be used to transform the camera back into the origin and it can also be applied to all homogeneous 3D points **M**.

$$\tilde{\mathbf{M}}' = \begin{pmatrix} R^T & | & -R^T\mathbf{C} \\ 0 & | & 1 \end{pmatrix} \cdot \tilde{\mathbf{M}}. \tag{2.11}$$

With a camera located in **C**, rotated by R and a point **M** observed, the inverse transformation can be applied before projecting the point with the camera matrix K and λ the projective depth as in eq. (2.6):

$$\lambda\, \tilde{\mathbf{m}} = K(R^T | -R^T\mathbf{C}) \cdot \tilde{\mathbf{M}}. \tag{2.12}$$

Combining the camera matrix and extrinsic parameters allows to describe the projection of a pinhole camera at an arbitrary position and orientation with one single projection matrix. The rank-3 projection matrix P is the combination of $(R^T | -R^T\mathbf{C})$ and K:

$$\lambda\, \tilde{\mathbf{m}} = P\, \tilde{\mathbf{M}} \quad \text{with the projection matrix } P = K(R^T | -R^T\mathbf{C}). \tag{2.13}$$

2.2.3 The Equidistant Projection

The equidistant projection is a non-perspective projection implemented by so called *fish-eye*-lenses. They typically have a very wide field-of-view, most often in a range between 160 and 190 degrees, while perspective lenses typically do not exceed 100 degrees.

Concerning the general plenoptic sample as introduced in section 2.1.1, the direction is parametrised by a pair of angles (ϕ, θ). A point $\mathbf{M} = (x,y,z)^T$ located on a unit sphere around **C** defines a ray of incoming light (see figure 2.1). The angles ϕ and θ are then defined as:

$$\phi = \arctan\left(\frac{y}{x}\right) \quad \text{and} \quad \theta = \arctan\left(\frac{\sqrt{x^2+y^2}}{z}\right), z \neq 0 \tag{2.14}$$

By placing an image plane tangential to the unit sphere with an offset of its origin so that the optical axis intersects at the principal point $(c_x, c_y)^T$, any image point $\mathbf{m}$ can be parametrised in polar coordinates using a radius $r(\theta)$ and the angle ϕ:

$$\mathbf{m} = r(\theta) \cdot \begin{pmatrix} \cos(\phi) \\ \sin(\phi) \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix} \tag{2.15}$$

What remains open is the relation between θ and r, the function $r(\theta)$. At this point, the spherical projection differs from the perspective projection. In the perspective case, the function is defined as:

$$r_{\text{persp}}(\theta) = f \tan(\theta), \tag{2.16}$$

with f a constant scale factor. This leads to the perspective division when inserting equation (2.14) with f equivalent to the focal length:

$$r_{\text{persp}}(\theta) = f \frac{\sqrt{x^2 + y^2}}{z} \tag{2.17}$$

For equidistant projection the relation between θ and r is linear with a constant scale k:

$$r_{\text{equidist}}(\theta) = k\theta, \tag{2.18}$$

By inserting (2.14) and (2.15) this yields the mapping from 3D point $\mathbf{M} = (x, y, z, 1)^T$ in camera coordinates to $\mathbf{m} = (x', y', 1)^T$ in image coordinates equivalent to the camera matrix K of the pinhole camera model:

$$\mathbf{m} = \begin{pmatrix} \tilde{r} \cos \arctan(\frac{y}{x}) \\ \tilde{r} \sin \arctan(\frac{y}{x}) \\ 1 \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \\ 1 \end{pmatrix}, \text{with: } \tilde{r} = k \arctan\left(\frac{\sqrt{x^2 + y^2}}{z} \right) \tag{2.19}$$

2.2.4 Back Projection

Having a 2D point $\mathbf{m}$ in image coordinates, the corresponding 3D point $\mathbf{M}$ has to be reconstructed by inverting the projection from $\mathbf{M}$ to $\mathbf{m}$. This process is described for the perspective projection only. For the equidistant projection instead of the camera matrix K, equation (2.19) has to be inverted.

Due to the fact that equation (2.13) reduces the dimension from three ($\mathbf{M}$) to two ($\mathbf{m}$), this projection cannot be inverted directly. Applying an inverse transformation

$$\mathbf{M} = \left[\begin{array}{c|c} \lambda \cdot (KR^T)^{-1} & \mathbf{C} \\ \mathbf{0}^T & 1 \end{array} \right] \cdot \begin{pmatrix} \mathbf{m} \\ 1 \end{pmatrix} \tag{2.20}$$

gives the direction from $\mathbf{C}$ through $\mathbf{m}$ to $\mathbf{M}$, but having only $\mathbf{m}$ and $\mathbf{C}$, the position of $\mathbf{M}$ depends on λ. All potential points $\mathbf{M}$ projecting onto $\mathbf{m}$ are located on the

ray from **C** through **m**. Having only one observation **m** of **M**, the depth λ cannot be reconstructed.

If there are more observations of **M** from cameras in different positions, the real position of **M** and λ can be reconstructed. This is the purpose of stereo vision. The crucial part of that is to find the corresponding observations $\mathbf{m}'$ without the knowledge of **M**. Stereo analysis is a research topic of its own [Koch, 1996, Roy and Cox, 1998, Yang and Pollefeys, 2003, Seitz and Dyer, 1997b] and it is not discussed here in detail. One approach, the plane-sweep algorithm, is described in section 4.2.6.

For most of the Image-Based Rendering methods presented here it is assumed that a stereo reconstruction can successfully be applied to the input image sequences. After the reconstruction of the λ for all, or at least nearly all, pixels of all images, they are stored in an image called *depth map* (see section 4.2.5). This is a view-dependent 2.5D representation of the scene and together with the images and the calibration of the cameras it is used as input data. Having the depth map, the back projection from equation (2.20) can be used.

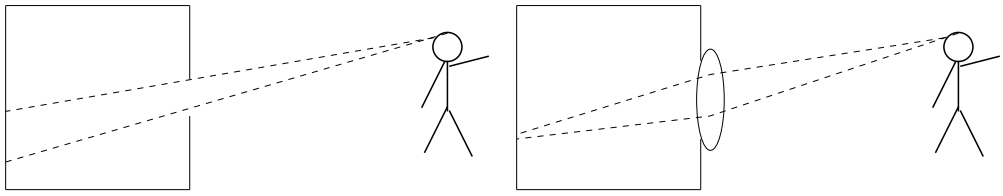
2.3 Limitations of the Model

The introduced camera models do not always suffice to explain the image formation of real cameras. Many different aberrations can be found, the most dominant ones will be discussed here. These apply to the perspective projection as well as to the equidistant projection

The pinhole camera model assumes the projection centre to be an infinitesimally small hole (aperture). In real cameras, the size of the aperture limits the amount of light passing through in a given time interval. Photosensitive material and sensor cells need a minimum dose of light to produce a detectable signal. With an extremely small aperture this dose can only be reached with a long exposure time (given standard lighting conditions).

Enlarging the pinhole to use more light rays results in blurred images. The reason for this is that light rays from the same 3D point on an object can intersect the image plane at different positions (figure 2.5(a)). Real cameras can only produce non-blurred images within reasonable exposure times when using lenses to control the path of the light rays as shown in figure 2.5(b).

Lenses introduce a variety of new artefacts which can be divided into geometrical and photometric distortions.



(a) Widened pinhole (b) Camera with a lens

Figure 2.5: Projecting through a wide hole as in (a) results in a blurred image because light rays from one scene point reach the image plane at different positions. A lens as in (b) can control the optical path.

2.3.1 Geometrical Distortions

The most dominant geometrical distortions are the radial- and tangential lens distortion. Radial distortion means that all image points are shifted along radii originating at the centre of the distortion depending on their distance to that centre. This leads to the well known pincushion- or barrel-distortion effect. According to Tsai [1987] the radial distortion can be modelled by using a polynomial as in equation (2.21). This assumes that the image coordinate system is located in the distortion centre and coordinates are normalised. Most often only the first two coefficients k_1 and k_2 are used.

$$\mathbf{m}' = \mathbf{m} + \mathbf{m} \sum_{i=1}^{n} k_i ||\mathbf{m}||^{2i} \tag{2.21}$$

Tangential distortion is an image point displacement along lines tangential to concentric circles around the distortion centre:

$$\mathbf{m}' = \mathbf{m} + \begin{pmatrix} -m_x \\ m_y \end{pmatrix} \sum_{i=1}^{n} k_i ||\mathbf{m}||^{2i} \tag{2.22}$$

The parameters of the radial and tangential distortion can be determined with standard calibration methods as described in [Tsai, 1987]. Applying an inverse transformation can undistort the images.

2.3.2 Photometric Distortion

Vignetting and image blur are the most common photometric distortions. Figure 2.6 illustrates the vignetting effect. With increasing distance to the centre, the intensity of the image falls. This can also be modelled by a polynomial modulation of the intensity:

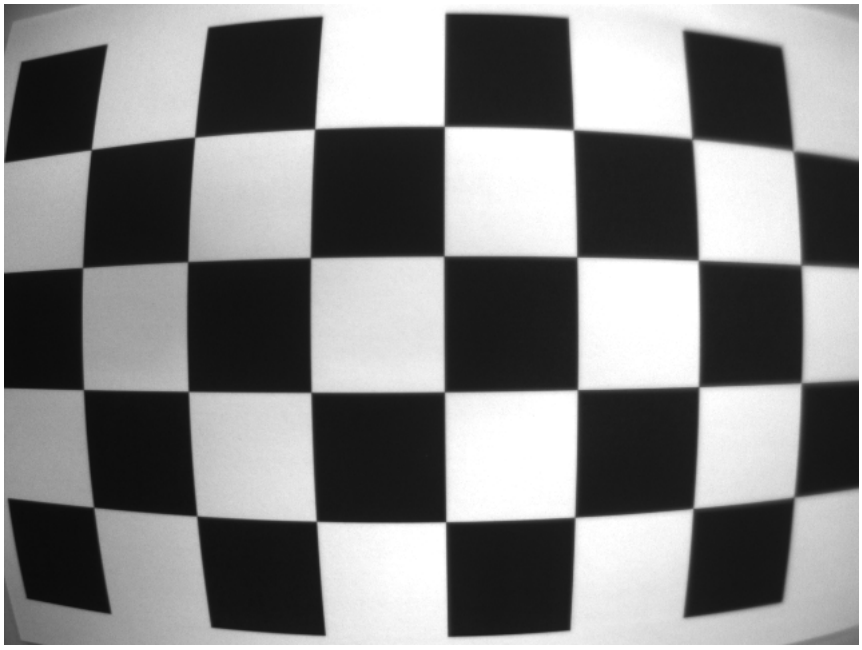


Figure 2.6: Radial distortion and vignetting of a 4mm micro-lens.

$$I'(\mathbf{m}) = I(\mathbf{m}) \cdot \sum_{i=1}^{n} k_i ||\mathbf{m}||^{2i} \tag{2.23}$$

By applying the inverse of the intensity fall-off function, vignetting can be removed from images.

Image blur caused by lenses is a low-pass filter process which cuts high frequencies. This information is lost and cannot be reconstructed. As discussed in section 2.1.1 the process can be modelled with the point spread function which is the impulse response for the lens. Each point projected by the lens is spread to a Gaussian distribution centred around the original point.

To capture the images projected by a camera, photo sensitive sensors are used. They consist of cells of a given area in a regular grid collecting light rays over time. After some integration the cells are read out and after a reset the next integration time starts. Each cell yields one pixel of the digital image. Although different techniques like *Charge Coupled Devices (CCD)* or *CMOS*-Sensors are used, some image artefacts are shared by all sensors: Noise and motion blur.

Motion blur occurs if parts of the scene or the camera is moved. This means that during the integration time movement of the scene or the camera becomes visible in such a way that a scene point **M** projects into different pixel position $\mathbf{m}_1$and $\mathbf{m}_2$. For non-stationary scenes the integration time must not

exceed a given threshold to avoid motion blur. This threshold depends on the velocity of objects.

Noise occurs, if the signal of the cells is very weak. Dark lighting conditions, small apertures and short integration times lower the signal. This has to be compensated by an increased gain level. The read-out amplifier has to boost the weak signal strongly. Due to thermal effects, the cells itself have a limited signal-to-noise ratio and amplifying the signal also amplifies the noise. Since noise is a statistical process, it cannot be measured and compensated in a single image. On the assumption that both camera and scene are stationary approaches to remove noise typically require image sequences over time.

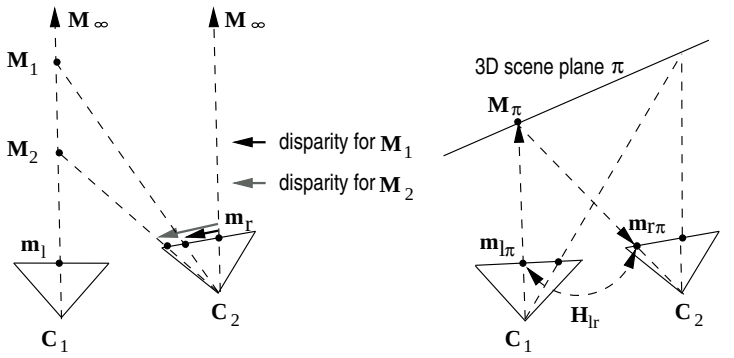
In the following it is assumed that image errors are either small and neglectable or otherwise have been corrected by the inverse transformations in a preprocessing step. All further processing assumes the pinhole camera model or a distinct spherical projection.

2.4 Disparity, Depth and Parallax

Image *parallax* or *disparity* describes the effects of scene depth in the image. This situation is explained in figure 2.7(a). A 3D point **M** projects onto a 2D image point **m** along the viewing ray $\overline{\mathbf{MC}}$ connecting the 3D point and the camera centre. For a perspective camera, the simple pinhole projection model as in equation (2.13) can be applied.

If a second camera at a different position observes the same 3D scene point, the projected image point in the second camera will lie on the epipolar line that is formed by the projection of the viewing ray $\overline{\mathbf{MC}}$ into the second image. The epipolar line itself is only determined by the relative pose between both cameras and the 3D point, while the position of the projected point on the epipolar line is determined by the scene depth relative to the first camera. The shift of the projected image point in the displaced camera is called disparity, see figure 2.7(a). If we assume a fixed setup of the two cameras, this disparity only depends on the projective depth λ as in equation (2.13).

The disparity defines whether direct visual interpolation from the plenoptic samples generates visual artefacts. Disparity is scene-dependent and needs compensation with depth-dependent warping.



(a) Parallax effects in the image

(b) Homography mapping for planar scenes

Figure 2.7: (a) Projection of an infinitely distant point $\mathbf{M}_\infty$ has disparity zero. Finite points generate image parallax with non-zero disparity. (b) Correspondences of planar scene points can be computed with a planar homography without any disparity error.

2.4.1 Infinitely Distant and Planar Scenes

In some instances, there may be no disparity between the real and synthetic views. In that case perfect reconstruction is possible without 3D scene knowledge. This is the case for infinitely distant or planar scenes where all geometry can be compensated with global homographies. For an infinitely distant point $\mathbf{M}_\infty = (x,y,z,0)$, the projection is independent of the camera displacement, because the homogeneous point coordinate is zero. Hence the point projection equation (2.13) simplifies to

$$\mathbf{m} = K(R^T | -R^T\mathbf{C})\mathbf{M}_\infty = K(R^T|0)\mathbf{M}_\infty. \tag{2.24}$$

Equation 2.24 applies to both cameras 1 and 2 in figure 2.7(a). As the 3D point $\mathbf{M}_\infty$ is identical for both projections, it can be eliminated and a correspondence transfer between the image points $\mathbf{m}_l$ and $\mathbf{m}_r$ is found with the homography H_{lr}:

$$\mathbf{m}_2 = K_2R_2^T\mathbf{M}_\infty = K_2R_2^TR_1K_1^{-1}\mathbf{m}_1 = H_{12}\mathbf{m}_1 \tag{2.25}$$

The homography H_{12} defines a planar projective mapping between both image planes. It maps each image point $\mathbf{m}_1$ onto the corresponding image point $\mathbf{m}_2$ via the relative rotation between the cameras. The correspondence transfer can also be seen as a mapping over the infinitely distant plane Π_∞, where the resulting disparity is zero. A similar relationship holds also for general 3D points in the following two cases:

1. The camera centres of both cameras coincide and both cameras define a single plenoptic sample, with rotated optical axes. This case applies to single-perspective panoramic imaging (see section 3.2) .

2. All scene points $\mathbf{M}_\pi$ lie on a real 3D-plane Π (see figure 2.7(b)). In that case, a homographic mapping between any 3D point on the plane $\mathbf{M}_\pi$ and the corresponding image points $(\mathbf{m}_{1\pi}, \mathbf{m}_{2\pi})$ exist:

$$\mathbf{m}_{2\pi} = H_{\pi 2}\mathbf{M}_\pi = H_{\pi 2}H_{1\pi}\mathbf{m}_{1\pi} = H_{12}\mathbf{m}_{1\pi} \tag{2.26}$$

2.4.2 Non-Planar Scenes at Finite Distance

In the general case the constraints for homography mapping are not fulfilled. Most scenes consist of objects in different depth and all objects within finite distance. If two cameras C_1 and C_2 with different centres of projection $\mathbf{C}_1 \neq \mathbf{C}_2$ observe such a scene, disparity can be observed in the images. To be more precise, projection of objects with different depth have different disparity.

Image synthesis has to compensate for this disparity. This can be shown with an example. If we assume two cameras as defined above, a virtual camera is placed halfway between both real cameras observing the same scene. The image for this camera can be computed by interpolation from the two real images. If both real images were blended without disparity compensation, the result would contain ghosting artefacts. The content of the images would not match together and each image would be visibile as a “ghost image”.

Disparity compensation can be achieved from implicit geometrical representations such as image flow and depth maps, or from explicit 3D shape representations. If applied properly, the content of both images coincide and the blended image is an interpolation of what the virtual camera would see.

2.5 Image Blending

When interpolating new views from two or more real images, colour and intensity values have to be blended properly. This is even more important if images of a virtual camera sweep are generated and assembled to a video sequence. Colours in images from real cameras and real scenes most often differ slightly over time and space and small inconsistencies can also occur. Blending has to ensure that these properties change smoothly in the virtual view.

Blending means that for each camera i the contribution of each intesity value I_i is weighted with a blending weight w_i and added to the final output value I:

$$I = \sum_{i \in N}^{i} w_i I_i, \text{ with } \sum_{i \in N}^{i} w_i = 1 \tag{2.27}$$

It is assumed that the intensity of the images has been equalised. For each new view generated, the blending weights have to be computed individually. When blending camera images, two different principles can be used to determine the weights w_i: Per-pixel weights or per-camera weights.

2.5.1 Per-Pixel Blending

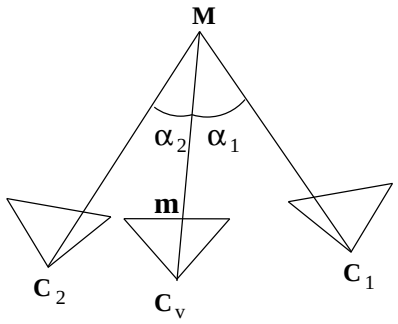


Figure 2.8: The contribution of C_1 for a particular pixel at **m** can be weighted according to the angle α.

One approach for a per-pixel blending method is described in [Verlani et al., 2006]. They propose a blending based on the angle between the line-of-sights from the virtual camera and each real camera as shown in figure 2.8 which can be implemented efficiently (as a fragment shader) by using the graphics hardware. The new image is generated by applying each real image as texture in a separate rendering pass. The alpha channel[2] is used to accumulate the blending weights used so far and ensure that the sum of all weights does not exceed 1. With a virtual camera located at C_v and a real camera located at C_i observing a point **M**, the angle α between the line-of-sights from the virtual camera and each real camera is:

$$\cos\alpha = \frac{(\mathbf{C}_v - \mathbf{M})^T (\mathbf{C}_i - \mathbf{M})}{|(\mathbf{C}_v - \mathbf{M})^T (\mathbf{C}_i - \mathbf{M})|} \tag{2.28}$$

[2]The alpha channel is an additional channel added to the existing red, gree anb blue channels of an image to encode additional information for each pixel. The resulting 4-channel representation is often referred to as *RGBA*

Based on α, the contribution for a pixel m is defined as a function $f(\alpha)$. For $f(\alpha)$ the authors of Verlani et al. [2006] suggest two functions:

$$f_1(\alpha) = \cos^k \alpha, \quad f_2(\alpha) = e^{-k\alpha} \text{ with } k \in \{2,3\} \tag{2.29}$$

2.5.2 Per-Camera Blending

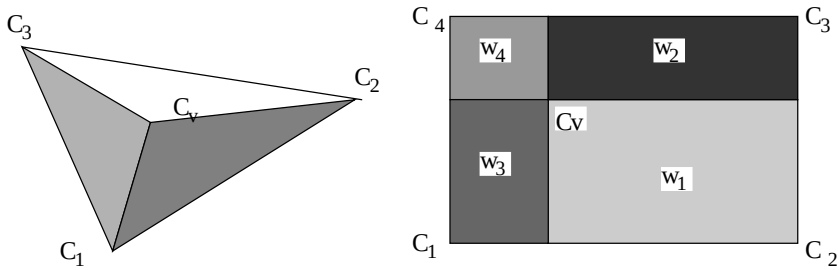
Instead of computing the weights independently for each pixel, weights can be computed for a given camera setup beforehand and then applied to all pixels of a real image. In contrast to a per-pixel blending weight computed in a pixel shader, all angles and distances are known and a global normalisation can be applied.

Barycentric Blending Weights

Starting with two cameras C_1 and C_2 and the virtual camera located in between them, linear interpolation gives the following blending weights:

$$w_1 = 1 - \frac{|\mathbf{C}_1 - \mathbf{C}_v|}{|\mathbf{C}_1 - \mathbf{C}_2|} \ , \ w_2 = 1 - \frac{|\mathbf{C}_2 - \mathbf{C}_v|}{|\mathbf{C}_1 - \mathbf{C}_2|} \tag{2.30}$$

If $\mathbf{C}_v$ does not lie on the line $\overline{\mathbf{C}_1\mathbf{C}_2}$, its projection onto this line has to be used in equation (2.30) instead of $\mathbf{C}_v$ itself.

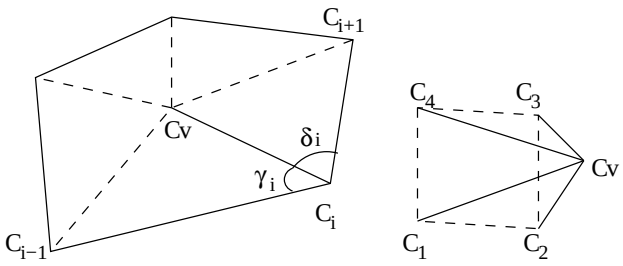


(a) 3 Cameras (b) 4 Cameras

Figure 2.9: Barycentric weights for $\mathbf{C}_v$ inside a triangle of three real cameras (a) and bi-linear interpolation between four cameras (b)(equivalent to barycentric weighting).

Having not only one-dimensionally distributed cameras, but N real cameras, any three centres of them span a 2D plane. It is assumed that the centre of the virtual camera is also located on that particular plane. If $\mathbf{C}_v$ lies in the range covered by real cameras, it is enclosed by a triangle formed by three real cameras. These three cameras can be selected for interpolation and their blending weights can be computed as the barycentric coordinates of the point $\mathbf{C}_v$ inside the triangle

$\mathbf{C}_1\mathbf{C}_2\mathbf{C}_3$. As shown in figure 2.9(a), these weights w_i correspond to the areas opposite to $\mathbf{C}_i$[3]. The white area is the weight for $\mathbf{C}_1$, the light grey area is the weight for $\mathbf{C}_2$ and the dark grey area is the weight for $\mathbf{C}_3$. If $\mathbf{C}_v$ is directly placed on the connecting line segment of two real cameras, the barycentric weight for the third camera becomes zero leading to linear interpolation between the remainging views as shown above.



(a) 5 cameras (b) Extrapolation

Figure 2.10: (a)Generalised barycentric coordinates for convex polygons and (b) extrapolation which lead to negative barycentric coordinates and cannot be solved with eq. (2.31).

Barycentric coordinates are not limited to triangles. Considering 4 cameras around $\mathbf{C}_v$ forming a quadrilateral, barycentric weighting directly leads to bi-linear interpolation as show in figure 2.9(b). A generalised extension to convex polygons is given by Meyer [2002]. In this case, as illustrated in figure 2.10(a), the barycentric weight w_i for each real camera corresponding to a vertex of the convex polygon can be computed from the adjacent vertices $\mathbf{C}_{i-1}$, $\mathbf{C}_{i+1}$ and $\mathbf{C}_v$ only:

$$w_i = \frac{\cot\gamma_i + \cot\delta_i}{||\mathbf{C}_v - \mathbf{C}_i||^2} \tag{2.31}$$

This is only defined if convexity is ensured and if $\mathbf{C}_v$ does not lie on an edge of the polygon. If $\mathbf{C}_v$ lies on a line between two vertices, one of the angles γ_i and δ_i becomes 0, which results in an undefined cotangent. Before computing the weights, the polygon has to be checked for this undefined case. If $\mathbf{C}_v$ is located on an edge, interpolation can be reduced to two cameras with linear blending weights. The convexity constraint can be fulfilled by proper selection of real cameras to use (see figure 2.11). A camera selection scheme ensuring convexity is presented in section 5.1.3.

[3]This is not only valid for this example, it is a property of barycentric coordinates.

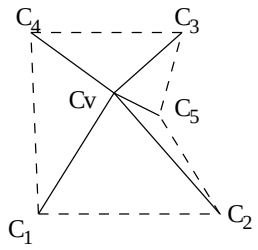


Figure 2.11: A non-convex camera setup. Eliminating one camera out of C_2, C_3, C_5 would result in a convex setup.

Extrapolation

In contrast to view interpolation, where the virtual camera is surrounded by real cameras, extrapolation is performed if the virtual camera is moved outside of the region covered by real cameras. In this case, barycentric weighting cannot be used for blending, because the virtual camera will be outside of any polygon spanned by the real cameras. This would yield negative barycentric coordinates for some cameras and those negative weights would corrupt the composed image. Instead, distance-based linear blending can be used for N active cameras:

$$w_i = 1 - \frac{|C_i - C_v|}{\sum_{j=0}^{j<N} |C_j - C_v|} \tag{2.32}$$

Hole filling

In some cases unfilled regions in the reconstructed image can remain even if surrounding cameras and the blending weights are selected properly. This problem occurs, for example, if the virtual camera is closer to the scene than the real cameras. Regions occluded in the nearest real cameras can become visible in the new view.

To fill these regions not seen by any of the real cameras used for blending, data from real cameras which have seen these regions have to be used. Typically, the real views closest to the virtual camera are selected for blending, so additional information can only be contributed by real cameras with increased distance. For each pixel to fill, the useful real views have to be determined and blending weights have to be computed.

This implies that image generation cannot be reduced to the nearest N real views. In the worst case, all real views have to be taken into account and many different combinations of real views have to be blended. To cope with this increased complexity in an implementation, per-pixel blending can be preferred to per-camera blending. Per-pixel blending has the advantage, that the number of

contributing views does not have to be known in advance (before computing the pixel value).

2.6 Programmable Graphics Hardware

The idea of specialised computer graphics hardware is that the demanding process of computing colour values for every pixel on the screen can be accelerated and computed concurrently by using optimised hardware. Driven by the development of computer games, the performance of graphics processing units (GPU) grew faster than that of the CPUs. Today's GPUs are not just configurable, but also programmable which allows to change the process of how pixels are computed. To make programmable graphics hardware understandable first the standard graphics pipeline as defined by OpenGL is presented.

Hardware-accelerated rendering only knows a few primitive items (*primitives*) to be drawn: Points, lines and polygons. They are all specified by vertices. A vertex is a 3D homogeneous point. These vertices are transformed by matrix multiplications to project them into a virtual camera. Following the pinhole camera model, intrinsic and extrinsic parameters are given in a *projection matrix* and a so-called *model-view matrix*, respectively. The model-view matrix transforms object-centered coordinates into camera-centred coordinates and the projection matrix transforms from camera- to normalised device coordinates followed by the perspective division. All these transformations together are called the *vertex pipeline*.

Using these vertices in normalised device coordinates, the rasteriser processes the primitives and generates *fragments* filling the space between the vertices according to the primitive type and the required output resolution. A fragment can be seen as a potential pixel. For every fragment, the fragment processing has to determine the colour. This is done by using static colour information, material properties, light source and textures. A fragment passes several tests, for example z-test to ensure correct occlusion, a user-defined alpha-test, a scissor- and a stencil test. A viewport-transformation into screen-space coordinates then addresses pixels on the screen. The corresponding pixel is coloured from the fragment that passes all tests. All these transformations together are called the *fragment pipeline*.

Figure 2.12 illustrates the graphics pipeline of OpenGL.

So far, this process can only be configured by changing parameters. Programmability is introduced at two positions. The vertex pipeline is replaced by a programmable vertex processing unit and the fragment- or pixel-pipeline is replaced by a programmable fragment processing unit. The rasteriser and the fragment tests are kept identical. By not replacing the whole pipeline with a generic processor, a high degree of parallelism can be achieved. The vertex computa-

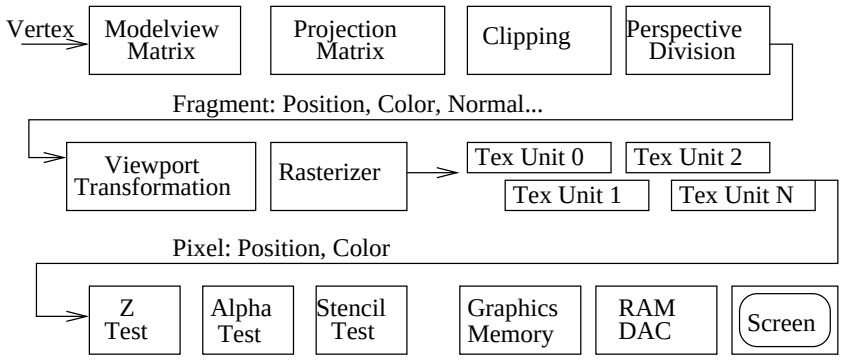


Figure 2.12: A simplified schematic diagram of the standard OpenGL pipeline sometimes referred to as *Fixed Function Pipeline*

tion is independent of the fragment computation and each of both can be done by multiple processing units.

The user can upload programs to the processing units called *vertex program* or *vertex shader* and *fragment program*/*fragment shader*. A vertex program is run whenever a vertex is inserted into the GPU. The input of the program can be the vertex position, colours, matrices and more. The result of a vertex program is always one vertex. Vertices cannot be created or discarded in a vertex program[4]. A fragment shader can access 3D positions, textures and texture coordinates, colour and more. The result can be a colour but the fragment can also be discarded. There are different programming languages for shader programs. For this work a C-like language called *Cg* [Fernando and Kilgard, 2003] has been used.

To access textures and look-up colours in a texture, GPUs have a limited number of *n texture units*. Each texture unit can bind a texture so that a fragment program can access *n* textures during operations. The binding of textures to texture units cannot be changed while the fragment shader is working.

GPUs need memory for their computations. This memory is located on the graphics card to allow high-bandwidth memory access. In addition, main memory can be mapped into the address range of the GPU but access to main memory is slower than to local memory. The local memory can be used to store textures or lists of vertices and more. So-called *Framebuffer Objects (FBO)* allow to manage and access memory. A common technique is to setup an FBO as destination for the GPU so that the result of the rendering is not displayed on the screen but kept in local memory. In a second render pass, this memory is used as a texture for further computations. Several of the Image-Based Rendering methods in chapter 5 make extensive use of the programmable graphics hardware.

The complexity and performance of GPUs has increased rapidly over the last

[4]The *geometry shader* of the latest GPU architectures can create or discard vertices,

few years, hence only a snapshot of the development can be given here. Current GPUs (late 2006) have up to 128 processing units working concurrently, which can be used for vertex or fragment processing. Up to $1.1 \cdot 10^9$ vertices and $36 \cdot 10^9$ pixels per second can be processed (Nvidia GeForce 8800GTX). This can be doubled or multiplied by four if two or four GPUs are coupled to work together. An overview of the GPUs used for this thesis can be found in appendix A.2.

Chapter 3

Related Work

The field of Image-Based Rendering has existed for roughly a decade now. In this chapter an overview of system and methods for view synthesis which have been developed during that period will be given. As mentioned in the introduction, this chapter has previously been published as a contribution to "3D View Synthesis" in [Koch and Evers-Senne, 2005] which appeared in July 2005. At nearly the same time, Shum, Chang and Kang published their book "Image-Based Rendering" [Shum et al., 2005], which is a comprehensive work covering all aspects of that research area. They also give a very detailed review of existing techniques.

3.1 Categorisation of Image-Based View Synthesis Methods

Image-based view synthesis is essentially a re-sampling of the plenoptic function (of light rays). Synthesising a virtual view is equivalent to generating the set of light rays that form the pixels of the new image. In the last ten years several different approaches have been developed.

3.1.1 Previous Categorisation Approaches

A standard classification scheme has not yet evolved and several survey papers all use different categorisations.

Kang [1997] presented one of the first approaches for structuring IBR methods. He proposed 4 categories: non-physical[1] based image mapping, mosaicing, interpolation from dense samples, and geometrically-valid pixel re-projection.

[1]The projection properties of the generated images are not related to existing physical imaging devices.

McMillan and Gortler [1999] distinguished IBR systems based on their intermediate data representation: approximate scene geometry, images in databases to represent different environment locations, and images as reference scene models from which new views are interpolated.

Shum and Kang [2000] classified IBR systems into three categories depending on how much geometric information they use: Systems not using any geometric information, systems using implicit geometry, and systems using explicit geometry. The term *implicit geometry* is attributed to all systems that extract the geometrical information directly from the given images, while the term *explicit geometry* is used to describe systems that utilise additional geometric sources or externally given 3D surface data. IBR systems utilising implicit geometry differ with respect to its use for depth compensation.

3.1.2 Taxonomy of IBR Systems

The following discussion follows the categorisation of Shum and Kang, however, geometry alone does not suffice for categorisation. An important additional factor is the spatial placement of the *PS* and the poses of the virtual views. Given an arbitrarily structured scene, the following three topics define the amount of geometry which has to be used for view synthesis.

1. **Sampling Density**. If samples are distributed densely over space, in such a way that for each synthesised viewing ray there is a real plenoptic sample close by, re-sampling is simplified to ray selection and colour interpolation between the nearby rays. The depth parallax will not distort the novel view and approximately geometry-free reconstructions are possible. Prominent examples of such reconstructions are panoramic viewing and light field approaches. This simplification does not hold for sparse sampling, where novel views are generated with possibly large parallax. For virtual views that are far from real views, parallax-dependent compensation is necessary and geometric warping must be employed.

2. **Distribution of sample positions**. The spatial arrangement of samples allows to design special configurations where parallax-free rendering is possible in a restricted spatial range. It is clear that direct rendering can be used if novel views coincide with real sample positions. This is the case for spherical panoramas where free look-around capabilities exist but no viewpoint change is possible. The samples can be distributed arbitrarily. Recently, a large number of systems have been designed that exploit spacial dense and regular 1D or 2D arrangements of the samples. A typical 1D arrangement is the concentric mosaic that places dense sampling of real viewpoints on a

circular path by rotating a camera on a fixed rod. Novel views can be generated if the rendered position is in the plane of the camera's movement and the viewing direction is restricted to lie inside the circle.

The light field approach is a 2D arrangement where samples are placed in a 2D planar or spherical patch around the scene. Novel views can be generated from viewpoints inside a restricted volume.

3. **Possible poses of virtual views**. The goal of IBR is rendering from freely-chosen novel viewpoints. However, some systems impose restrictions on the possibles poses. In a panoramic viewer, full rotational interaction is possible but one may only hop between discrete viewpoints. For a light field, free viewpoint selection is possible within a restricted viewing volume and viewing direction, and for 3D reconstructions, the virtual camera may move continuously with full 6 degrees of freedom.

The taxonomy used here groups the different IBR approaches w.r.t. the above mentioned categories. Three dimensions of influence have been identified that are drawn in the category overview figure 3.1:

1. **Geometry axis**: The use of geometry needed to compensate for scene parallax, ranging from systems without parallax compensation, to coarse geometrical approximations, local image correspondences and on to full 3D information. This axis is mostly correlated with the complexity of the system, because the precise extraction of geometrical information from images is difficult.

2. **Image axis**: The spatial distribution of the *PS*, ranging from few to many unstructured samples in the lower half, and a structured dense sampling in the top half of the graph. This axis is correlated with the memory demand of the system and the complexity of image acquisition, because very many images may be needed in a possibly highly structured way.

3. **Virtual viewpoint selection (colour-coded)**: The ultimate goal of each IBR system is full freedom of the virtual camera pose, but some systems like panoramas restrict the camera motion to discrete positions, others allow only restricted motion. I have coarsely categorised the methods into three motion categories: predetermined discrete positions (black), constrained motion along predetermined path (dark grey) and unrestricted motion (light grey).

In the following sections, known algorithms are presentd and discussed in relation to this taxonomy.

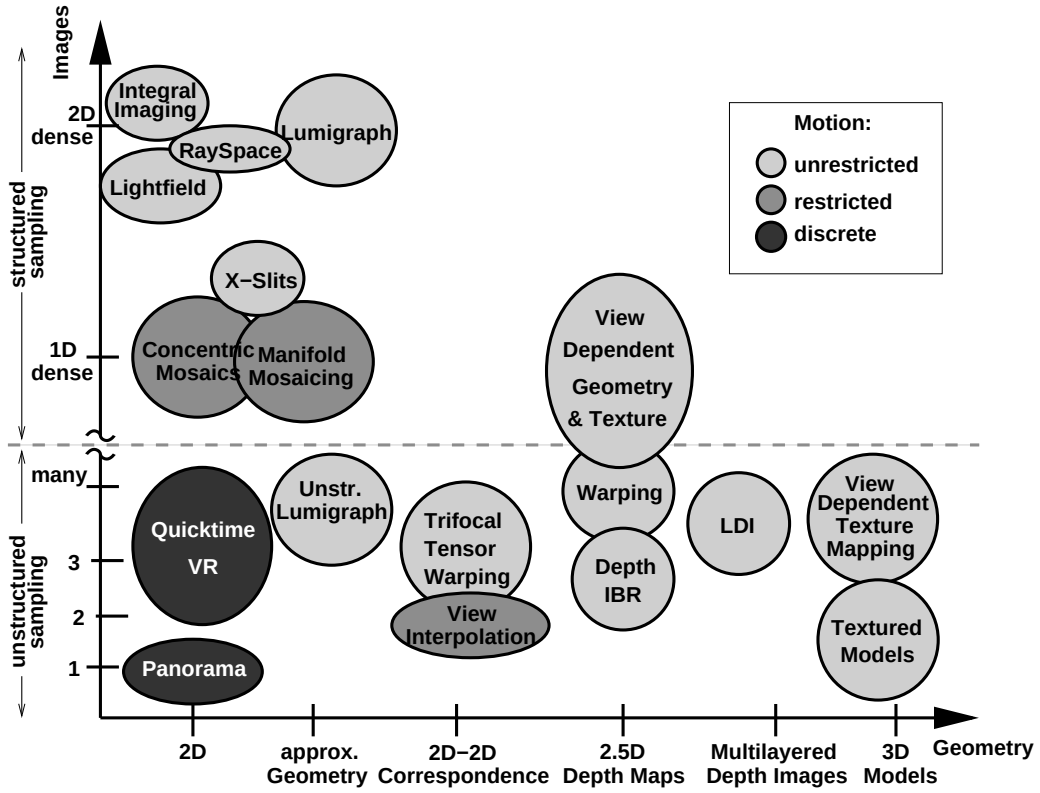


Figure 3.1: Categorisation of different IBR methods. Methods are arranged according to the amount of geometry used (geomatry axis) and the number of images required (image axis). The image axis is split into two sections: Unstructured sampling and structured sampling. The colours encode the possible motion of the virtual camera.

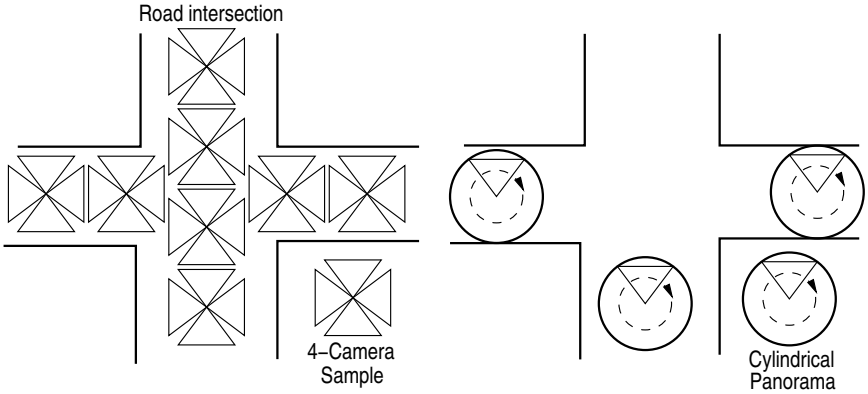
3.2 Rendering without Geometry

IBR Methods within this category use no geometry information at all. Since no parallax compensation is possible, the sampling must be either very dense, or the possible motion is restricted to lie near to the sample positions. Basically, these methods use a high sampling density and avoid geometric complexity.

3.2.1 The Aspen Movie-Map

The earliest system to obtain restricted interactive look-around capabilities was the Aspen Movie-Map by Lippman [1980]. A car was driven along the streets of Aspen, Colorado, recording simultaneously 4 camera streams looking at right angles to cover a cylindrical view of the scene for every 3 m (see figure 3.2(a)).

The streams were captured on video disk for interactive play-back. This allowed to render novel views from the given camera path with interactive look-around capabilities. As the video stream could be edited at street intersections, one could interactively select the route by switching between segments of the video stream. However, no deviation from the given camera path and positions was possible.



(a) Aspen Movie-Map (b) QuicktimeVR

Figure 3.2: (a) Aspen Movie-Map is based on dense sampling with 4 cameras. For each sample point, 4 orthogonal views were stored and could be selected interactively. (b) QuicktimeVR uses less dense sampling but obtains high-resolution cylindrical or spherical panoramas.

3.2.2 Quicktime VR

A similar approach is used in Apple's Quicktime VR (QVR) system [Chen, 1995], where panoramic images are taken at discrete viewpoint positions(see figure 3.2(b)). A camera is rotated on a tripod at the fixed position and all images are stitched together to form a plenoptic sample with a cylindrical or spherical panoramic view of the scene [Shum and Szeliski, 1997]. By selecting the camera positions during capturing properly, the user can interactively explore the scene by switching between different panoramas. The virtual viewpoints are restricted to lie exactly on the sampled real views, but look-around capability is provided by the cylindrical or spherical representation.

The Aspen Movie map system and QVR differ with respect to the spatial sampling. Both systems collect plenoptic samples of the scene, but the samples in the Aspen system are arranged in a systematic 2D linear grid, while QVR samples are located arbitrarily.

3.2.3 Central Perspective Panoramas

The panoramas discussed here use the single centre perspective assumption. Therefore, there is no parallax between the images, which allows high fidelity in the reconstruction. More recent implementations of such a system can also be found in [Antone and Teller, 2002, Teller et al., 2003], where additional spatial information and view interpolation is described. Kawasaki et al. [2001] use sequences of panoramic images (same as in [Takahashi et al., 2000]) to render cities with this approach.

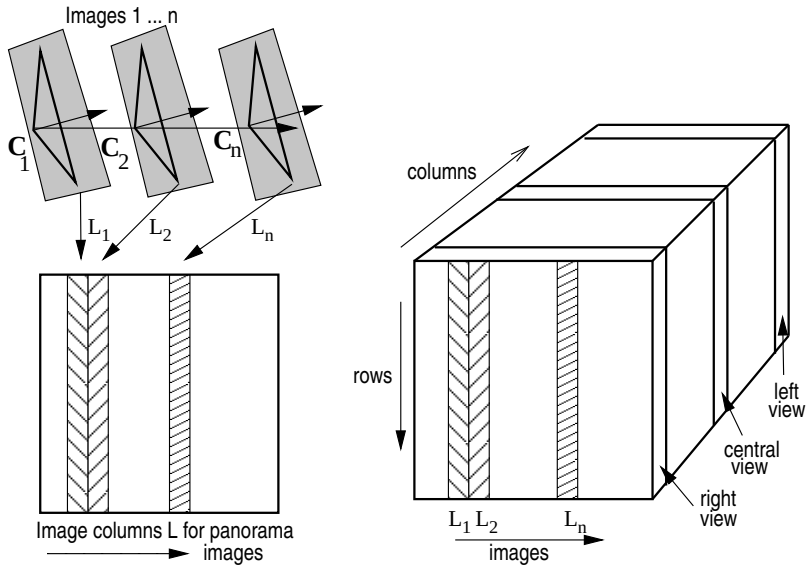
Despite their simplicity, panoramas are a very attractive means of IBR and versatile in use. Central perspective panoramas are easy to capture and mosaic stitching software nowadays comes with many digital cameras. They deliver best image quality for static scenes, and novel multi-camera hardware developments even allow panoramic video streams with high fidelity [Kang et al., 2003]. Panoramas can also be used to capture environment maps of the incident light with high dynamic range images for integration of virtual objects in real scenes [Debevec, 1998].

For video communications, panoramic images can serve to create and visualise static background, but due to the missing parallax they cannot be used to generate novel views of near-by objects.

3.2.4 Manifold Mosaicing

More recently, a number of approaches have been introduced to overcome the limitation of fixed viewpoints, and alternative sampling and reconstruction approaches have been investigated. They are all based on the assumption that novel views may be generated from a dense sampling of the scene. As a full 3D sampling of the *PF* is not feasible, subspaces are sampled densely to allow bounded continuous view reconstruction.

A panorama has a fixed viewpoint and is parametrised by the two direction angles of the sphere. Peleg and Herman [1997] developed a new type of non-central panoramas called *manifold mosaics*. The camera is not fixed but moves on some predefined trajectory while recording dense sequences. The camera motion is typically 1-dimensional, following a linear or circular track. Unlike conventional central perspective mosaic stitching the resulting manifold panorama is composed of small image stripes from all the different views into one multi-perspective manifold image. Figure 3.3(a) shows the capture and strip selection for a manifold mosaic. The work was inspired by the 1-d push-broom line scanner cameras that are used in aerial imaging of a flat terrain. All data can be stored together in a spatio-temporal image volume (figure 3.3(b)). The manifold panorama can therefore be described by an MCOP (Multiple Centre Of Projection) image



(a) Manifold mosaicing (b) Epipolar Plane Image Volume (EPI)

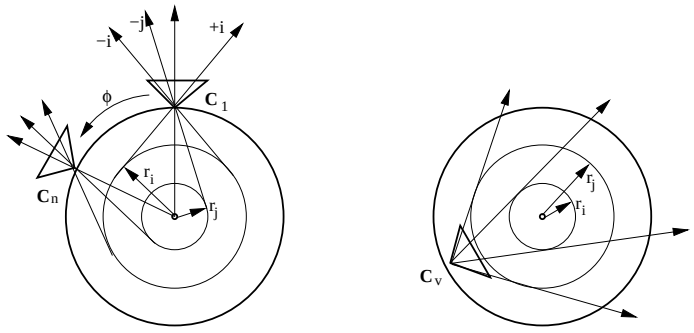
Figure 3.3: (a) Top: Manifold recording along a linear path by selecting fixed columns in each image. Bottom: Depending on the selected column, a sheared left or right view for stereo viewing can be generated. (b) Images are arranged in the epipolar plane image volume (EPI). Slicing the EPI along a fixed column over all images will generate a manifold mosaic.

[Rademacher and Bishop, 1998]) where each image stripe (image lines perpendicular to the image motion) has another projection centre and is taken from another image. There is a strong relation to the EPI (epipolar plane image analysis by Bolles and Baker [1986]) that slice the space-time image volume to analyse 3D scene parallax.

Typically, the camera motion will be linear in horizontal direction only, and from each image a central 1-d slit column will be used to paste into the panorama. The resulting image has central perspective in the vertical column direction but parallel perspective in the horizontal direction of camera motion. Therefore, a parallax compensation is needed for the vertical perspective image direction. It is easy to generate stereoscopic panoramas from this configuration by selecting sheared columns that control the gaze direction. See also Rousso et al. [1997, 1998] for detailed description.

3.2.5 Concentric Mosaics

The concentric mosaic as proposed by Shum and He [1999] is a special configuration of a manifold mosaic. A camera is mounted on a horizontal arm and rotated on a circular path around an axis of revolution looking outwards, see figure 3.4(a). A dense image sequence is recorded along the path. Similar to the linear manifold mosaic, vertical image stripes are cut from the image sequence to form MCOP images that are parametrised by the rotation angle and the elevation angle of the column height. Depending on the gaze angle of the column, the panorama for each slit i can be viewed as a bundle of rays that all lie tangential to a concentric circle of radius r_i. By selecting the image column, different radii are used, each forming a concentric mosaic. The central column generates a circle with radius 0 (a conventional cylindrical panorama).



(a) Concentric Mosaic (b) Synthesis from Concentric Mosaic

Figure 3.4: (a) A camera is rotated with angle ϕ on a circular path. Image columns $+i$ and $-i$ coincide with rays tangential to a circle with radius r_i. (b) Image columns for a virtual view are taken from the tangential rays of the concentric mosaic.

Essentially, a concentric mosaic records a dense circular disk of view positions of the *PF*. The resulting plenoptic function is parametrised in three dimensions by the rotation and elevation angles as well as the radius r that describes the circle selected by the column.

$$I_{(\lambda)} = \mathrm{Plen}_{\mathrm{Concentric}}(\phi, \theta, r) \tag{3.1}$$

Virtual views can be rendered from the concentric mosaics if the viewing position is constrained to lie within the bound of the concentric circles. To compose a new view, the horizontal viewing direction of each image column is computed and the tangent ray for each light ray is computed from interpolation between the recorded concentric mosaics (see figure 3.4(b)). The interpolation is possible only

for horizontal rays that lie within the concentric plane. The vertical elevation angle is not interpolated but the stored image columns are stitched together directly. The resulting images are not perspective since parallax on the elevation angle is not compensated, but image deformations are minor as long as the elevation opening angle of the camera is small [Shum et al., 2002]. Again, stereoscopic views can be created by rendering two displaced views from the same concentric mosaic. These stereo images can be used for direct stereoscopic viewing, or scene depth can be extracted with conventional binocular stereo analysis.

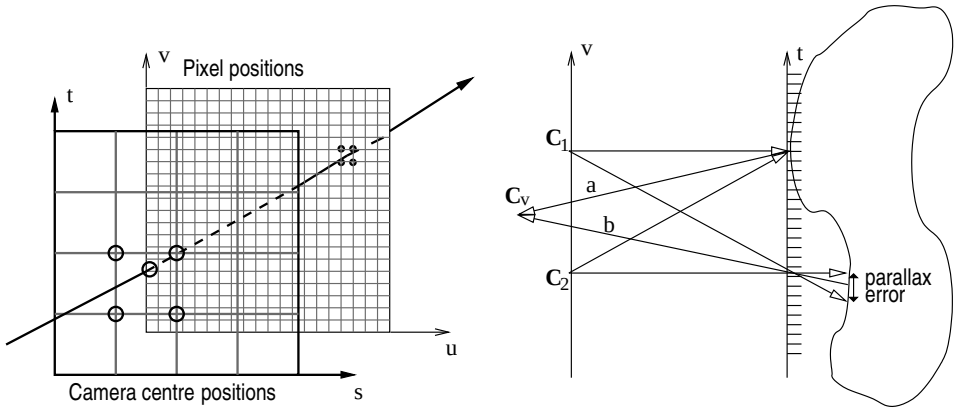
3.2.6 Cross-Slit Panoramas

The cross-slit projection (X-slits) proposed by Weinshall et al. [2002] and Zomet et al. [2003] is a generalisation of most of the above mentioned techniques. A novel camera model is proposed that uses two displaced 3D lines (slits) to describe the projection process. In that general model, the possible projection rays are formed by all possible connections between both slits. If both slits intersect in a single point then the X-slit model degenerates to the central perspective model with a single focal point and a spherical ray representation. If both slits do not intersect, some constraints on the projection will hold. One interesting configuration is to use a horizontal and a vertical slit. Fixing a single point on the horizontal slit will generate a vertical 1-d slice of the image (push-broom image). It can be used to render novel views from manifold panoramas where the horizontal slit defines the positions of the input images and the vertical slit defines the image slices. This is equivalent to cut slices in the spatio-temporal image volume of the input sequence. Bakstein and Pajdla [2003b] use this model to construct novel views with high fidelity. Concentric mosaics can also be described with this model. Combining a ray-space representation, omnidirectional plenoptic sampling and x-slit projection Bakstein and Pajdla [2003c] generate new omnidirectional views and stereo mosaics from high-resolution concentric mosaics.

3.2.7 Light Field Rendering

Levoy and Hanrahan [1996] introduced the *Light Field* which allows free 3D motion of the virtual view in a bounded volume. The light field interpolates new views using a 4D representation of the plenoptic function. It is based on the assumption that light rays are emitted from the surface of an object that is enclosed by a rectangular bounding box. The system records all rays that leave one side of the bounding box by placing a very dense regular 2D-grid of cameras looking into the bounding box. Two coplanar planes (u,v) and (s,t) are placed in such a way that the optical centres of the real cameras are located in the (s,t) camera plane (the sample positions x,y) while the image planes are all rectified to the (u,v) pixel

plane (the viewing directions ϕ, θ). This setup is shown in figure 3.5(a). To capture the appearance of an object from all sides, six (u,v,s,t)-configurations have to be placed around it.



(a) Two-plane parametrisation of the light field

(b) Novel view rendering using ray interpolation (v,t-slice)

Figure 3.5: (a) 4D Lightfield parametrisation with two planes. Each light ray is defined by the intersection with (u,v)-(s,t)-coordinates. (b) Ray interpolation for novel view synthesis. Ray **a** is interpolated correctly, while ray **b** has parallax artefacts due to displaced scene geometry.

Because of the specialised capturing geometry, the Plenoptic Function can be re-parametrised from the 5D function (2.4) to a 4D representation (figure 3.5(a)). Each light ray passing through the volume between the planes can be described by its intersection points with the planes. Thus the Plenoptic Function can be written as:

$$I_{(\lambda)} = \mathrm{Plen}_{\mathrm{Lightfield}}(u,v,s,t) \tag{3.2}$$

During recording, for all pixels of all real views, light rays are computed and parametrised as in equation (3.2). All these light rays are stored in a ray database called the Light Field.

A new view can be generated if it is placed outside of the bounding box looking inside or vice versa. For each pixel i of the new view, a viewing ray $\mathbf{r}_i$ is computed that passes through the 2-plane parametrisation and generates a particular sample (u_i, v_i, s_i, t_i). If such a sample exists in the database, the appropriate colour value is assigned to the pixel i. If no matching ray can be found, the nearest ones are selected and blended. In general, a desired ray may not pass through the optical centre of one real camera, but it passes between 4 centres of real cameras. For that reason there are four potential (u,v)-coordinates. When the desired ray passes through the (s,t) plane in general it will not hit one centre of a pixel but

will pass between four pixels centres giving four (s,t)-coordinates. For these four corresponding pairs of (u,v)-(s,t)-coordinates, rays can be looked up in the ray database and quadrilinear interpolation is performed.

The chosen parametrisation assumes a flat scene lying in the (s,t) plane. The visual reconstruction can achieve high accuracy, if a given scene matches this criterion or if the sampling is very dense so that the distance between adjacent camera centres is small. Less dense sampling of the viewing space and non-flat structures result in visual artefacts due to interpolation between views, as no parallax compensation is used. In figure 3.5(b) the effect of scene parallax in the lightfield is shown. A virtual camera is placed at position $\mathbf{C_v}$ and novel rays **a** and **b** are interpolated from the nearby cameras at positions $\mathbf{C_1}$ and $\mathbf{C_2}$. For ray **a**, a valid interpolation is found since the scene surface coincides with the pixel plane. For ray **b**, however, parallax effects will occur as the wrong viewing rays are interpolated.

3.2.8 Lumigraph

Simultaneously to the light field, Gortler et al. [1996] developed their *Lumigraph* system. Very similar to the light field, a ray database parametrised by two planes is built and view synthesis selects rays from it. To handle image acquisition from a hand-held camera, images from cameras which are not coplanar to the (s,t)-plane are rectified and mapped onto the light field plane. This rectifying interpolation uses a convex 3D shape approximation of the scene, the visual hull obtained from silhouette intersection. Thus the categorisation of the Lumigraph is chosen to be with some approximate geometry for preprocessing. The rendering itself does not use any geometry for scene warping.

3.2.9 Ray Space

Another 4D re-parametrisation of the Plenoptic Function is the *Ray Space*. First published by Fujii [1994] it uses a plane in space to define bundles of rays passing through this plane. For the (x,y)-plane at $z = 0$ each ray can be described by its intersection with the plane at (x,y) and two angles (θ,ϕ) giving the direction:

$$I_{(\lambda)} = \mathrm{Plen}_{\mathrm{RaySpace}}(x,y,\theta,\phi) \tag{3.3}$$

After capturing a scene with several cameras the re-parametrised data can be stored in a 4D structure. New views can then be synthesised by looking up intensities from this ray database. This method is a hybrid between 4D-Light field and EPI. Fujii and Tanimoto [2002] exploit this approach in their Free-Viewpoint TV (FTV) system.

3.2.10 Related Techniques

Several extensions to the light field and Lumigraph methods have been presented in the last few years. Buehler et al. [2001] proposed *Unstructured Lumigraph* rendering, which is a hybrid design between view dependent texture mapping (VDTM) and Light Field rendering. Unlike VDTM, they do not rely on a high-quality geometric model, but they need a geometrical approximation of the scene.

Tong et al. [2002] enhanced the Lumigraph by layers of different resolution for effective Level-of-Detail control based on the optimal sampling for a given scene complexity. Via epipolar plane image analysis [Bolles and Baker, 1986] surface planes are identified in the scene and a geometry approximation is generated for VDTM texturing.

Kurashima et al. [2002] set up a video conferencing system using a geometry approximation and View Dependent Texture Mapping to generate perspectively corrected views. From up to four cameras observing the user, a plane+parallax approximation is calculated.

Isaksen et al. [2000] propose a framework for dynamic re-parametrisation of 4D Lightfields. It allows arbitrary scenes and camera setups and opens up several other possible effects like aperture or depth-of-field.

Takahashi et al. [2000] use image sequences from hemispherical cameras taken from a driving car and analyse where the virtual camera can be placed. New views are then synthesised by collecting slits (rays) from different images.

Naemura et al. [2001] describe an approach to synthesise arbitrary new views from Integral Photography images for auto-stereoscopic displays. Integral Photography (IP) is a technique for dense spatial sampling of the plenoptic function by using micro-lens arrays in front of one single standard CCD sensor. Each micro-lens has its own centre of projection which makes this setup similar to a grid of small classic cameras. Due to the limited resolution of the sensor which is shared by all lenses, the angular resolution is small compared with a standard camera. Auto-stereoscopic displays use the inverse method by displaying multiple-perspective images on screens with lens or prism-arrays, see chapter 14 for more details.

An efficient Light Field representation for known or estimated geometry has been proposed by Chen et al. [2002]. By factorisation of 4-dimensional light field data into surface maps and view maps, compression can and rendering is performed with the use of programmable graphics hardware. The restriction of a flat scene is circumvented.

3.3 Rendering with Geometry Compensation

So far, only methods have been described which do not handle scene parallax explicitly because it is assumed that a dense sampling or planar scene will avoid parallax effects. The trade-off is that one is restricted in the interactive camera pose selection for the virtual view. If full 3-dimensional interaction is desired, then parallax effects must be compensated during rendering. It can be distinguished between methods that *interpolate* between given views along a predefined path, and the *extrapolation* of virtual views by freely selecting the pose.

3.3.1 Disparity-Based Interpolation

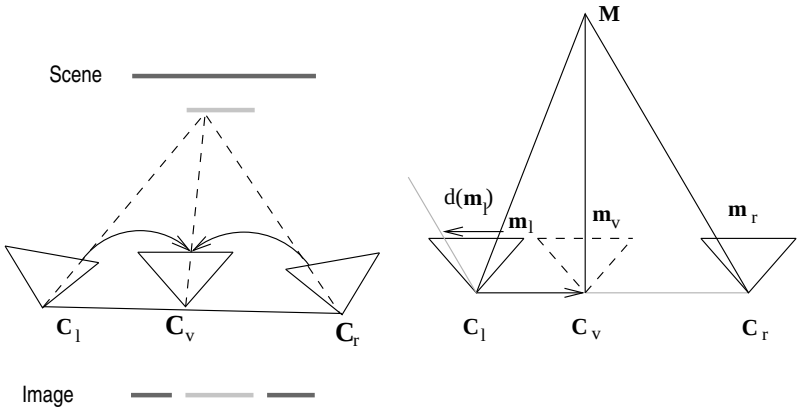
Disparity-based interpolation methods, or 2D-morphing, interpolate novel perspective views from a given pair of real views and a given dense correspondence map between the views. The relative pose of both real views is known, hence one can compute the epipolar geometry between the views which encodes the direction of the correspondence vector for each image point. The disparity encodes the length of the epipolar displacement for each image correspondence. Chen and Williams [1993] introduced view interpolation by linear interpolation of the length of the 2D-correspondence vectors (the disparity). With this method, the virtual view is placed on the line between the real camera centres. Correct perspective views will be generated only if the real cameras are in rectified standard stereo geometry. Seitz and Dyer [1996] extended this method to *view morphing*, where the real views are pre-rectified to standard stereo geometry before disparity interpolation. This allows for perspectively correct linear interpolation between tilted cameras P_l and P_r (see figure 3.6(a)).

In the rectification step, each real camera image is rectified by homography mapping onto a rectified image plane that is parallel to the baseline $\mathbf{C_r} - \mathbf{C_l}$ between the cameras, and epipolar lines correspond to horizontal pixel coordinates. The disparity between corresponding image pixel $\mathbf{m_l}$ and $\mathbf{m_r}$ is simplified to the displacement of x-coordinates $d = x_r - x_l$. Linear interpolation of the rectified image onto a virtual camera centre $\mathbf{C_v}$ is performed on the x-coordinates only:

$$x_v = x_l + \frac{|\mathbf{C_v} - \mathbf{C_l}|}{|\mathbf{C_r} - \mathbf{C_l}|} d \tag{3.4}$$

Cooke et al. [2002] use view morphing in 3D-video conferencing. From segmented disparity maps and images, special representations with low redundancy are assembled for transmission.

Proper interpolation is possible if the disparity of each pixel is known. Finding correspondences from the image data is the most critical task. Problems occur at object boundaries where the background is uncovered or occluded in one of the



(a) Geometry for view morphing (b) Rectified interpolation

Figure 3.6: (a) View morphing interpolates along a linear camera path for arbitrary camera orientation. The (dark) background object is partially occluded in the real images due to the (light) foreground object, hence some image regions may not be interpolated. (b) Interpolation geometry for rectified images is simplified to horizontal interpolation.

images. In those regions no correspondence can be established and holes may occur in the interpolated image.

3.3.2 Image Transfer Methods

Image Transfer Methods generate new images by a direct mathematical correlation between pixels of the source images and the pixels of the destination image. No intermediate representations are used.

The fundamental geometric relations between two and three images can be exploited for view synthesis. Disparity interpolation actually is such a method where the 2D correspondence is separated into epipolar line (direction) and disparity (length). In uncalibrated systems, the fundamental matrix (chapter 6) can also be exploited to define the epipolar geometry.

Image transfer can also be used to extrapolate view, thus allowing more freedom in virtual viewpoint selection. Laveau and Faugeras [1997] use epipolar transfer to extrapolate novel views from a set of given reference views. For the virtual view, epipolar geometry is computed with two reference views and correspondences are searched for in those views. A combination of disparity estimation and image warping has been proposed by Schaufler and Priglinger [1999]. Restricting the disparity search along epipolar lines reduces the complexity to $O(n^2)$ from $O(n^3)$.

Avidan and Shashua [1997] introduced *Trifocal Tensor Warping* to compute

correspondences with the trifocal tensor T, see chapter 6. T defines the fundamental geometry for three views, which describes the image transfer between these views. If the tensor between three views and the correspondence between two views is known for calibrated cameras, one can directly specify the correspondence in the third image and transfer the image intensity to the third image. The third view can be used to define the virtual camera and trifocal image transfer can be applied to synthesise the novel view.

3.3.3 Depth-Based Extrapolation

So far, image correspondences have been used to compensate scene parallax. As discussed in section 3.3.1 (figure 3.6(a)), occluding boundaries will block parts of the view and leave unmodelled regions that cannot be interpolated from the image data.

If a dense depth map is given, novel views can be synthesised using depth-compensated warping. A depth map contains the scene depth for every image point of a given view. Sometimes it is called 2.5 D view, as it contains 3D information, but from a single view point only.

Depth could come from additional sensors like a range scanner, or from stereoscopic depth estimation from multiple views. Dense correspondence estimation from a single image pair will always leave some background regions undefined, but the fusion of multiple views into a unique depth map allows dense depth computation [Koch et al., 1998].

If the disparity of a pixel is known and the relative pose between the real cameras, the depth can be computed. If more than two views are available, multi-view stereo approaches can be used to handle occlusions, and holes in the disparity maps can be filled from other views. The typical result of a such a stereo algorithm is a dense depth map: an image which codes the distance of each pixel in the original image. Having a per-pixel depth, the movement of the virtual camera is not restricted and allows to extrapolate novel views from one single image and the associated depth map.

The simplest method is forward-warping of the pixel from the real view into the virtual view. A scene point $\mathbf{M} = (x,y,z)^T$ in euclidean notion is projected to an image point $\mathbf{m}$ with the given pose $(R,\mathbf{C})$ using equation (2.13):

$$\lambda \cdot \mathbf{m} = KR^T(\mathbf{M} - \mathbf{C}). \tag{3.5}$$

λ is the pixel depth value which is lost during projection. Having the projective depth λ for the pixel from depth estimation, the 3D scene point $\mathbf{M}$ can be reconstructed as given in equation (2.20)

The desired point $\mathbf{m_v}$ in the virtual camera P_v is then determined by projecting $\mathbf{M}$ into the virtual camera (see figure 3.7(a)):

$$\mathbf{m_v} = P_v\mathbf{M} = \lambda \cdot (K_v R_v^T)(KR^T)^{-1} \cdot \mathbf{m} + (\mathbf{C} - \mathbf{C_v}). \tag{3.6}$$

Equation 3.6 is another form of epipolar pixel transfer from the real to the virtual view. In fact, it directly describes the epipolar line with z as free parameter.

One implementation of the described warping is *Depth-Image Based Rendering* as proposed by Fehn [2004]. It generates stereo image pairs from one monoscopic image and a depth map.

Depth extrapolation does not solve the occlusion problem, because parts that are occluded in the real view may become visible for a new view. Without any further information this results in holes or artefacts even if the depth map is completely filled. Having depth maps for adjacent views, forward mapping can be applied for these too, which can solve the occlusion problem. But forward mapping from multiple images is redundant for non-occluded regions and can also produce holes if the resolution of the source and the destination image are different.

3.3.4 Layered Depth Images

To overcome the problem of occlusions, *Layered Depth Images* (LDI) were introduced by Shade et al. [1998]. The idea is to generate a multi-valued depth map combining depth information from several real views into a single LDI-view by fusing depth information and adding additional layers for each pixel corresponding to the different layers in the scene. In this way, both foreground and background object are stacked in the same representation and occlusion artefacts can be avoided.

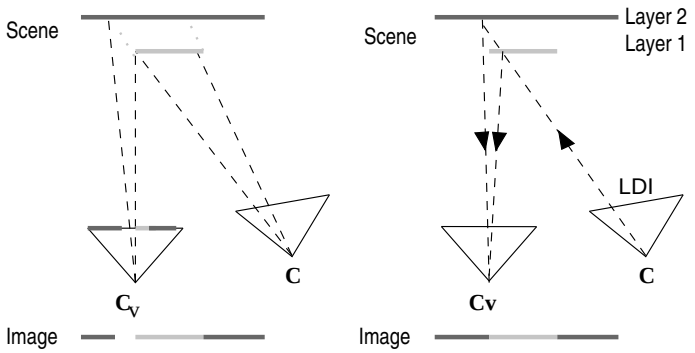
Chang and Zakhor [1999] have also proposed similar multivalued representations. In an acquisition phase, dense depth maps are computed and planes are fitted in low contrast regions. These planes are identified and tracked trough the image sequence. After transformation of all depth maps into one reference frame, a multi-layered representation with both colour and depth information is constructed. View generation from multi-layered images can be performed by forward-mapping each pixel of each layer via its depth information. Care has to be taken to ensure that nearby objects are not overdrawn by background objects. The layers have to be traversed in back-to-front order and depth-tests are needed when updating pixels in the destination image (see figure 3.7(b)).

LDI representations can be computed from multiple real views simultaneously if a multi-camera plane-sweep algorithm is used [Collins., 1996]. A plane is positioned in space, all real images are projected onto the plane, and the image consistency of the projection for all images is tested. If all projections yield a consistent

photometric measure (the photo-consistency), the colour and depth of the projection is saved in the LDI.

Yang et al. [2002] use a plane-sweep directly for rendering from given real views based on photo-consistency. Instead of computing an LDI, they directly render the colour and depth values of the plane sweep into the new view. For each pixel, photo-consistency decides if it is accepted or rejected. Then the plane is moved along the optical axis of the virtual camera. This process is repeated until the virtual view is completely filled.

A slightly different method of *texture slicing*, also known as elevation maps, is proposed by Vogelgsang and Greiner [2003]. The per-pixel depth is stored in the alpha-channel of the textures and a plane-sweep algorithm uses alpha test to decide which pixel to render. The planes are rendered in depth intervals which are calculated by taking projection errors into account. If a pixel on a plane corresponds to a depth map pixel, the colour is taken from the associated pixel in the real view.



(a) Depth Based extrapolation (b) Multi-layered extrapolation

Figure 3.7: (a) Using depth maps allows extrapolation, occluded regions have to be filled from other cameras. (b) Fusing depth from multiple cameras into a multilayer representation allows extrapolation from the LDI without gaps.

3.4 Rendering from Approximate Geometry

Multi-layered depth images are a first step towards globally consistent models of the scene. Many other methods have been proposed to generate intermediate representations from the images and depth maps. This can be called *modelling*, because a static 3D model is created. If the resolution of the model is quite rough it is sometimes referred to as a *geometry proxy*, a geometrical approximation.

3.4.1 Planar Scene Approximation

LDI representations use plane hypotheses that are perpendicular to the optical axes. Often, the scene is rather restricted in depth. If the region of interest is roughly of continuous depth to the camera, a planar approximation may help to compensate parallax effects. One example is the Lumigraph approach where the scene geometry is approximated by a fitting plane that coincides with the (s,t) image plane. The scene geometry can now be thought of as decomposed into some fitting planes and additional local parallax components. Virtual views are interpolated by rectifying the real views into the virtual view by way of planar homography mapping. The deviations of the real scene depth from the fitting planes will cause image distortions due to the local parallax effects. If the fitting plane describes the dominant scene geometry well, global homography mapping is a suitable rendering method. Buehler et al. [2001] and Bolles and Baker [1986] use this method.

3.4.2 View Dependent Texture Mapping

If a consistent 3D surface is available, traditional polygon rendering techniques can be applied. In conjunction with dense input imagery, the polygon model can be used as approximate global geometry, and the real views that are nearest to the virtual view are mapped onto the surface. This is called *View Dependent Texture Mapping* (VDTM) and has been introduced by Debevec et al. [1998]. VDTM bridges the gap between traditionally textured surface models and IBR with approximate geometry.

3.5 Dynamic Scenes

The rendering from static scenes is now well understood. Challenges lie in the domain of dynamic scene modelling for 3D television and Free-viewpoint Video.

Generation of virtual views for dynamic scenes is not totally different from static scenes. Each of the discussed static methods could potentially be used for dynamic scenes on the assumption that it can be implemented to run fast enough and to handle the amounts of data. One trend in the last years has been to transfer significant computational load to the graphics GPU hardware. For visualisation of polygonal models this is obvious, but today's programmable GPUs allow much more complex algorithms to be executed. For nearly every presented method, exists a real-time version using GPU support. Real-time interaction in this case means, that the user can move the virtual camera interactively and the virtual views are generated with 10 to 50 frames per second. Most often, pre-computation

requires much more time. Dense depth estimation, camera pose estimation, calculation of optimised intermediate data structures are expensive tasks. The step towards dynamic scenes requires to reduce and speed-up these steps so that video sequences from several cameras can be processed and the user can control the virtual camera at interactive rates.

Some recent real-time systems use volumetric modelling. Volumetric models are similar to medical computer tomography data-sets. Typically, the 3D space is partitioned into volume elements called voxels. For each voxel it is determined if it belongs to an object or not. This representation can be constructed from depth maps, too, but the more popular approach is called "shape from-silhouette". A visual hull is computed for objects by intersecting the silhouette cones from different real views. Li et al. [2003] use shape-from-silhouette to construct the visual hull by back-projecting images and using alpha and stencil calculation. To generate novel views from volumetric models again different methods are available.

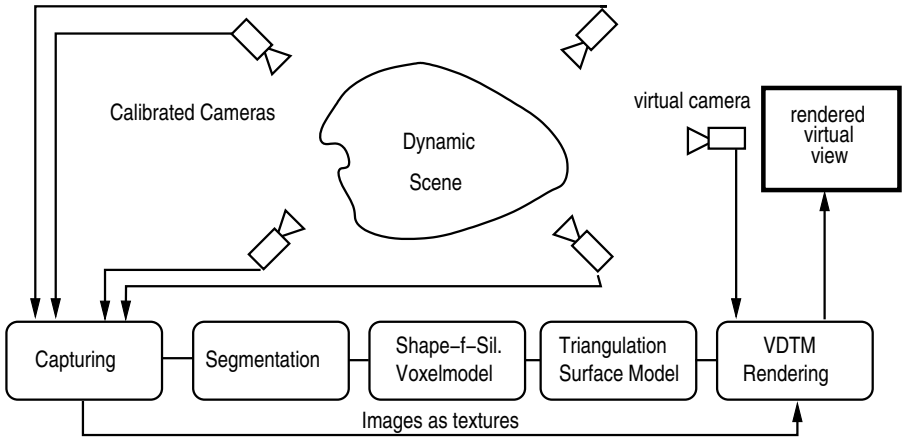


Figure 3.8: IBR pipeline for view generation with visual hulls.

One common approach for IBR view synthesis of dynamic scenes is shown in figure 3.8. The pipeline processing starts with capturing the scene using multiple calibrated fixed cameras. After object segmentation, shape-from-silhouette algorithms are used to create a volumetric model on the fly. After conversion to a surface model using polygon meshing the standard polygonal rendering with view dependent texture mapping is used for visualisation.

Saito et al. [1999] use 49 calibrated cameras, and compute a volumetric model. This is transferred into a polygonal surface model and during rendering it is used to generate correspondences in selected real views. From these correspondences per-pixel interpolation is performed after the determination of the disparity vectors.

Yamazaki et al. [2002] have introduced *billboards*. A billboard or *micro facet*

is a small polygon always facing the virtual camera. They approximate surface models, re-sample them to binary voxel-models and create a multi-resolution-octree. Depth maps are used for per-pixel visibility-culling to prevent texturing facets with inappropriate pixel.

Goldluecke and Magnor [2003] calculate volumetric models from different views with a Shape-from-Silhouette approach. This model is then rendered using billboards textured from original views. It is also possible to convert volumetric models into surface models, one approach is discussed in chapter 3.

Recently, direct depth-based view interpolation for dynamic scenes has been proposed by Zitnick et al. [2004]. They capture dynamic scenes from a set of fixed video cameras and compute depth-compensated view interpolation from multiple views interactively. The results look promising and show that indeed the challenge of interactive free-viewpoint video can be mastered in the near future.

The main challenge in capturing and visualising dynamic content is the amount of data to store and process. In contrast to stationary scenes one camera moved to scan the scene does not suffice. To decouple time and space the plenoptic function has to be sampled at different positions over the required time. This means that many cameras have to operate in synchronicity. The image feeds have to be stored, off-line processing has to be done and during rendering, data from a high dimensional function have to be accessed. Shum et al. [2005] devoted one part of the book only to compression schemes for plenoptic data from dynamic scenes. This thesis restricts itself to stationary content, but the proposed methods can be extended to dynamic scenes. Adding sophisticated memory management and compression techniques would suffice.

Chapter 4

Design of Plenoptic Rendering Methods

After reviewing the work of others in the field of Image Based Rendering, this chapter deduces the design of new rendering methods from the goals which should be met and the requirements which should be fulfilled. The design and the requirements also lead to the specification of input data required for rendering. Methods to prepare the input data are presented. The final design of the rendering methods is chosen to handle errors in the input data that can be put down to preprocessing gracefully.

4.1 Primary Design Goals

As seen in chapter 3, many different systems have been proposed. This variety has been provoked by the following aspects which greatly influence the possibilities and the resulting quality of the generated images: scene or depth complexity, sampling density, movement of the capture device, possible movements of the virtual camera. Each of the reviewed methods implies limitations in one or several of these aspects. Obviously, Image Based Rendering is always limited by the plenoptic samples: Invisible parts of the scene cannot be reconstructed. But besides this, the primary goal of this work is to design IBR methods which do not have any further limitations. In relation to the previously mentioned aspects, the following goals can be defined:

Scene complexity The methods should not be focused on special setups like single objects, planar scenes or special non-planar scenes. The only assumption is that the scene is stationary, i.e. objects do not move and the lighting conditions do not change.

Movement of virtual camera: Apart from the limitations given by the sampling, the possible position and orientation of the virtual camera should not be restricted (e.g. rotation only, predefined path). Only parts of the scene which are visible in the original views can be reconstructed.

Sampling density Dense sampling can simplify the interpolation but complicates the capturing process, if it is feasible at all. In that sense, a sparse distribution of real views has to be assumed, starting with only 2 photographs and ranging to a scan with a multi-camera system.

Geometry Geometrical information should be gained from the real views alone and will be used when available. But partially missing or incorrect depth information has to be tolerated and compensated for as well as possible.

In addition to these specifications, the rendering process should be as fast and interactive as possible. If best-quality image generation cannot be done at interactive frame-rates, the process has to be scalable to let the user choose between high quality or interactive frame-rates. For a given configuration of images and free movement, the best possible image quality should be aimed at both for interpolation and extrapolation. Nevertheless the configuration or the virtual camera may cause different errors in the novel image.

4.2 Required Input Data

4.2.1 Calibration

A necessity for Image-Based Rendering is the calibration for each image. The parameters and models which describe image formation (section 2.2) are needed to incorporate colour information as samples of the plenoptic function. The calibration process can be seen as a parameter estimation of the plenoptic function sampling process. Intensity values of rays are known from the images but the geometrical parameters of the rays are unknown. for only one particular ray, these parameters cannot be reconstructed. But for images from well-defined cameras at several points in space, enough constraints apply to estimate the position and orientation of the cameras giving the parameters for each observed ray. The following sections 4.2.2 and 4.2.3 describe a method to compute the calibration from the images themselves.

4.2.2 Calibration for Static Setups

For static setups with multiple fixed cameras all parameters can be calibrated before capturing the scene. This can be done by placing a well-defined calibration

object in such away that it is visible in the cameras [Tsai, 1987]. With known 2D-3D correspondences between the calibration object and the images as seen by the cameras, the intrinsic and extrinsic projection parameters for all cameras can be computed (see also [Horn, 2000, J. Heikkilae, 1997]).

After the calibration process the calibration pattern can be removed and on the assumption that the camera parameters are not changed the relevant scene can be recorded .

4.2.3 Calibration for Mobile Setups

Using mobile (hand-held) cameras to scan the scene, calibration objects are only an option if they can remain in the scene. Otherwise, the projection parameters for all real views have to be computed from the images of the scene alone.

Development of so-called *Structure-from-Motion* approaches as described by Pollefeys et al. [1999], allow to estimate camera parameters from the image sequence itself without special markers or patterns. Even changing and unknown intrinsics can be handled, but it often stabilises the process if the intrinsics are calibrated beforehand and the Structure-from-Motion approach only computes the extrinsic parameters (camera poses).

The calibration starts by identifying and tracking salient features through the image sequence. For the first two views, the 3D positions of the points and the position and orientation of the two observing cameras can be computed simultaneously in a bootstrap process exploiting epipolar constraints [Hartley and Zissermann, 2004]. For all following views, 2D feature correspondences are created by tracking or matching features from already calibrated images into new images. This yields 2D-3D correspondences to the already reconstructed 3D points. From these 2D-3D correspondences, the camera pose can be computed by non-linear minimisation of the projection error [Hartley and Zissermann, 2004]. Because this minimisation is very sensitive to wrong correspondences (outliers), a robust RANSAC-based pose estimation [Fischler and Bolles, 1981] has to be used and outliers have to be removed before the minimisation. Having the pose for the new view, all 2D observations can be used to refine the position of the 3D points or establish new 2D-3D correspondences by triangulation.

Without any knowledge about the scene or the camera setup, a reconstruction can only be done up to an unknown scale. Because the size of the scene and the distance to the cameras is ambiguous, the absolute scale cannot be determined. To have some defined lengths, camera positions and 3D point positions are scaled consistently so that the distance between the first two views equals 1. This scale ambiguity has no direct effect on Image-Based Rendering, because it is an image-to-image transfer method. By reconstruction of a scene representation, a scale is assumed and by projection into a virtual camera, the absolute size of the scene

is removed. Apart from that, the scale of the reconstruction has to be taken into account when mixing Image-Based Rendering images with other virtual objects.

4.2.4 Why Scene Geometry is required

The basic input data for Image-Based Rendering are plenoptic samples or, in other words, calibrated images as described above. One question here is: How many images are required, or which sampling density is required?

The answer to this question depends on the amount of available geometrical knowledge of the scene. If a full 3D model can be used, very few images suffice to generate arbitrary new views of high quality. The situation where no geometry information can be used can be analysed by a small thought experiment.

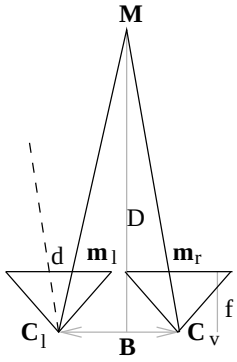


Figure 4.1: Two cameras in standard stereo geometry observing a point M in distance D.

We assume a structured scene which is observed by a camera in one meter distance. This camera is moved in x-direction of its own camera coordinate system by small distances to capture images for view interpolation. The camera is not rotated, thus the optical axes of all views are parallel. A virtual camera is placed directly between two real cameras and no depth or disparity information is to be used. The only interpolation which can be performed is blending the images of the surrounding views together. To avoid any visible interpolation artefacts, the maximum disparity between one real view and the virtual view, which can be compensated, is below one pixel. The mathematical relation between depth and disparity in this special case is:

$$\frac{D}{B} = \frac{f}{d} \tag{4.1}$$

with D the distance (depth) of the 3D point from the baseline, B the length of the baseline connecting the centres of two cameras, f the focal length and d the disparity in the image plane. The setup is shown in figure 4.1. The question

now is: What is the maximum distance between adjacent views to ensure that the maximum disparity is smaller than 1.0 pixel? This question can be answered using equation (4.1) and inserting some realistic values. Small digital cameras have a pixel spacing of $5 \cdot 10^{-6} m$ which is the disparity of one pixel. A typical lens for such a camera has a focal-length of $5mm$ and the distance D is assumed to be $1m$. This results in a baseline length B of $1mm$.

Conclusion: Without any geometry information, "dense" sampling has to be used, which means placing a camera every $1mm$. As we expect a two dimensional distribution of real views, a regular grid of that spacing has to be used. Due to physical size of cameras, this cannot be built as multi-camera setup. The only possibility to realise this would be to have a single camera moved precisely by a robot.

With increased distance between real views, new views can only be interpolated properly, if information about the structure is available. This information can be a perfect or approximate model, depth- or disparity information. For arbitrary scenes, typically no model is available, but depth or disparity information can be computed from the respective images.

Disparity-based view interpolation methods have been presented in section 3.3.1. Using only disparity restricts the position of the virtual camera: $\mathbf{C}_v$ must lie in between the real views (interpolation). From disparity, depth information can be computed for each pixel. This type of geometry representation does not restrict the position of the virtual camera and also allows extrapolation. In the following, only depth information is considered. This can be computed beforehand in a preprocessing step to avoid full disparity search during interactive rendering.

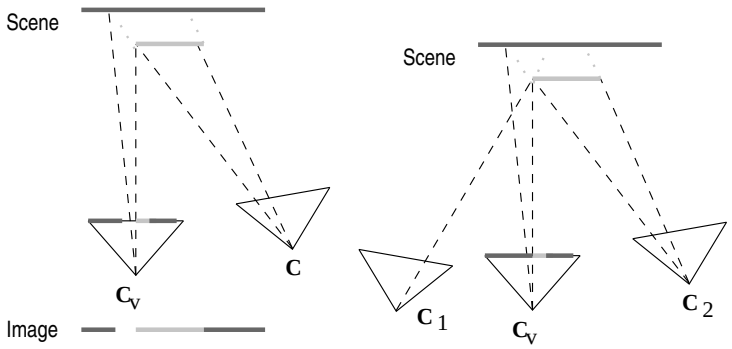
4.2.5 Properties of Depth Maps

Precise and complete depth information for a real view means to have a depth value for each pixel of this view forming a depth map. A depth map looks like a grey-value image where the level of grey codes the distance of the scene. Light values represent larger distances than dark values.

The perspective depth value corresponds to the λ in equation (2.6). The knowledge of λ allows to use equation (2.20), reconstruct the 3D point $\mathbf{M}$ and warp the colour information of every pixel depth compensated into any new virtual view. This allows to extrapolate novel views from existing ones.

Depth maps computed from image sequences typically associate one depth value with each image pixel. They only encode the distance of the camera centre to the front-most surface of the scene. Other layers or objects remain occluded, they are not visible in the image. On the assumption that the depth map does not contain any depth discontinuities, the continuous surfaces can be warped into

different views without errors. This allows to reconstruct closed single objects easily.



(a) Depth Based extrapolation (b) Multi-layered extrapolation

Figure 4.2: (a) Using only one depth map, problems occur from occluded regions . (b) Using depth from multiple cameras can fill these gaps.

If the scene is composed from multiple objects which occlude each other partially, depth discontinuities appear in the depth map. If a novel extrapolated view can see "behind" the occlusion, no colour nor depth information is available to fill this gap (see figure 4.2(a)). These regions in the novel view cannot be filled without additional information.

4.2.6 Depth Estimation

The Structure-from-Motion algorithm gives a sparse 3D representation of the scene consisting of the reconstructed 3D positions of the tracked features. Dense depth estimation is used to compute the projective depth λ from eq. (2.6) for each pixel of an image. Stereo algorithms work on image pairs and for each pixel in one image, they try to find the corresponding pixel on the epipolar line in the second image. This yields the disparity for each pixel. As we have the camera calibration and the disparity, the 3D position of the associated point in the scene can be triangulated and the projective depth can be computed.

Multi-view Stereo uses 3 or more images to stabilise the depth estimation and to handle occlusions. For one reference view, correspondences are searched for in 2 or more support views. The result of dense depth estimation is the depth map encoding the projective depth λ for every pixel in the image.

Many different algorithms have been proposed for stereo or multi-view stereo [Koch, 1996, Roy and Cox, 1998, Yang and Pollefeys, 2003, Seitz and Dyer, 1997b]. The plane-sweep algorithm presented in section 5.4 can also be used to compute depth maps.

In the following, it will be assumed that calibration and dense depth estimation have been performed on the image sequence and the depth maps along with the calibration and the images are considered to be input data for the modelling and rendering.

4.3 Expected Errors in Input Data

Due to the fact that image sequences from real cameras are used, input data for novel view generation as discussed in this thesis can be erroneous. Two error sources, traceable back to the preprocessing steps, have been identified: The image calibration and the depth estimation.

4.3.1 Calibration Errors

The calibration of cameras with a calibration object or a pattern is assumed to be robust and good enough so that the remaining error is small enough not to influence view interpolation. Typically, an error of about one pixel remains invisible. But when freely moving cameras are used only the intrinsic projection parameters are calibrated with patterns. The calibration of the extrinsic parameters with a structure-from-motion system can introduce errors in the position and orientation of each real view.

Caused by the minimisation process which changes extrinsic parameters to minimise the distance between the 2D image points and the projected 3D points, typically each positional error in x or y direction (camera coordinate system) is accompanied by a rotational error in opposite direction and vice versa. Errors in z-direction (along the optical axes) do not necessarily correspond to a rotational error.

For view interpolation the absolute calibration error for each camera pose is not as relevant as the relative error between adjacent views. If all views share a common offset, the view generation will not produce visible artefacts, but if the relative calibration between two views is not correct, the interpolation will produce errors from blending wrong colours. Structure-from-Motion typically produces very small relative calibration errors between local views.

A source of calibration errors on large image sequences is drift. If each view has a small calibration error, these can accumulate over all views and result in severe camera displacement. For long linear scans this does not necessarily impose artefacts in the generated new views because adjacent views used for local interpolation only have a small relative registration error. Using zig-zag scans registered in a time-linear fashion without inter-line stabilisation, spatial adjacent views can have significant drift. To calibrate two dimensionally distributed scans,

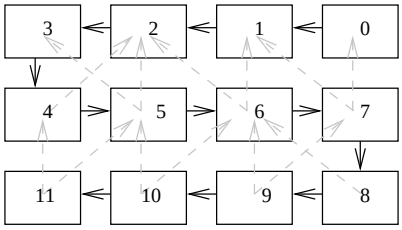


Figure 4.3: Calibration of a Zig-Zag scan can be done time-sequentially (black arrows). Using already calibrated geometrical adjacent views (grey arrows) helps to reduce drift significantly.

Structure-from-Motion systems have to establish topology information and calibrate cameras with respect to their geometric neighbours (figure 4.3)[Koch et al., 1999].

Error accumulation from the calibration can also be compensated with a bundle adjustment approach [J.C.McGlone, 2004], where all camera parameters and all reconstructed 3D points are taken into account to minimise re-projection errors.

4.3.2 Depth Estimation Errors

The depth estimation process for one particular real view uses adjacent real views to find pixel correspondences. Compared with the overall number of real views in a data set, this can be assumed to be a local operation. Thus, the influence of error accumulation in the calibration phase can be neglected. Local consistency suffices and can be assumed because the structure-from-motion algorithm itself ensures it. Even if a data-set's calibration includes drift, depth maps can be computed and will be correct for this local view. On the other hand, this implies that two locally correct depth maps from drifted calibrations do not give a globally consistent description of a scene. This fact complicates the fusion of all depth maps into one global 3D model.

Another problem is that depth estimation is typically done by discrete disparity estimation. Due to pixel quantisation, disparity can only be computed with limited precision (typically one pixel). When this disparity is transferred to depth, the quantisation results in better depth resolution near the camera and less depth resolution in regions further away. By fusing multiple depth maps this quantisation can be reduced significantly. For one particular view this can be accomplished by computing multiple depth maps from different reference views and computing the mean depth for each pixel.

The major sources of error in depth maps from real cameras are unreconstructed regions and regions with completely wrong depth. In many algorithms

the fill-rate and the reliability are contrary goals which cannot be optimised at the same time. A trade-off between those two has to be found.

Unfilled pixels typically occur at object boundaries with depth discontinuities. For two-view stereo some image parts cannot be matched because they are occluded in one image. Multi-view approaches have the ability to fill regions from other views. Most stereo algorithms cannot estimate reliable depth in homogeneous image regions because of the missing structure. Other algorithms propagate depth information into these regions to fill them, but that may produce unreliable results. Object surfaces with non-lambertian properties (e.g. glass, polished metal) typically cannot be reconstructed because the basic assumption in stereo matching is that the appearance of objects does not change between views.

Small regions of completely wrong depth estimation can result from mismatches. This means that the correspondence search has found a wrong pixel correspondence which satisfies all constraints. In depth maps this can be observed as small light spots in dark regions or dark spots in lighter regions (often called *salt and pepper noise*). An increased correlation window size reduces the matching uncertainty and increases the position uncertainty.

4.4 Design of Methods

To solve the visual reconstruction problem under the constraints described in section 4.1, new approaches have to be developed. The geometry information represented by the depth maps has to be used to forward-map the colour information from the real view into the desired virtual view. The following three methods have been developed to fulfil the above requirements. Each method is described in detail in chapter 5.

4.4.1 View-Dependent Geometry

The main idea behind *View-Dependent Geometry* is to find a surface which approximates the scene geometry and exactly fits the visible part of the screen space of the virtual camera. The real images can then be transferred into the virtual camera via this surface.

Because it is very difficult and most often not possible to construct one single globally consistent surface, a locally valid surface has to be adapted for each new view (this property gives the name of the method). This surface does not need to cover the whole scene, it suffices to cover the currently visible parts. Starting with a regular 2D mesh in the image plane of the virtual camera, the depth for each vertex has to be determined from depth maps of surrounding cameras. The naive

idea of backward-mapping[1] from the virtual into the real cameras is not possible due to the missing depth. Only forward-mapping[2] from the real cameras into the virtual camera is possible. One problem which occurs here is that this geometry transfer and geometry fusion is expensive. To reach interactive rendering rates, the geometry has to be sub-sampled significantly. This leads to a smoothed warping surface and does not reconstruct details.

4.4.2 Multiple Local Models

Forward-mapping and fusing dense geometry is possible at interactive frame-rates when the GPU is utilised. This leads to an approach called *Multiple Local Models*: For each depth map a local 3D model is generated in a preprocessing step. A new image is generated by rendering models generated for adjacent real views into the virtual view. The contributions from all rendered models are blended appropriately.

Different modelling approaches have been implemented: point-based modelling, polygon mesh modelling and adaptive surface reconstructions. Each of them has special properties which will be presented in section 5.3 and the results will be discussed in chapter 6.

4.4.3 Plane-Sweep

The Multiple Local Models approaches all rely on dense and reliable depth information. If the quality of the depth maps is not good enough, the reconstruction quality decreases significantly. The *Plane-Sweep* reconstruction method known from Yang et al. [2002], on the other hand, does not need any depth information, it aims at finding photo consistency. Combining precomputed 3D information with a Plane-Sweep method [Collins, 1996] leads to the development of a *geometry guided Plane-Sweep*. Photo consistency[3] constraints [Seitz and Dyer, 1997a] are used to verify and refine the depth where available or reconstruct the visual appearance of regions without depth. In contrast to the standard Plane-Sweep method the search range can be limited in regions with approximate depth. This can increases the quality by reducing matching ambiguities and reduce the required computational effort at the same time.

[1] backward-mapping: For a pixel position in virtual camera, find the corresponding position in a real camera.

[2] forward-mapping: For a pixel position in a real camera, find the corresponding position in the virtual camera.

[3] A 3D point is photo-consistent if its projection into multiple images leads to the same pixel colour in each image.

4.5 Summary

In this chapter, the design goals for the view generation have been presented. Analysing the required input data consisting of images, calibration and depth maps reveals two different error sources: calibration and depth estimation. This leads to the design decision for three Image-Based Rendering methods, namely View-Dependent Geometry, Multiple Local Models and Plane-Sweep. In the next chapter, each of these methods will be presented in detail.

Chapter 5

Rendering Methods

Due to the large variations in input data, no single Image-Based Rendering method which fulfils all needs has been developed so far. The development of plenoptic rendering methods presented in this chapter can be seen as the main contribution of this thesis. To meet the requirements listed in section 4.1, three different prototypes of plenoptic rendering methods have been developed:

Prototype $P1$ (section 5.2): For each novel view, an approximate geometry is constructed and used to warp the real images into the virtual camera. This is called *View-Dependent Geometry*.

Prototype $P2$ (section 5.3): For each real view, a model is computed and selected models are blended to form the novel view. This prototype is called *Multiple Local Models* and three different subtypes ($P2_a$,$P2_b$,$P2_c$) are introduced.

Prototype $P3$ (section 5.4): New views are generated by finding photo-consistency between real views with a so-called *Plane-Sweep* algorithm. Without depth information this leads to subtype $P3_a$. Having depth information helps to stabilise and accelerate the process and leads to subtype $P3_b$.

All protoypes and their subtypes (shown in figure 5.1) are presented in detail in this chapter. But first some common tasks shared by these methods are described.

5.1 Common Tasks

5.1.1 Camera Ranking

One common task for several rendering methods described here is the selection of real views to interpolate from. For each novel view to render, a set of real cameras

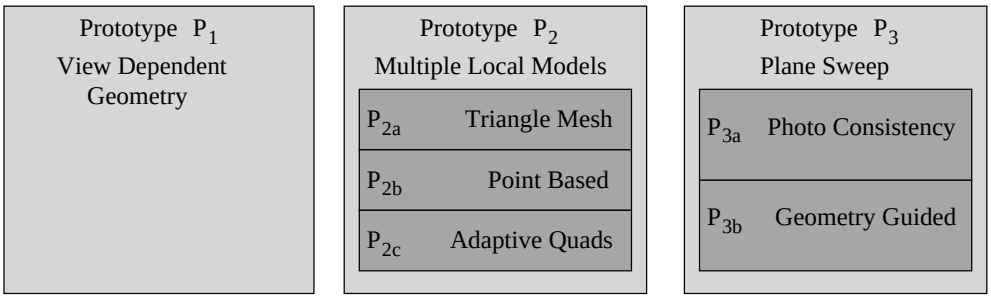


Figure 5.1: Three different prototypes and several subtypes have been implemented.

for interpolation has to be chosen. Several criteria are relevant for this decision. In [Evers-Senne and Koch, 2003] we have developed a ranking criterion for ordering the real cameras. The user chooses the number of neighbouring cameras that should be used, and, depending on the ranking, the best N cameras are selected. Two relevant criteria are presented in the following.

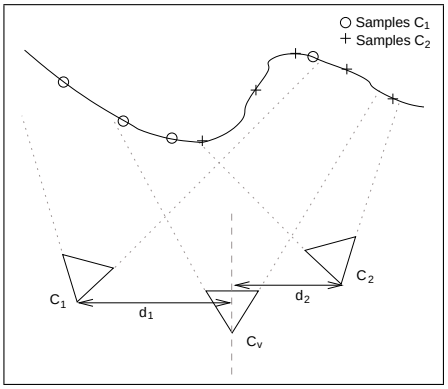


Figure 5.2: Criteria evaluation. The ranking criteria are shown for two real cameras C_1 and C_2. *Distance:* C_2 is closer to the optical axis of C_v then C_1. *Visibility:* C_v sees 3 samples from C_1 and 4 samples from C_2.

Distance: The first criterion is the *distance* between a real camera and the virtual camera. With the viewing direction $\mathbf{A}_v$ and the centre $\mathbf{C}_v$ of the virtual camera, the orthogonal distance d_i of each real camera centre $\mathbf{C}_i$ to this ray can be determined:

$$d_i = ||\frac{(\mathbf{C}_i - \mathbf{C}_v) \times \mathbf{A}_v}{||\mathbf{A}_v||}|| \tag{5.1}$$

Visibility: The second criterion is more complex, it is based on *visibility*. It evaluates the scene volume that a given real camera has seen in the context of the virtual camera. For this purpose very few ($n_{\max} = 20$) image points are chosen so that the whole image of the real camera is covered. By back-projecting them using the corresponding depth values *3D samples* are calculated. These samples are then projected into the virtual camera and checked for visibility. The number of visible samples n_i divided by the number of possible samples $n_{\max}$ gives a rough approximation of the region covered by a real camera. The squared ratio is used to penalise cameras with only a few samples more. The visibility term v_i will be defined as a penalty which decreases for increasing n_i:

$$v_i = 1 - \frac{n_i^2}{n_{\max}^2} \quad \text{with:} \begin{pmatrix} v_i = 0 \text{ :best} \\ v_i = 1 \text{ :worst} \end{pmatrix} \tag{5.2}$$

This yields a penalty function with $v_i(0) = 1$ (no common viewing volume) and $v_i(n_{\max}) = 0$ (100% overlap).

Figure 5.2 sketches the different selection criteria. Two real cameras C_1 and C_2 are evaluated w.r.t. the virtual camera C_v. For visibility, the depth samples from camera 1 (circles) and camera 2 (crosses) are projected into the virtual camera and evaluated.

Combination and Ranking: Similar to [Evers-Senne and Koch, 2003] the two criteria are combined into one scalar value q_i which represents the inverse quality of the real camera i to generate the new view. In contrast to the linear-weighted sum, the distance criterion is used primarily for ranking but modulation with the visibility penalty function ensures that only those cameras which can contribute to the new view are ranked:

$$q_i = d_i \, (1 + c \, v_i) \quad \text{with} \begin{pmatrix} q_i \text{ smaller: better} \\ q_i \text{ greater: worse} \end{pmatrix} \tag{5.3}$$

c is a parameter which determines the influence of the visibility. After calculating q_i for each camera, the list of valid cameras is sorted in ascending order.

5.1.2 Per-pixel vs. per-Camera Blending

In section 2.5, two different blending strategies have been presented: Per-pixel blending and per-camera blending. In this section problems with the per-pixel blending approach are discussed.

Experiments with the per-pixel blending have shown that problems occur with both blending functions from equation (2.29). If the virtual camera coincides with one of the real cameras (e.g. $\mathbf{C}_v = \mathbf{C}_1, \alpha_1 = 0$), one would assume the $\mathbf{C}_1$ has

a contribution of (or at least near) 100% and the next closest real camera $\mathbf{C}_2$ is weighted with 0. For $\alpha = 0$, the weights are 1:

$$f_1(\alpha) = \cos^k \alpha : f_1(0) = 1, \quad f_2(\alpha) = e^{-k\alpha} : f_2(0) = 1 \tag{5.4}$$

The contribution of $\mathbf{C}_2$ only depends on α_2 which results in 0 only if $\alpha_2 = \frac{\pi}{2}$. Due to incremental computation no normalisation can be applied.

By increasing k in equation (2.29), the desired blending behaviour can be approximated. But the choice of k depends on the distance of the cameras used for interpolation and therefore cannot be set constant.

The problem with this approach is based on the iterative accumulation based on the angles α_1 and α_2 only without taking the complete distance $\alpha_1 + \alpha_2$ into account. In contrast, the linear blending in equation (2.30) normalises the weights with the distance between the real cameras. This finally leads to the developement of the barycentric weighting scheme and the following complex camera selection.

5.1.3 Camera Selection for Blending

The mathematical task of computing correct blending weights has been described in detail in section 2.5. It is inherently connected with the camera selection. If one decides to use per-camera blending, the selection of the real cameras determines the weighting function which can be used.

The strict camera selection scheme as described above does not work well in conjunction with barycentric blending weights. If only the closest N cameras are selected, it is not certain that they span a convex polygon enclosing $\mathbf{C}_V$. For $N = 3$ convexity is always given (triangle), but depending on the setup, the virtual camera can be located outside of this triangle. For increasing N it becomes more likely that the resulting polygon is not convex and again it is not certain that $\mathbf{C}_V$ is inside the hull. But because interpolation ($\mathbf{C}_V$ inside the polygon) is preferred to extrapolation ($\mathbf{C}_V$ ouside of the hull), a more flexible approach is needed to fulfil the requirements for proper blending.

The first step for camera selection is to define a plane Π parallel to the image plane of the virtual camera with its origin located at $\mathbf{C}_v$, the projection centre of the virtual camera. $\mathbf{C}_v$ on the plane is called $\mathbf{c}_v$. Then all real camera centres are projected onto this plane orthographically. This can easily be done by applying the inverse rotation of the virtual camera and using only the x and y components of $\mathbf{c}_i$:

$$\mathbf{c}_i = (1,1,0)^T \; R_v^T \mathbf{C}_i \tag{5.5}$$

The distance $||\mathbf{c}_i - \mathbf{c}_v||$ is the same as the distance in equation (5.1) and the real cameras can be sorted according to their absolute distance.

The user can specify a number N of cameras which should be used for interpolation but the camera selection scheme can modify it to optimize the blending. The algorithm to select the best set of cameras depends on the number N:

***N*=1:** The nearest camera is activated and the weight 1 is applied.

***N*=2:** The nearest two cameras are selected and weights are computed based on the distances $\mathbf{c}_0$ and $\mathbf{c}_1$ (equation (2.30)).

***N*=3:** First, the nearest camera is selected, then two more cameras are chosen and it is checked, if $\mathbf{c}_v$ is inside the convex hull (a triangle). If this is not the case, the second and third cameras are altered and the check is repeated. If this procedure cannot find a combination of three cameras surrounding the virtual camera interpolation is not possible and extrapolation has to be performed.

If a convex hull of three cameras can be found, it is checked if $\mathbf{c}_v$ lies on one of the edges of the polygon. In fact, an ε-range around the line is taken into account. If this check succeeds, only the two cameras defining this line are selected for linear blending.

***N*>3:** The algorithm starts with $N = 3$ and then tries to activate the nearest one of the remaining cameras and compute the convex hull. If all active cameras are vertices of the convex hull and no camera is inside the hull, the algorithm proceeds with the next camera. If the addition of one camera results in one or more cameras located inside the hull, the last activated camera is deactivated and the algorithm stops.

This scheme ensures that all constraints necessary for the blending weight equation are fulfilled.

5.1.4 Texture Coordinates

To avoid explicit calculation of texture coordinates, automatic texture coordinate generation is used. This reduces the amount of data to be transfered significantly and is done by current graphics hardware at no extra cost. This approach has been presented by Segal et al. [1992].

Therefore the original image of the real view to use is assigned as current texture in OpenGL. Using automatic texture coordinate generation ensures that correct texture coordinates for each vertex are generated from the vertex itself. The vertex coordinate $\mathbf{M}$ is multiplied with the current texture matrix T and the result is taken to access the texture:

$$\mathbf{m}_{\text{tex}} = T\mathbf{M} \tag{5.6}$$

This is equivalent to equation (2.13) with $\lambda = 1$ in normalised device coordinates. It suffices to decompose P into $\mathbf{C}, R, K$ as in equation (2.13) and normalise K by the image size $(x_{\max}, y_{\max})$ to K_{tex} to get the normalised texture matrix T:

$$T = K_{\text{tex}}[R^T \,|\, -R^T\mathbf{C}] \quad \text{with:} \quad K_{\text{tex}} = \begin{pmatrix} \frac{f_x}{x_{\max}} & \frac{s}{y_{\max}} & \frac{c_x}{x_{\max}} \\ 0 & \frac{f_y}{y_{\max}} & \frac{c_y}{y_{\max}} \\ 0 & 0 & 1 \end{pmatrix} \tag{5.7}$$

Using T as texture matrix results in homogeneous texture coordinates generated on-the-fly. This maps the current texture projectively onto the structure.

5.2 Prototype $P1$ View-Dependent Geometry

The basic idea behind View-Dependent Geometry can be considered as *depth-compensated warping*. Similar to the Lumigraph by Gortler et al. [1996], a geometry proxy of the scene is used to warp real views as textures into the virtual camera. But instead of using one global 3D model, a local model of sufficient size for the virtual view is interpolated from the depth maps of the surrounding real views. This geometrical approximation is view-dependent, which means it has to be updated for each new view with the following steps:

- selection of best real views (section 5.1.1),
- fusion of geometry from the multiple views,
- viewpoint-adaptive mesh generation,
- viewpoint-adaptive texture blending.

The fusion of full resolution depth maps from several cameras is not possible in real time, so an optimised sub-sampled representation is generated off-line.

5.2.1 Sampling the Depth Maps

The depth maps serve as geometry input for the view-dependent on-line modelling that will be described in the next section. Using the depth maps directly with full resolution is not feasible due to the vast amount of data. Standard hardware does not support such high-resolution geometry for view interpolation. Therefore, each depth map is sub-sampled. In this case the sub-sampling is done by applying a regular grid with a spacing that is parametrised so that it can easily be adapted to specific needs. This grid is located in the image plane of the real camera. At each grid point, the surrounding region in the depth map is processed by a spatial

2D median filter to reduce the effects of outliers and to find the most probable depth. By taking the j-th 2D point $\mathbf{m}_i^j$ in the image plane of camera i and the corresponding distance λ_i^j from the depth map, the euclidean 3D scene point $\mathbf{M}_i^j$ can be calculated with equation (2.20). The 3D point $\mathbf{M}_i^j$ from camera C_i is called *geometry sample* j of camera i. For each grid point in each camera one geometry sample is created. Grid points for which no depth values can be found are discarded. Later on, in the rendering stage, the geometry in these regions is constructed from the samples of other cameras if possible. The valid samples serve as geometric approximations that are used to define the view-dependent interpolation surface.

To allow scaling between performance and quality, a *Level-of-Detail* (LoD) is introduced at this point. When the geometry samples are created , the desired number of levels $L_{\max}$ can be chosen. After their generation all geometry samples belong to level zero L_0 by default. To generate level L_{k+1} each second geometry sample from level L_k in both dimensions is moved to level L_{k+1}. Thus the number of geometry samples in level L_{k+1} is $\frac{1}{4}$ of the previous number of geometry samples in level L_k. The number of geometry samples n L_k is now reduced to $\frac{3}{4}$. Filtering the geometry samples before sub-sampling is approximated by the median filter which was applied when the geometry samples were generated.

For rendering with a specific LoD n, the geometry samples of several levels are used in combination. For the coarsest level only the geometry samples of $L_{\max}$ are used. For the next level $L_{\max-1}$, the geometry samples of $L_{\max}$ and of $L_{\max-1}$ are used. In general, if level L_n is requested, all levels L_x with $x \geq n$ are used.

The geometry samples associated to the calibrated views can now be used as input to the interactive rendering engine.

5.2.2 Multi-View Depth Fusion and Mesh Creation

Geometry samples from depth maps have been created in the above described preprocessing phase. The interactive rendering starts by selecting n cameras as explained in section 5.1.3. To generate a novel view, views from different cameras have to be fused into one locally consistent image. To efficiently warp images from different real views into the novel viewpoint a 3D warping surface approximating the geometry of the scene is generated. This surface is constructed in the image plane (2D) of the virtual camera and, after depth fusion, transferred into 3D space.

Starting from a regular 2D-grid that is placed in the image plane of the virtual camera this warping surface will be updated for each camera motion. The spacing of this grid $S = (s_x, s_y)$ with s_x, s_y in pixels can be scaled to the complexity of the scene. With each *grid point*, a 4-tuple $g = \{\mathbf{M}_g, \mathbf{m}_g, i_g, b_g\}$ is associated. $\mathbf{m}_g$ is

the 2D position in the image plane, $\mathbf{M}_g$ is the 3D point to be constructed, i_g is the number of the camera responsible for $\mathbf{M}_g$ and b_g is a flag marking this grid point valid or invalid. The b_g components of all grid points are set to the default value of $b_g = invalid$.

To fuse 3D information from the best N cameras, for each camera i the following algorithm is used.

- Each valid geometry sample $\mathbf{M}_i^j$ for camera i is projected into the virtual camera with $\mathbf{m}_i^j = P_v \mathbf{M}_i^j$ and the distance $d_i^j = ||\mathbf{M}_i^j - \mathbf{C}_v||$ is calculated.

- If $\mathbf{m}_i^j$ is not in the visible area, the following steps are skipped and the next geometry sample $\mathbf{M}_i^{j+1}$ is taken. If it is in the visible area, the nearest grid point g_n is selected. Due to the regularity of the grid, this is easily done with $g_n = \text{rnd}(\frac{\mathbf{m}_i}{S})$.

- If the current sample $\mathbf{m}_i^j$ has a smaller distance d_i^j to $\mathbf{C}_v$ than the selected grid points 3D point ($\mathbf{M}_{g_n} - \mathbf{C}$), the grid points data are updated from this geometry sample, otherwise the update is skipped. When a grid point is updated from a geometry sample, $\mathbf{M}_{g_n}$ is set to $\mathbf{M}_i^j$, $\mathbf{m}_{g_n}$ is adjusted to $\mathbf{m}_i^j$, the cameras index i is stored in i_{g_n} and the new distance value d_i^j is assigned to d_{g_n}.

The algorithm proceeds with the next geometry sample $\mathbf{M}_i^{j+1}$. Updating grid points only with geometry samples which are nearer to the virtual camera ($d_i^j < d_{g_n}$) ensures that occlusions are handled correctly. The geometry samples of the highest-ranked cameras are projected first. This ensures that the largest part of the new view is interpolated from the best cameras. Only in regions that are occluded for the best camera, a lower-ranked camera geometry sample is used. Figure 5.3 depicts this situation.

After projecting all geometry samples and adjusting grid points, most grid points are valid and contain 3D information suitable to represent the part of the scene visible in the virtual camera. But because some grid points could still be invalid and the 2D positions in $\mathbf{m}_{g_n}$ are fitted to projected samples, the connectivity of the grid points cannot be derived from the grid itself. With the use of the 2D position of all valid grid points, a 2D Delaunay triangulation[1] of all grid points in the image plane of the virtual camera is performed. Transferring this 2D mesh to the 3D points $\mathbf{M}_{g_n}$ gives a scalable approximation of the 3D scene with triangles. The approximation can be scaled with respect to the sampling density of geometry samples from the depth map (see 5.2.1) and the density of the grid points

[1] The word *triangulation* is ambiguous. In this context it means creating a triangle mesh.

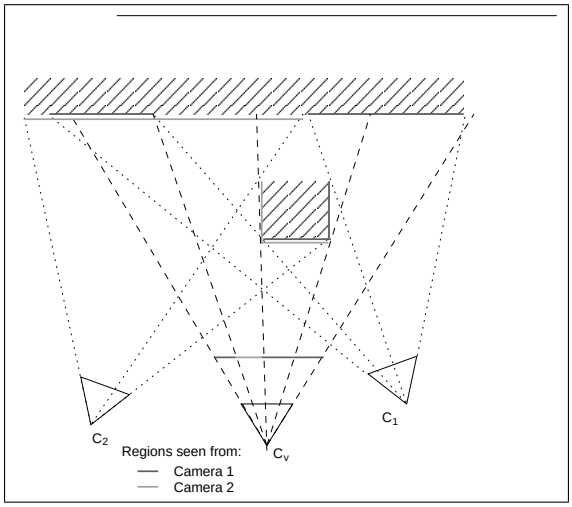


Figure 5.3: Geometry fusion from two cameras, with C_1 ranked higher than C_2. Based on the ranking, C_1 will supply most of the information. Only in parts that are occluded for C_1, data from C_2 are filled in.

for triangulation. This surface mesh is recreated after each camera movement as viewpoint-adaptive geometry.

5.2.3 Texture Warping

The texture warping step effectively maps the real cameras into the virtual view with the help of the viewpoint-adaptive surface mesh. Two slightly different methods for texturing are considered.

- The simplest one is to choose the best-ranked camera as texture source. If this real camera is not too far away from the virtual camera and both have a similar field of view, the results are good. This is the fastest texturing method since switching between different textures in one render cycle is not necessary and each triangle has to be drawn only once. Problems arise when parts of the mesh are not seen from the selected camera. These parts remain untextured.

- Because the triangle vertices are geometry samples where the originating real camera is known, all triangles can be textured according to the cameras where the geometry originated from. . However, since each vertex is generated independently, a triangle may have vertices from up to three cameras. Here one may decide to either select the best-ranked camera or to blend all associated camera textures on the triangle. Proper blending of all tex-

tures will result in smoother transition between views and can be executed concurrently with modern graphics hardware using different texture units.

5.2.4 Limitations

Having incomplete depth information, this method can help to generate completely filled images. But the process of sampling the depth map, fusing several sets of geometry samples in screen space by using another discrete sampling is crucial. Care has to be taken to avoid aliasing artefacts. The sampling rates have to be sufficiently high which increases the number of geometry samples. All processing is done on the CPU, which results in non-interactive frame rates for high-quality image interpolation. One other problem is that the information which vertex of the resulting mesh belongs to which real view does not suffice to blend the texture on the triangles appropriately. For exact blending, weights for each triangle point have to be computed.

5.3 Prototype $P2$ Multiple Local Models

Multiple Local Models means that for each real view a local 3D model is created offline and new views are generated by rendering and blending a number of those models. This is a different approach than View-Dependent Geometry, where one single geometry approximation is created for each new view.

To avoid problems caused by sub-sampling the depth maps and resampling all depth information in screen space, all available depth information should be used. This amount of data can best be handled on the GPU. To process geometry with a GPU, vertices and primitives have to be generated. Therefore before view interpolation starts, each depth map has to be transferred into a locally consistent 3D model integrating as much depth information from the depth map as needed. When local models are created, new views can be generated by choosing some real views, renderig the corresponding local models and applying the images as textures.

Different approaches for the representation and creation of 3D models are described first. The process of mixing and blending multiple models is described in detail afterwards.

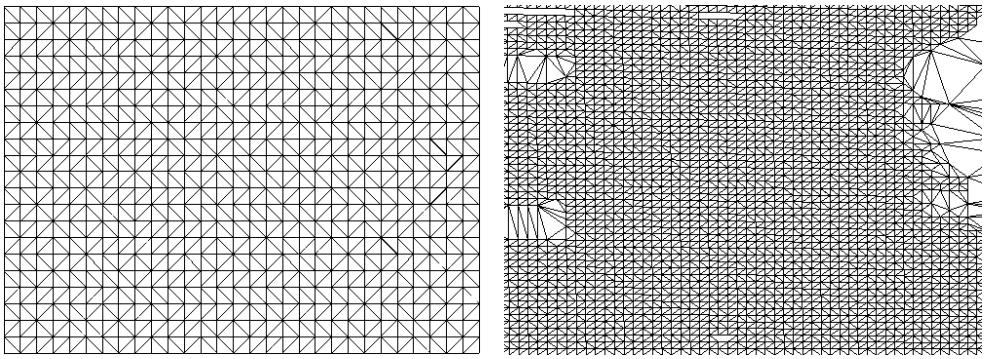
5.3.1 Subtype $P2_a$ Mesh Generation

The most obvious strategy to generate polygon models based on depth maps is to generate a 2D mesh covering the depth map and then back-projecting each vertex from 2D to 3D using the depth information.

On the assumption of completely filled depth maps, a regular grid is given by the pixel raster of the image. So every pixel can be connected to its direct neighbours resulting in a mesh of quadrilaterals. By adding one diagonal connection per quad as shown in figure 5.4(a), a triangle mesh is created. Each 2D image point can be back-projected with equation (2.20) into 3D space yielding a polygonal mesh suitable for rendering.

If the depth estimation process has failed for some image regions, the resulting depth map is not completely filled. This prohibits the creation of a regular grid as described above. It has to be decided how to handle regions without geometry information. One possibility is to accept holes in the mesh. During rendering these holes can be filled from other views.

Another possibility is to interpolate the missing geometry. Even if this interpolation does not approximate the real geometry it enables the use of texture information in these regions from the original image.



(a) (b)

Figure 5.4: A regular mesh (the diagonal connections are chosen randomly) and a partially irregular grid.

The creation of an irregular triangle mesh on a surface can be done using *Delaunay* triangulation [Shewchuk, 1996]. For a given set of points on a plane, this method ensures that triangles never intersect with each other and that the plane is completely covered. The algorithm maximises the smallest angle of all triangles. So by using all image points with estimated depth, Delaunay triangulation creates a mesh which interpolates holes in a linear manner. An example is shown in figure 5.4(b).

Problems occur at depth discontinuities (edges), because the triangulation process is not aware of the 3D geometry. Triangles will be created across object boundaries (see figure 5.5). This does not reflect the real geometry of the scene and will result in distorted images during rendering. Observing the model from the position of the real view (projecting into the associated real camera) does not

show any distortions because the vertices of such triangles will project onto adjacent screen pixels and the applied texture fits exactly. But when viewing this model from slightly different positions, triangles connecting objects are revealed. These stretched triangles have a large area in 3D space and incorporate texture information of only a few pixels.

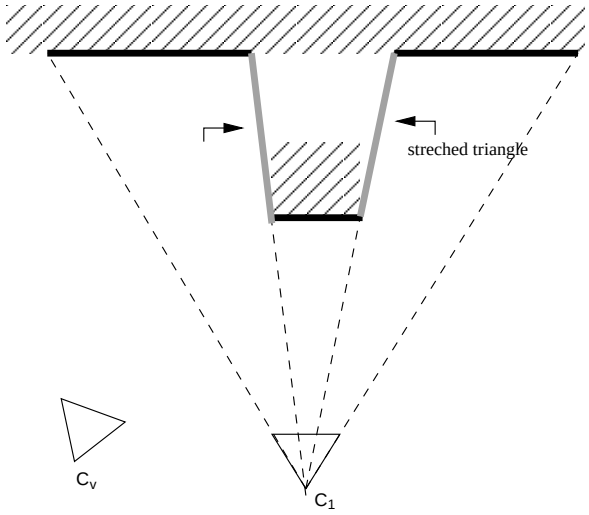


Figure 5.5: Edges crossing depth discontinuities connect foreground and background. The stretched polygons have to be discarded to avoid distortions in new views.

To avoid this particular problem, stretched triangles are removed in a processing step after mesh creation. The decision if a triangle connects foreground and background objects is based on its normal. If the normal of the triangle is nearly orthogonal to the ray from the centre of the real view to the centre of the triangle, this triangle is discarded. Discarding triangles at edges reduces distortion but also produces uncovered regions which have to be filled later.

This problem has also been addressed lately by Zitnick et al. [2004]. They divide the depth map into layers which are treated differently during rendering. One layer only contains continuous regions, while a second layer only contains the regions around edges.

Using all available image points gives a very fine granular mesh modelling all geometrically reconstructed details. This can imply some disadvantages:

- Noise from the depth estimation is not filtered.
- Even in smooth regions many triangles are used increasing the amount of vertices unnecessarily.

- If the virtual camera observes the scene from a great distance, multiple vertices project onto the same screen-space coordinates. This can result in aliasing artefacts.

Proper sub-sampling and multiple levels of detail of the depth maps can avoid these problems. Sub-sampling is a process which reduces the sample rate and requires filtering the signal beforehand to ensure that the sampling theorem has been obeyed. Otherwise artefacts due to aliasing will occur. Filtering has to be done adaptively, otherwise object boundaries are smoothed.

The major problem of the described modelling approach is that the depth map does not contain any topological information, but the representation as a closed mesh assumes a topology where all pixels are connected. Removing triangles connecting objects and background is the only modification to this topology. This observation leads to modelling approaches which do not assume a topology.

5.3.2 Subtype $P2_b$ Point-based Modelling

The usage of triangles as graphics primitives has been a technically necessity. In the early days of computer graphics, projecting vertices was very expensive and polygons allowed to generate fragments or pixels for planar surface elements using only a few vertex operations. Today's graphics hardware can compute millions of vertex operations per second which allows to use different primitives.

Points as graphics primitives have been proposed in many different publications [Pfister et al., 2000, Kobbelt and Botsch, 2004] and the authors claim that points are the more natural graphics primitives than polygons. One aspect which differentiates points from polygons is that no explicit topology is required.

The missing connections between neighbours also means that holes can appear in point-based models if not rendered properly. Point-based modelling can be interpreted as sampling a continuous 3D-surface and the rendering can be seen as a projection followed by a re-sampling to fit the sample rate of the pixel raster in screen space. For this kind of sample rate conversion the surface (the original signal) has to be reconstructed from the samples.

When polygons are rendered, generally the vertices do not project to adjacent pixels in screen-space. To determine the colour of the pixels in between projected vertices, the rasteriser processes each polygon and generates fragments between vertices. Projecting points without connectivity into the screen space requires defining a region around each point. If these regions overlap each other in screen space, no holes will remain visible.

The simplest re-sampling strategy is to use the point primitive supported by OpenGL. It allows to define coverage size on the screen after projection and all points are drawn as squares of the given coverage. This approach works well if

the point size required to avoid holes is small (1-2 pixel). For close-up views the point size has to be increased to avoid holes but then the squares become clearly visible as block artefacts. More sophisticated approaches called *splatting* can be found in Zwicker et al. [2001] and Ren et al. [2002].

Using standard OpenGL points brings about another type of problem. Because projected points are always mapped onto the nearest neighbour pixel position in screen space, a displacement of up to 0.5 pixels in x and y direction can occur. While not visible if consistent over the complete image, a varying displacement can be recognised as cracks. This problem can be addressed by using *point anti-aliasing* which allows to place points in between pixel positions.

The preparation of the point representation is simple. Starting from a depth map, points as samples of the geometry are computed by back-projecting all image points for which depth information is available. Texture coordinates and point sizes are computed during rendering. The implemented point-based rendering methods use the OpenGL point primitive.

In a slightly different approach, each point in 3D space is represented by a *billboard* or *microfacet* as presented in Yamazaki et al. [2002] (see also section 3.5). A billboard is a quad which is always parallel to the virtual camera but it is defined by four vertices instead of only one vertex for standard point primitives. A billboard can be textured and the rasteriser produces pixels as required. The drawback is that the number of vertices to be processed is four times higher than for simple points. If one generates one billboard per depth map pixel, each single depth value is represented by four vertices with three components: 12 floating point values. Due to the amount of data to be fused, this approach is not well suited for realtime rendering. In the approach described here, the point primitive as defined by OpenGL is used.

5.3.3 Subtype $P2_c$ Adaptive Modelling with Quads

Points without connectivity are well-suited for edges and regions with large variations in depth, but for planar regions this representation is very inefficient compared with textured polygons. A hybrid representation combining properties from polygons and points is best suited for automatic model generation from depth maps.

In the following, an approach is presented which models the 3D surfaces depth map adaptively. Smooth regions are modelled by large planar quadrilaterals while fine granular structures (small quadrilaterals) are used for surfaces with high curvature and around depth discontinuities. The resulting structure is a quad-tree [Finkel and Bentley, 1974] where elements are not necessarily connected with each other.

Starting with a given size s_0, the depth map is divided into tiles of the size

$s_0 \times s_0$. For each tile, a quad in 3D space is created and evaluated for its quality to approximate this part of the scene. A quad is created by back-projecting the corners of the tile using equation (2.20).

For quality evaluation, four different criteria are checked and if one criterion fails, the tile has to be *refined*. Refinement is done by subdividing a tile into four new tiles with $s_{i+1} = \frac{s_i}{2}$ recursively. The four criteria (normals, aspect ratio, orientation and regression plane) are explained in detail in the following.

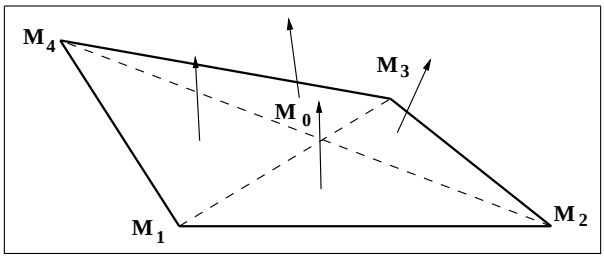


Figure 5.6: A quad does not necessarily have to be planar. Five points $\mathbf{M}_{0-4}$ are calculated for each quad resulting in 4 triangles. Evaluation is done using angles between normals, ratio of diagonals and angle between the mean of of the normals and line-of-sight.

- **Normals:** For all four corners and the centre of the tile, corresponding 3D points are calculated by back-projection with equation (2.20). These 5 points form four triangles, as shown in figure 5.6. For each triangle the plane normal is calculated and difference angles are computed between each pair of normals. If none of these difference angles exceeds a given threshold, the four corners are assumed to be in one plane. Otherwise the quad is rejected and the tile has to be refined.

- **Aspect Ratio:** The ratio of the diagonals of the quad can be used as quality indication, too. If the ratio exceeds a given threshold (typically 2.0), this means that one of the four corners does not share a depth level with any other corner. The rendered quad would be distorted, therefore it is rejected and has to be refined.

- **Orientation:** If a quad has passed both previous tests, a mean-normal is calculated from the four normals and compared to the line-of-sight from the real camera to the centre of the quad. If the angle exceeds a threshold (80°), the quad is assumed to connect foreground and background, which would result in artefacts. Therefore the quad is rejected and has to be refined.

- **Regression plane:** The use of only 5 samples to span and verify a quad can lead to aliasing if the underlying 3D structure has high frequencies. If, for

example a small object is not matched by the samples which all are located on the planar background, the resulting quad approximates the background. During rendering this will result in artefacts because the colour of the objects is applied to the background. To avoid this, the last check computes a regression plane over a smoothed dense sampling of all 3D points inside the quad and evaluates the mean distance of all these samples. If the mean distance exceeds a given threshold, the quad is rejected and has to be refined.

The recursive refinement terminates when the size of a tile s_i reaches 2. For a quad from such a small tile, only the aspect ratio and the surface normal is checked. If any of these tests fails, the quad is rejected leaving a hole in the model.

Figure 5.7 shows an image of a synthetic scene consisting of a planar floor, a nearly planar background (castle) and an occluding object (arch). The wireframe model demonstrates the tessellation. If possible, large quads are used to approximate the geometry. Only the regions with discontinuities in depth around the arch are sampled very densely.

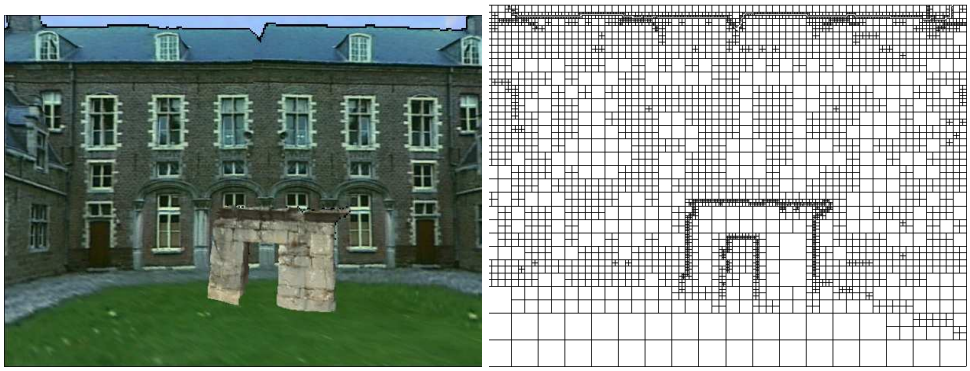


Figure 5.7: This synthetic scene demonstrates the adaptive refinement. Planar regions are sampled with large quads, while regions with high curvature or depth discontinuities are approximated with smaller quads. Overall 6382 quads are generated. Black regions indicate rejected quads.

5.3.4 Geometry Fusion and Rendering

Rendering a new view from multiple local models starts with the camera selection as described in section 5.1.1. The selected real views and associated local models are used for view generation. The selected models should be combined in such a way that

- holes in one model are filled from others,

- regions with similar geometry are blended,
- foreground objects correctly occlude background objects.

To fuse different local models using programmable graphics hardware an approach described by Verlani et al. [2006] is used. It provides depth comparison within an ε-environment and computes per pixel blending weights based on the angle-of-observation from the real views. The algorithm works in two passes and in each pass, the local models of all selected real views are projected. The first pass is used to build a reference depth map for the virtual view while during the second pass, this reference depth map is used to determin the visibility for each fragment from all projected local models. This two-pass process is explained in the following.

- In the first pass, all selected local models are rendered into the novel view the distance of the geometry to the virtual camera is increased by a small ε. This can be done easily in a vertex shader by shifting each vertex along the line-of-sight. For this purpose, the extrinsic camera parameters are used to transform **M** into the camera coordinate system. In this system, the length of the vector pointing to **M** is scaled with $1+\varepsilon$ and finally the OpenGL projection matrix is applied:

$$\mathbf{M}' = K_{\text{GL-Projection}}(1+\varepsilon)(R^T, -R^T\mathbf{C})\mathbf{M} \tag{5.8}$$

  No colour information is generated in this pass, only depth information is accumulated in the Z-buffer. The Z-Test accepts only fragments if their projective depth z_{new} is smaller than those of the fragments of previously rendered models:$z_{\text{new}} \leq z_{\text{old}}$ and updates to the Z-Buffer are activated. This scheme allows to fuse surfaces with similar depth but to handle occlusions correctly if the difference in depth exceeds ε.

- In the second pass, again all selected local models are rendered but with their original distance to the virtual camera. Updates of the z-buffer are prohibited and the z-test accepts only fragments with a depth smaller or equal to the depth in the z-buffer. Due to the shifted z-buffer content from the first pass, all fragments within a range of ε are accepted. For all accepted fragments, the colour is blended from the colour accumulated so far and the colour contribution of the new fragment. The realisations of two blending approaches are described in sections 5.3.5 and 5.3.6.

Rendering multiple models is a serial process. To allow the blending of an arbitrary number of local models with non-standard blending, a double-buffer ping-pong algorithm using a pair of frame buffer objects is used. A frame buffer object

describes an amount of memory on the graphics card which can be used as a render target or as texture source, alternatively. With two such objects B_0 and B_1, a ping-pong algorithm can be implemented. For the first step $i = 0$, frame buffer object B_0 is used as render target and the first local model is rendered with its own projective texture. For each following step i, $B_{(i \bmod 2)}$ is used as render target and $B_{(i+1 \bmod 2)}$ provides the result of the previous step. For $i = 1$ the next local model is rendered into B_1 using its own projective texture but also using the content of B_0 as non-projective texture[2] and accumulating the colour information from step zero to step one.

Finally, the content of the target buffer of the last step $i = n$, which is $B_{(n \bmod 2)}$ is copied to the visual buffer by setting up orthographic projection, using one texture unit to access the image from the buffer and rendering a simple quadrilateral.

5.3.5 Per-Pixel Blending

In section 2.5.1 a per-pixel blending approach has been discussed where the colour contributions are weighted according to the angle between the viewing ray of the real and the virtual view. In this section the technical aspects of a GPU-based implementation are described.

Due to the serialised rendering of local models the blending has to be serialised, too. This blending is done for each fragment inside the rendering of the second pass. Accumulating the colour from a previous step with the current step is done by a fragment program executed by the GPU for each fragment to process. This fragment program has to compute the colour of the current fragment and for this purpose it has to access the texture for the current model with projective image coordinates and the accumulation buffer from the previous step with the current pixel position. Two different texture units and their associated texture coordinate engines are used to pass colour data to the fragment program. The first texture unit contains the original view of the current model and the corresponding texture matrix is loaded with the appropriate matrix to generate projective texture coordinates from the fragment position. The second texture unit contains the accumulation buffer from the previous step and the texture matrix is set up to generate screen space coordinates corresponding to those of the current fragment.

To calculate the blending weight based on the angle-of-observation for the current fragment the 3D point $\mathbf{M}$ of the scene point associated with this fragment has to be known inside the shader program. This $\mathbf{M}$ is generated by the rasteriser of the GPU and cannot be accessed by the main program. To pass $\mathbf{M}$ into the fragment shader, an additional texture unit is prepared for projective texture mapping

[2] The content of a frame buffer object is accessed as a texture. In this case, the pixel coordinates of the current fragment have to be used to address the pixels in the frame buffer object.

using automatic texture coordinate generation (see section 5.1.4). By setting the associated projection matrix to identity, the fragment program can access the 3D point via this texture coordinate. The also required 3D points of the centres of the current real camera and the virtual camera are constant for the current local model and can be passed to the shader as constant parameters.

5.3.6 Per-Camera Blending

Experiments have shown, that per-pixel blending does not always gives smoothly blended transitions between real views. The parameter of the underlying blending equation has to be adapted to the special needs of the setup. The per-camera blending as described in section 2.5.2 using barycentric weights for interpolation and linear weights for extrapolation is the more robust solution. No parameter has to be adapted to the given setup. The adaptive camera selection scheme as referred to in section 5.1.3 ensures that all requirements for barycentric blending are met. The setup of texture units is the same as above, only the fragment shader responsible for blending is changed. Instead of dynamically computing weights, the shader program receives the weights as constant parameters.

5.3.7 Limitations

Constructing local models from all depth maps and fusing them during rendering is a highly efficient approach. Applying texture and blending colour values either with a per-pixel weighting or with a per-camera weighting can generate high-quality images and interpolate smooth transitions between adjacent real views. If depth information of sufficient quality is available, the underlying geometry is dense and precise. But right here is the crucial point of the system! If the depth information is incomplete, inconsistent between different views or noisy, this directly degrades the image quality.

5.4 Prototype $P3_a$ Plane-SweepRendering

The Plane-Sweep algorithm is designed to find pixel-wise correspondences between real images by scanning the volume observed by the real cameras. This scanning is done by sweeping a plane through the space, which gives the name of the algorithm.

Correspondences for every pixel of a set of real views can be exploited in different direction: depth estimation for geometrical reconstruction or colour interpolation for visual reconstruction. The remaining part of this section is structured as follows:

- first, the basic principles of the Plane-Sweep algorithm for correspondence search are introduced,
- the usage of the Plane-Sweep for depth estimation and
- the usage for view generation without geometry are described.

In the following section 5.5, it is shown how geometry information can be used to enhance the Plane-Sweep view interpolation.

5.4.1 Correspondence Search with Plane-Sweep

Many different algorithms have been proposed to solve the correspondence problem. Nearly all of them use a *matching function* to compare points or regions in images based on their intensity values. The underlying assumption here is that any point **M** projects into every image I_i with the same intensity and colour (lambertian surface). The Plane-Sweep algorithm establishes correspondences between two images by evaluating the matching function for each pixel in a given interval and finally choosing the minimum and its associated 3D point defining the correspondence.

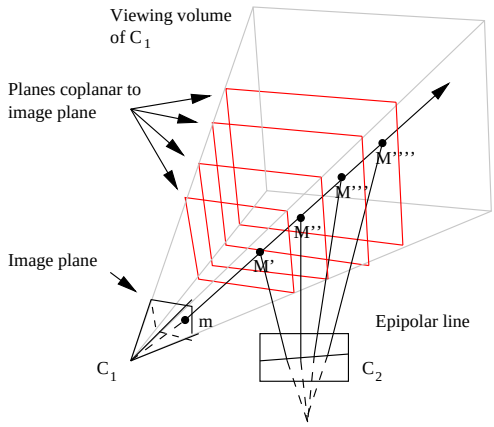


Figure 5.8: Correspondence search along epipolar line and with plane-sweep.

Evaluation and sampling of the matching function of all pixels is done by placing a plane Π_i in 3D space and projecting all image points of both views onto this plane and computing the difference between the intensity from the left and the right view. This is equivalent to back projecting a 2D $\mathbf{m}_l$ image point from one view to the 3D point $\mathbf{M}_z$ on the plane Π_i:

$$\mathbf{M}_z = \mathbf{C} + \lambda \frac{RK^{-1}\mathbf{m}_l}{||RK^{-1}\mathbf{m}_l||} \quad \text{with} \quad \mathbf{M}_z \in \Pi_i \tag{5.9}$$

Projecting $\mathbf{M}_z$ into the second view with equation (2.13) yields a potential correspondence in $\mathbf{m}_r$. For all potential correspondences the *matching costs* d_i are computed and stored together with λ, the projective distance of $\mathbf{M}_z$ from the camera. Different functions for the matching costs are presented in the next section.

The algorithm proceeds by placing a new plane Π_{i+1} with a given distance coplanar to the Π_i and again computes the matching costs. By iterating over n planes, for each pixel the matching costs d_i and the associated λ form a sample of the matching function $d(\lambda)$. The minimum $\min(d(\lambda))$ can be assumed to be the best correspondence for the given views and the evaluated planes.

The quality of the matching strongly depends on the number and positions of the planes. When dealing with discretised images, the planes have to be placed in such a way that the minimal and maximal pixel disparity is matched. Using exactly 2 real cameras for a Plane-Sweep, the spaces between planes should match the disparity steps implied by the pixel raster. For more then 2 views, the spacing should match the smallest possible disparity steps between any two views. Figure 5.8 shows a schema of the correspondence search for two views. This algorithm has been proposed for view interpolation by Yang et al. [2002].

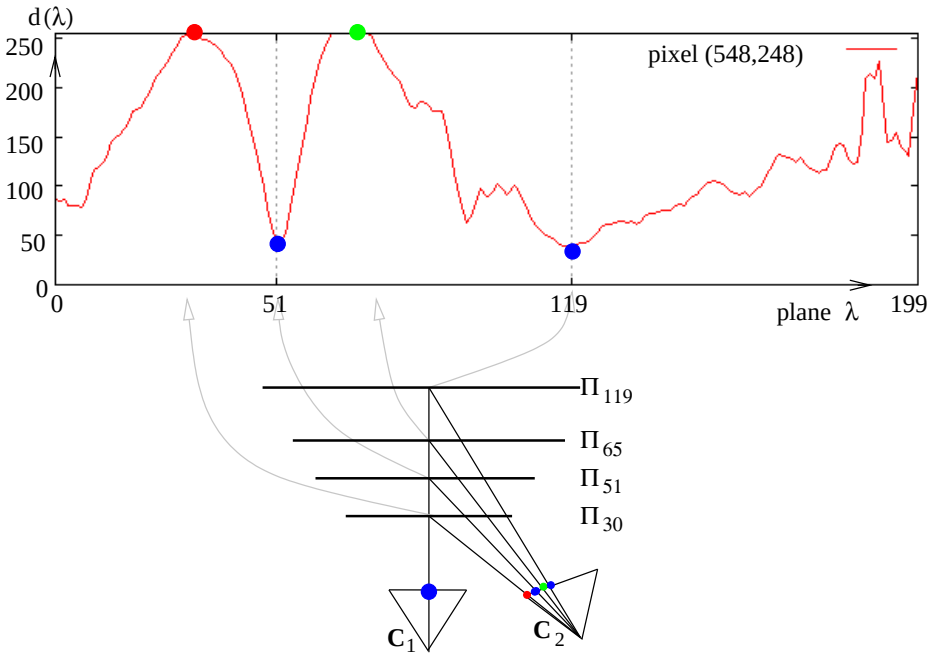


Figure 5.9: The matching function for one particular blue pixel over all planes.

The matching function $d(\lambda)$ for one blue pixel in camera C_1 over the depth λ is shown in figure 5.9. For plane Π_0, the corresponding pixel in camera C_2 is red, leading to a large dissimilarity, which is visible as a local maximum in

the function plot. The correspondence on plane Π_1, a blue pixel, gives a low dissimilarity (good match) resulting in a local minimum. Plane Π_2, is another local maximum, while the correspondence on plane Π_3 gives another minimum which is ambiguous to the match on plane Π_1.

5.4.2 Different Matching Functions

Like most algorithms for correspondence computation, the plane-sweep algorithm compares intensity values of images. This comparison can be done with different methods. The simplest comparison is the absolute difference between two values, one from each image.

$$d_{\mathrm{p}} = |I_r(\mathbf{m}_l) - I_l(\mathbf{m}_l)| \tag{5.10}$$

It can be shown that this matching function is not robust against noise in the images, variations in exposure and especially non-lambertian surface properties. More advanced matching functions take into account a region called *support window* of size $[2w+1, 2h+1]$ around $\mathbf{m}_l$ and $\mathbf{m}_r$:

$$d_{\mathrm{SAD}} = \sum_{j=-h}^{h} \sum_{i=-w}^{w} |I_r(\mathbf{m}_l + (i,j)^T) - I_l(\mathbf{m}_l + (i,j)^T)| \tag{5.11}$$

The sum of squared differences (SSD) penalises larger differences using a support window:

$$d_{\mathrm{SSD}} = \sum_{j=-h}^{h} \sum_{i=-w}^{w} (I_r(\mathbf{m}_l + (i,j)^T) - I_l(\mathbf{m}_l + (i,j)^T))^2 \tag{5.12}$$

More complex matching functions such as *Normalized Cross Correlation (NCC)* can improve the robustness but these are expensive to be computed and can currently not be used for the purpose of real-time matching.

The use of d_{p} as matching function often results in mismatches due to ambiguous local minima. An example is shown in figure 5.10(a). $\mathbf{M}_1$ and $\mathbf{M}_3$ have a similar appearance in both images which leads to a match on plane Π_2 which occludes the match in Π_1 from the point $\mathbf{M}_2$. For depth estimation every mismatch results in a depth estimation error. For view interpolation mismatches can remain invisible if the interpolated colour is similar to the correct colour.

Mismatches and local minima can be reduced by using matching functions with larger correlation windows. But correlation windows bring about problems of their own. Perspective distortion between two cameras can disturb the matching. Figure 5.10(b) shows an example in which the appearance of a region between $\mathbf{M}_1$ and $\mathbf{M}_2$ changes between C_1 and C_2 due to different perspectives. Similar effects occur at depth discontinuities.

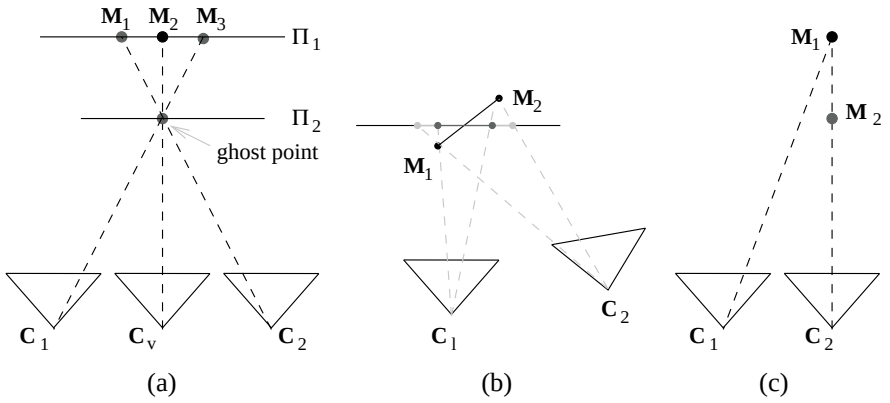


Figure 5.10: Several problems leading to mismatches: (a)Points M_1 and M_3 give a good match on Π_2 which hides the match in Π_1 from M_2. (b) The perspective distortion of the region $\mathbf{M}_1\mathbf{M}_2$ leads to different footprints in C_1 and C_2 when sweeping coplanar to C_1. (c) Point $\mathbf{M}_2$ occludes $\mathbf{M}_1$ in C_r.

Another problem results from occlusions. Points seen by a camera $\mathbf{M}_1$ and occluded in camera $\mathbf{M}_2$ cannot be matched properly (figure 5.10(c)). This can only be resolved by using additional views for matching (multi-view matching, section 5.4.7).

5.4.3 Hierarchical Matching

Computing matches using a Gaussian resolution pyramid of both images is an alternative method to use information from the surrounding region. Starting from the original image a new pyramid level is constructed by low-pass filtering and sub-sampling the image. Each evaluation of the matching is done for every pyramid level and the results are summed up. This takes a region around the potential correspondence into account without the need for complex correlation computations. The combination of a small correlation window with hierarchical matching proved to be robust against perspective distortions and helps to avoid local minima.

As proposed by Yang and Pollefeys [2003] a hierarchical matching can be done on current graphics hardware by using Mipmap textures as resolution pyramid. One problem of Mipmap textures is that those are only an approximation of low-pass filtered images. Mipmap levels are generated by averaging four adjacent pixels which is equivalent to applying a box filter. Because the box filter is not applied across the boundary between groups of four pixels, this can lead to problems.

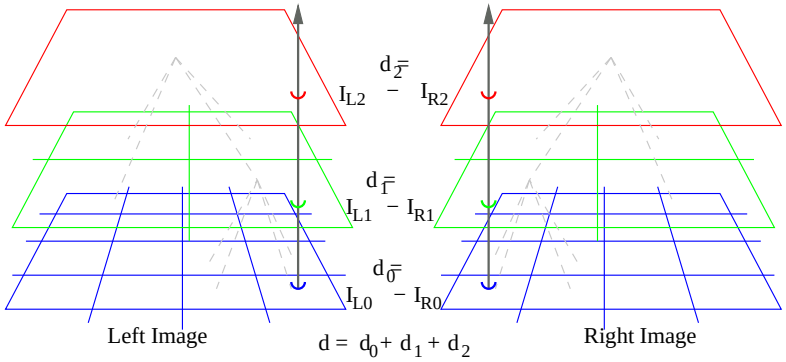


Figure 5.11: Hierarchichal matching with a gaussian resolution pyramid. The matching function is evaluated for each level and the final matching value d is the sum over all levels.

5.4.4 Depth Estimation

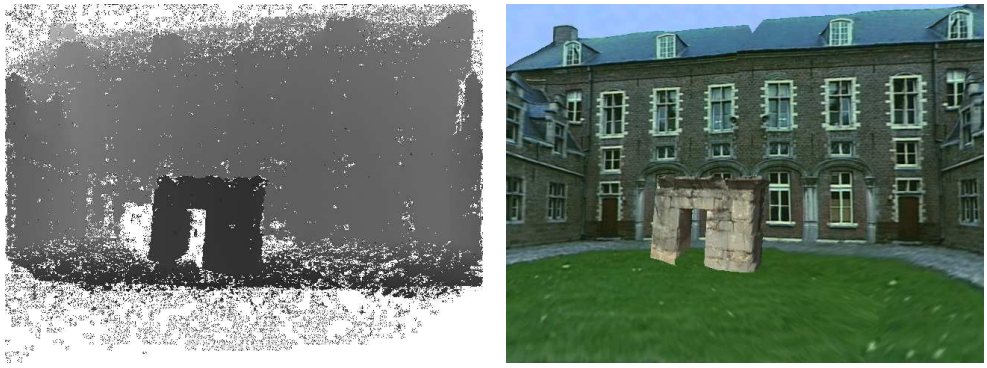
Correspondence search is the key issue for depth estimation and the Plane-Sweep algorithm, designed to find correspondences, can be used to compute depth maps.

In the context of Image Based Rendering the depth estimation process for the real images can be done as an off-line pre-processing step before view generation. This allows to use more complex matching algorithms and the number of planes is not limited by processing time constraints. The main focus here is on the completeness and reliability, not the absolute precision of the depth of each pixel. The algorithm is similar to that proposed by Yang and Pollefeys [2003] later extended by Woetzel and Koch [2004]. To compute a depth map for the real view I_r with P_r the following steps are taken:

- The virtual camera is set to match C_r.
- k neighbouring real views are chosen as support views.
- For each support view $s, 0 \leq s < k$
  - Images I_r and I_s are selected as textures.
  - Projection matrices P_r and P_s are set to generate texture coordinates.
  - For each of N planes
    - the matches between I_r and I_s are computed by rendering the plane. Use d_{SSD} with 3x3 window as matching function.
    - For each pixel: compare matching with best match so far. If the current match is better than the best match, update the best match with the current match. Remember the plane number of the best match per pixel.

  - Store 2D array U_s with plane numbers from best matches for this P_s

- Per Pixel: Process all $U_s, 0 \leq s < k$ and find the most probable depth by evaluating the distribution of best matching planes in space.



(a) (b)

Figure 5.12: Example of a depth map computed with the Plane-Sweep algorithm (a) and the corresponding original image (a) .

5.4.5 Using Graphics Hardware for Plane-Sweep

The Plane-Sweep algorithm is well suited to be implemented with the use of graphics hardware (GPU). The task of projecting images onto planes can be done by placing a quadrilateral in space and applying images from the real views as textures. With projective texture mapping, this results in homography mapping from the image plane of the real camera onto the plane defined by the quad. Instead of computing colour values for each screen pixel, today's GPUs can be programmed to compute an intensity or colour difference as matching. Multi-view matching can make use of the number of texture units available, which allows to project multiple textures in one render pass. Due to the high degree of parallelism the GPU can evaluate several hundred planes per second even with multiple real views.

The GPU implementation cannot serialise the matching over all planes and the evalution of the matching function because it is not possible to save all match images efficiently. To circumvent this problem, a *winner-takes-all* strategy is chosen. For each pixel, only the best match so far and the resulting colour is saved. The best match is initialised with a high value. Then, whenever the new value of the matching function for a particular pixel is smaller than the best match so far, the new match and the colour update the buffer. When the computation of the

matching for the last plane is completed, the buffer contains the best match for each pixel and the final colour values.

5.4.6 View Interpolation

Novel view generation with a Plane-Sweep algorithm belongs to the group of algorithms which do not use geometry information. By projecting images from several views onto a plane scanning the viewing volume, the algorithm tries to find photo consistency for each pixel to reconstruct. This implies that the reconstruction does not necessarily have to be correct in terms of geometry. For view interpolation it suffices to generate a new view which is photo consistent with the scene, the correct depth is not the primary goal. Computing best possible depth maps is a problem of its own and typically takes much computational time.

Starting with two real views C_l and C_r defined by their projection matrices P_l and P_r and associated images I_l and I_r, the goal is to generate a new view for C_v, the virtual camera. Figure 5.13 depicts this situation.

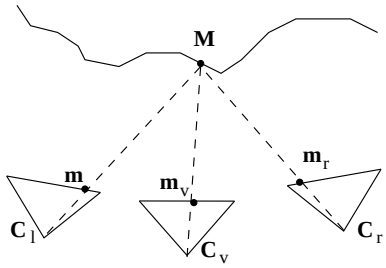


Figure 5.13: The image point $\mathbf{m}_v$ has to be interpolated from the observations $\mathbf{m}_r$ and $\mathbf{m}_l$ of the point $\mathbf{M}$ on the surface of the object.

For a pair of images and pixel-wise correspondences, the colour value for each pixel in C_v can be computed. In addition to the depth estimation process as described above, colour values have to be kept. Whenever the current matching function reaches a minimum during the sweep, the colour for this particular pixel is updated with the blended values from the real images.

5.4.7 Multi-View Plane-Sweep

The standard Plane-Sweep approach can easily be extended to compare more than two images simultaneously. When multiple views are used, a pair-wise matching is computed and interpreted as a vote for the particular plane. Having several votes for each plane, the final score for the matching function can be computed

using statistical approaches. One method can be found in Nozick et al. [2006]. It works as follows:

1. From all (remaining) votes, the mean and the variance is calculated.

2. All votes with distances to the mean greater than a given threshold are removed.

3. Step 1 and 2 are repeated until only two matches are left or no more matches are above the threshold.

4. The matching result for this plane is then the mean of the matching costs of the remaining votes.

After the best matching plane for each pixel of the virtual camera has been found, the final colour values are computed by blending the colour values from the corresponding pixels in the real views as defind by the match. Per-pixel blending as well as per-camera blending can be used.

5.5 Prototype $P3_b$ Depth Guided Plane-Sweep

The described standard plane-sweep algorithm can be used to interpolate images by accepting the best correspondence per pixel and combining the colour information from both images at the corresponding points. No geometry is required, but to ensure an appropriate reconstruction, a large number of planes has to be used to match all possible depths. Processing many more planes than absolutely necessary increases the probabilty of mismatches and visible artefacts dramatically. To compensate for this, complex matching functions have to be used which degrade the performance even more.

In this thesis it is proposed to integrate geometry information if available and show that this improves both: quality and performance of the image generation. The basic idea is to use depth maps to guide the view interpolation in order to reduce visible artefacts but not rely on high accuracy.

5.5.1 Overview

The following novel view interpolation approach uses depth information to guide a plane-sweep view generation to achieve two major goals:

1. increase the quality and precision of the matching,

2. decrease the rendering time to reach interactive frame rates.

In a pre-processing phase, for each real view a depth map is computed with the use of all available real views for matching. This is described in more detail in section 5.4.4. The scene space is sampled with planes densely and complex matching functions make sure that mismatches are avoided. The quality of these depth maps is not comparable with those computed by other depth estimation schemes, but the following view interpolation takes this into account.

The depth maps serve as starting points and search range limitation for real-time correspondence search during rendering. Each depth map is adaptively subdivided into a quad-tree based structure with regions of consistent depth similar to the approach described in section 5.3.3. Instead of sweeping a large plane, the quad-tree and the depth information are used to create tiles (partial planes) and sweep these only in small intervals.

By tiling the original plane and performing partial sweeps, the number of matching evaluations is reduced dramatically and the sampling density for each sweep can be increased at low extra costs. At the same time, mismatches due to local minima are avoided.

5.5.2 Preprocessing

After depth estimation, all depth maps are processed in order to create one quad-tree of tiles for each depth map. The algorithm works on each depth map as follows:

1. At the beginning, the maximum depth difference z_δ is determined as the difference between the smallest and the largest depth value.

2. Then the depth map is divided into regions of similar depth by a quad-tree, starting with a region size of 64x64 pixels.

3. In each region, the minimal (z_{min}) and maximal (z_{max}) depth is computed and the difference is compared with the global difference in depth z_δ:

$$z_{max} - z_{min} < \alpha z_\delta \tag{5.13}$$

   The factor α is typically chosen to be $1/10$.

4. If the above term evaluates as true, the tile is accepted, otherwise it is subdivided into 4 tiles. For each tile, the algorithm proceeds with step 3.

5. The recursive refinement stops when the size of a tile is 2x2. If even this 2x2 tile has too much variation in depth it is rejected and depth information for this region is discarded.

Figure 5.14: Example of the tiling based on the depth map in figure 5.12.

In figure 5.14 the estimated depth map and the tiling is shown. For each tile the search range in depth for the following reconstruction is determined by $[z_{\min} - z_\varepsilon, z_{\max} + z_\varepsilon]$ with z_ε a small offset to compensate for depth estimation errors (figure 5.15).

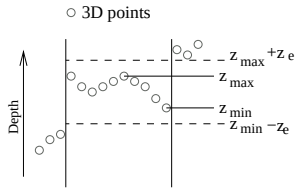


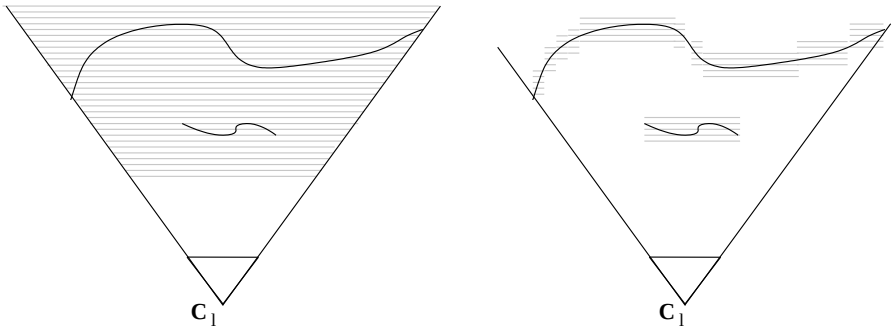
Figure 5.15: Search range in a region with similar depth.

5.5.3 View Generation

New views are generated by a piece-wise plane-sweep algorithm guided by the previous depth estimation. For this purpose, one reference view and up to $N-1$ (assuming N texture units beeing availabe) adjacent other real views are selected as support views[3].

The first step in view generation is the ranking of all real views according to their distance and angle to the new virtual view as described in section 5.1.1. Then the best ranked real view is chosen to be the reference view and up to $N-1$ next best views are chosen as support views. All selected views are set as textures for projective texture mapping.

[3]In this implementation, the maximum number of views to use depends on the number of texture units. Other implementations are possible.



(a) (b)

Figure 5.16: (a) The standard Plane-Sweep has no knowledge about the scene. (b) With the use of depth guidance, adaptive tiles can be swept to refine the regions which contain 3D structure.

Based on the tiled depth map of the reference view, for each tile a quad is constructed and placed in 3D space coplanar to the virtual camera. The distance to the virtual camera is set up $z_{\min} - z_\varepsilon$. Then these quads are successively moved towards $z_{\max} + z_\varepsilon$ in αN steps of $(z_{\max} - z_{\min} + 2z_\varepsilon)/(\alpha N)$ size.

The matching function used here is a single-pixel d_{SSD} without Mipmap. Due to the limited search range of the tile sweep, severe mismatches are avoided. The d_{SSD} can be computed efficiently on the GPU by using the *dot()* operation which computes the dot product of two vectors.

For every plane and every pixel in the new view, the matches from N views have to be merged and an intensity or colour value has to be computed. Because the matching is always computed with regard to one reference view, for N views only $N-1$ matches have to be considered. To achieve robustness against occlusions, a statistical approach as described in 5.4.7 is used to identify and remove outlier matches. The colour contribution of the remaining inlier matches is then blended according to the distance of the associated cameras to the virtual camera. This ensures that the colour changes smoothly over time when moving the virtual camera from one real view towards another one.

If the new virtual view is too different from the chosen reference view, the view generation can leaves regions unfilled. This is caused by the fact that the generated tiles are moved towards the reference view and thus can expose occluded regions.

5.5.4 Advantages of Depth Guidance

The depth guided Plane-Sweep approach has advantages compared with the standard Plane-Sweep view interpolation. In the pre-processing phase, the plane-sweep is performed for the view of the real camera which is chosen as the ref-

erence view for matching. This results in more stable matching compared with a virtual view not located at a real view. In addition, no time constraints apply and all available real views can be incorporated to fill holes and handle occlusions. A matching function d_{SSD} with 3x3 correlation window and no Mipmap has turned out to provide the best results.

In the on-line phase, interactive frame rates should be reached. The matching has to be done with a small correlation window (1x1) and the number of planes which can be evaluated is limited. The precomputed depth maps allow adaptive limiting of the final sweep and therefore the search range, without impact on the precision and matching quality. The increased depth resolution can provide better results. For example, if a given scene is properly reconstructed by 200 planes in a traditional sweep, it suffices to use 20 different levels per tile.

Local minima of the matching function can only influence the view if they occur within the search range. Limiting the search range reduces the probability of local minima considerably leading to more robust matching and less visible errors.

5.6 Summary

Based on the theoretical foundation in chapter 2, the various known approaches as presented in chapter 3 and the requirements referred to in chapter 4, three different prototypes $P1$, $P2$ and $P3$ have been developed in this chapter. These prototypes can be summarised as follows:

Prototype $P1$ (View-Dependent Geometry): A warping surface fitting screen space is approximated for each new view and the real views are transferred into the virtual camera via this geometry proxy.

Prototype $P2$ (Multiple Local Models): In a preprocessing phase, for each real view, a local geometry is modelled. During rendering, selected models are blended to form the novel view. Three different subtypes ($P2_a$-$P2_{2c}$) implementing different modelling strategies have been realised.

Prototype $P3$ (Plane-Sweep): Finding photo-consistency between real views is the basic principle of the Plane-Sweep algorithm. This has been exploited to generate novel views ($P3_a$). By adding geometry guidance ($P3_b$) the quality and the performance can be increased.

In the next chapter, all prototypes are evaluated according to four different error metrics using different data sets as input.

Chapter 6

Detailed Analysis and Discussion

In this chapter, the image reconstruction quality and the performance of the proposed algorithms from chapter 5 are analysed. At first, four error categories are introduced and appropriate measurement methods are presented. Then these error metrics are applied to the rendering methods from chapter 5 different input data sets are used, each of which has its own characteristics. The results will be discussed for each metric separately. In addition to the reconstruction quality, the required time for image generation is also relevant. Therefore the last part of this chapter discusses the run-time performance of the rendering methods with respect to the GPU.

6.1 Image Reconstruction Errors

When generating novel views from real images, the reconstructed image is not always perfect. Visible image errors can be categorised as follows:

I. Unreconstructed regions: Image regions which have not been filled from any real view. Most often, the scene parts which have to appear there have been occluded in all real views.

II. Colour deviation: Misreconstructed image regions typically result in wrong colour values. Small deviations can stay perceptionally unrecognised if the colour seems reasonable, but large deviations disturb image quality significantly.

III. Wrong depth: The generated depth map can be partially incorrect even if the colour image is correct. This remains invisible until virtual objects are inserted. Wrong depth then leads to false occlusion.

IV. Time coherence: Assembling several generated images to a motion picture sequence can expose time coherence errors. If each of the generated images contains errors from category I or II and the positions of those erroneous regions vary from frame to frame, this results in flicker artefacts, which disturb the assembled video sequence.

The proposed Image-Based Rendering methods have been analysed according to these expected errors. The methods and metrics are presented in the following.

6.2 Generating Ground Truth Data

Ground truth data in means of reference images have to be used to analyse the generated images. For all single image artefacts (cat. I - III) the difference between the generated image I and a reference image I_{ref} with an identical camera is used. This is done by fitting the virtual camera exactly to one real camera of a sequence.

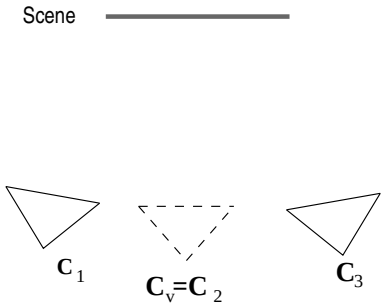


Figure 6.1: Taking C_2 as reference and generating the image from surrounding views.

Then the associated real view is taken as reference and the view generation is done without this particular real view by considering only the neighbouring views (see figure 6.1). The reference view itself must not be used for view generation. Otherwise the result would be a perfect image from trivial reconstruction. Then, for each of the categories, the difference between generated image and reference image is measured by its own metric. No ground truth data have been used for category IV. The time coherence has only been evaluated relatively by comparing the results of the different algorithms.

6.3 Error Metrics

Error measurements like the *Sum of Absolute Difference (SAD)* or *Sum of Squared Differences (SSD)* have been used to describe the distance between two images

[Evers-Senne and Koch, 2003]. However, this global metric does not allow to analyse the specific errors. For this thesis, error metrics have been developed to match the introduced error categorisation and to quantify the quality of the generated images in more detail.

Unreconstructed regions (category I) The number of remaining background pixels can be used to compute a fill rate. This is done by initialising the new image with a distinct background colour not found in the real images and then executing the view generation. Pixels not touched by the view generation can then be identified easily by their colour. For an image with N pixels overall and N_{empty} unreconstructed pixels the rate of non-filled pixels is:

$$e_{\text{nf}} = \frac{N_{empty}}{N}, 0 \leq e_{\text{nf}} \leq 1, \quad \text{(Lower values are better.)} \tag{6.1}$$

Wrong colour (category II) The most widely used metric to compare reconstructed images with reference images is the *Peak-Signal-to-Noise Ratio* (PSNR) given in decibel (dB). This is an objective metric comparing the squared errors (to emphasise larger differences) with the biggest possible error using a logarithmic function.

The PSNR can be computed by using the reconstructed regions only ($\text{PSNR}_{\text{filled}}$) and over all pixels in the image whether they are reconstructed or not. (PSNR_{all}). For this purpose, each pixel is compared with the corresponding pixel in the reference image. If the pixel has been filled during rendering, the difference is computed as mean absolute difference using the three colour channels red, green and blue:

$$\delta_{\text{filled}} = \frac{1}{3}(||R - R_{\text{ref}}|| + ||G - G_{\text{ref}}|| + ||B - B_{\text{ref}}||) \tag{6.2}$$

This allows to evaluate the filled pixels. To take unfilled pixels into account, a penalty of 127 (for intensity values from $[0, 255]$) has been taken for each unreconstructed pixel:

$$\delta_{\text{all}} = \begin{cases} 127 & \text{pixel is unreconstructed} \\ \delta_{\text{filled}} & \text{else} \end{cases} \tag{6.3}$$

For each image with N pixels, the mean squared error (MSE) is then computed by using all pixels:

$$\text{MSE}_{\text{all}} = \frac{1}{N} \sum_{i=1}^{N} \delta_{\text{all},i}^2 \tag{6.4}$$

The $\text{MSE}_{\text{filled}}$ for all filled regions is computed in a similar way but using δ_{filled} instead of δ_{all}. With the MSE_{all} from eq. (6.4), the PSNR_{all} for an image representation with b bits per colour value is given by (for $\text{PSNR}_{\text{filled}}$ respectively):

$$\text{PSNR}_{\text{all}} = 10 \log_{10} \frac{(2^b - 1)^2}{\text{MSE}_{\text{all}}} \; [dB] \quad \text{(Higher values are better.)} \tag{6.5}$$

The range of the PSNR starts by 0 for maximum deviation of all pixels and it is undefined for no deviation between images. If an 8bit-per-channel RGB image of size 640×480 only differs in one channel in one single pixel by one, the resulting PSNR is approximatly 112 dB.

The PSNR does not take the human perception into account. Metrics which incorporate human perception have been developed for various purposes. Nearly all of them are based on the so-called *Just-Noticeably-Difference (JND)*. The ΔE metric [Hunt, 2004], for example, is optimised to compare colours and colour deviations. For this purpose it exploits the JND of large homogeneously coloured regions and colours are defined in the CIE-Lab colour space. But this cannot be directly transferred to the evaluation of reconstructed images. Experiments have shown that if the Delta-E metric is applied to each pixel of an image, only a few pixels are classified as correctly reconstructed but a human observer cannot see any visible artefacts. For the evaluation of video codecs similar specialised metrics have been developed. But these metrics are optimised for motion pictures and cannot be transferred either to image reconstruction problems. The development of an adapted JND metric is a complex task which also involves empirical tests. This has not been possible in the context of this thesis. To allow a perceptional quality evaluation in addition to the PSNR, reconstructed images and difference images are printed in appendix B.

Wrong depth (category III) Even if the image generation is the major goal, depth information is required for mixed reality applications where several virtual objects are to be combined. This compositing can only be done automatically if appropriate depth is available.

The depth reconstruction can only be evaluated by using a synthetic scene because no ground truth information is available for real scenes. Subtraction of the reference depth map D_{ref} from the reconstructed depth map D shows the differences. Due to the non-linear perception of the projective depth, small differences in the foreground impose more problems than the same differences in the background, the relative depth deviation is more appropriate then the absolute difference. The mean relative difference of depth in the filled regions is used as error metric e_{depth}. Unfilled Regions are masked out to only the depth reconstruction

quality without influence of non-reconstructed regions.

$$e_{\text{depth}} = \frac{1}{N}\sum_{i=0}^{N}\frac{||D(i) - D_{\text{ref}}(i)||}{D_{\text{ref}}} \quad \text{for all} \quad D(i) > 0. \tag{6.6}$$

The range for e_{depth} starts at 0 for no depth errors and increases with the number and severity of the depth errors.

Time coherence (category IV) The error metric for category IV requires analysing an image sequence. The comparison of two consecutive frames of a sequence from a moving camera cannot be done by direct image subtraction. The image content is displaced depending on the scenes depth and the movement of the camera. For comparison this has to be compensated for (disparity estimation). Most video compression algorithms implement this (MPEG1, MPEG2, MPEG4) as a feature called *motion prediction*. A detailed description of the used MPEG video encoding scheme is given in appendix A.1.

If the only difference between two consecutive images is caused by moving image regions, the motion prediction can predict the second frame from the first frame by motion vectors for each macro block (typically 16x16 pixels) perfectly. The achieved *Inter-frame* compression ratio is high. If differences between the images cannot be predicted from a previous frame by motion, the video codec has to store the remaining difference to reconstruct the current frame. This increases the amount of data and the compression ratio is reduced. Thus, the inter-frame compression ratio (P-frames) can be used as metric for image consistency over a sequence. The function size() determines the amount of storage space used for an image and MPEG-P() denotes the compression function. The final error metric e_{time} is the average P-frame compression ratio over a sequence with N P-frame coded images:

$$e_{\text{time}} = \frac{1}{N}\sum_{i=0}^{N}\frac{\text{size}(I_i)}{\text{size}(\text{MPEG-P}(I_i))} \tag{6.7}$$

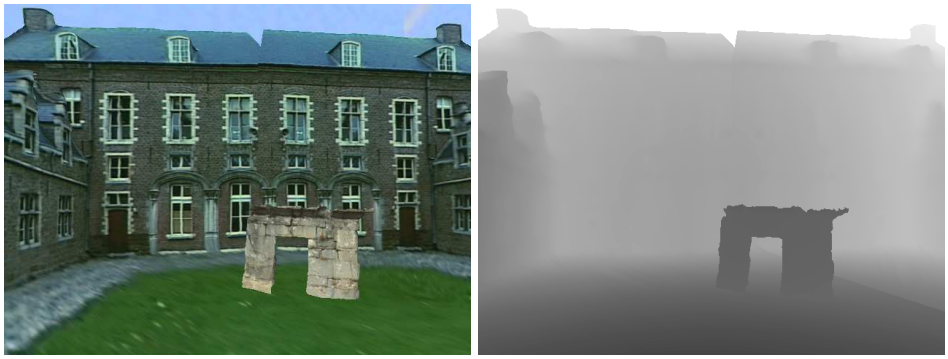
The range for e_{time} starts at 1 if no compression has been applied and with increasing compression the value of e_{time} increases, too.

6.4 Experiments

6.4.1 Sequences

All implemented methods are analysed by using three data sets with different properties. Overview images of the used data set can be found in appendix B.

Castle (fully synthetic) The castle data set is synthesised by generating screen shots and associated depths map from a textured VRML-model. 208 views in four rows are available. The calibration for each image is precisely known, the camera perfectly follows the pinhole camera model and the depths maps are completely filled without any noise or errors. This data set is used as ground truth to analyse the quality and performance of the algorithms for perfect input data. One real view and the depth map are shown in figure 6.2.



(a) (b)

Figure 6.2: One original view from the Castle data set and the corresponding depth map.

Studio (real footage, depth: good quality) The studio scene has been recorded with a broadcast-quality TV camera mounted on a dolly and moved smoothly along the scene in four different heights. The scene is well structured and the calibration (extrinsic parameters) of the camera was done with a structure-from-motion system. Depth maps are computed with a dynamic programming multi-view-stereo. Some holes occur in the depth maps but most regions are estimated with good confidence. The camera calibrations are of high quality in a local environment, but because no global bundle adjustment has been used, global inconsistencies remain. One real view and the depth map is shown in figure 6.3. Theses images from a TV camera are wide-screen views (16:9) stored in 720x576 PAL frames with a pixel aspect ratio of 1.33. So the size of the input images is 720x576 and the size of the output images is 976x576 pixels.

The Studio sequence has been contributed by BBC Research, Kingswood, UK, in the context of the MATRIS project in 2004.

City (real footage, depth: moderate quality) The city sequence has been recorded with cameras mounted on the roof of a vehicle driving in an urban environment.

(a) (b)

Figure 6.3: One original view from the Studio data set and the corresponding depth map.

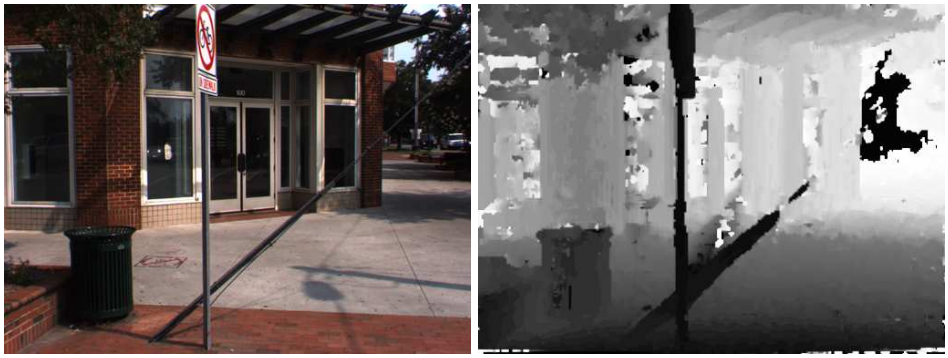
Two cameras were looking sideways. One was levelled horizontally, the second one was tilted upwards. The radial distortion has been compensated and the calibration (extrinsic parameters) was done with a structure-from-motion system. Depth maps have been calculated with the use of a multi-view Plane-Sweep algorithm with a very limited sweep range. The complete processing pipeline was designed to run in near-real-time. The characteristics of this data set is that the calibration is very good but the depth maps are quantised into 48 layers only. In addition they contain erroneous regions. The static scene assumption is violated and in some parts of the data set moving objects appear. Three tracks with overall 1612 views were available. For quality analysis only small parts have been used. The original images are 1024x768, but due to severe Bayer pattern artefacts a down-sampled version of 512x384 has been used. The supplied depth maps only have 256x192 pixel, they have been up-sampled to match the size of the images.

The City sequence has been contributed by University of North Carolina, Chapel Hill, NC, USA.

6.4.2 Setups and Data Preparation

To ensure that a broad range of scenarios is evaluated, for each sequence different setups have been analysed. For the definition of the spatial sampling density, the avarage angle α between two adjacent real views observing a point in the center of the scene (see figure 6.5) is used. These can be classified as follows:

- dense sampling of the plenoptic function, view interpolation,
  (Castle: $\alpha = 2.2^o$, Studio: $\alpha = 2^o$, City: $\alpha = 1.1^o$)



(a) (b)

Figure 6.4: One original view from the City data set and the corresponding depth map.

- sparse sampling of the plenoptic function, view interpolation (Castle: $\alpha = 11^o$, Studio: $\alpha = 4^o$),
- extra sparse sampling of the plenoptic function, view interpolation (Studio:$\alpha = 8^o$),
- view extrapolation (the virtual camera is outside the area covered by real views).

Not all combinations of data sets and setups have been analysed. The quality of the depth maps of the City data set for example is not sufficient for any other setup as dense plenoptic sampling and view interpolation. Extrapolation has only been analysed on the Castle data set and extra-sparse plenoptic sampling with view interpolation has only been analysed on the Studio scene.

Finally, all the mentioned aspects to be analysed can be arranged in four orthogonal directions: Error Metrics, Rendering Methods, data sets and setups. All

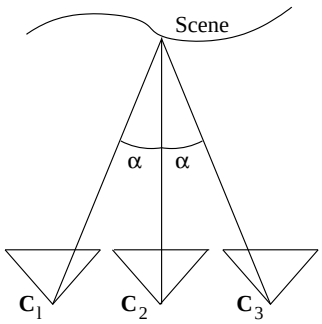


Figure 6.5: The spatial sampling density is given in relation to the average scene distance. It can be expressed by angle α.

in all, 435 images have been generated and analysed.

In the following sections I present and discuss the results grouped by the introduced error metrics. For each method, data set and setup, multiple measurements are accumulated. The plots show all analysed combinations of data sets, setups and methods on the x-axis and the accumulated mean values on the y-axis. For evaluation of the distribution, the standard deviations are plotted as error bars.

6.5 Results for Cat. I (Unreconstructed Regions)

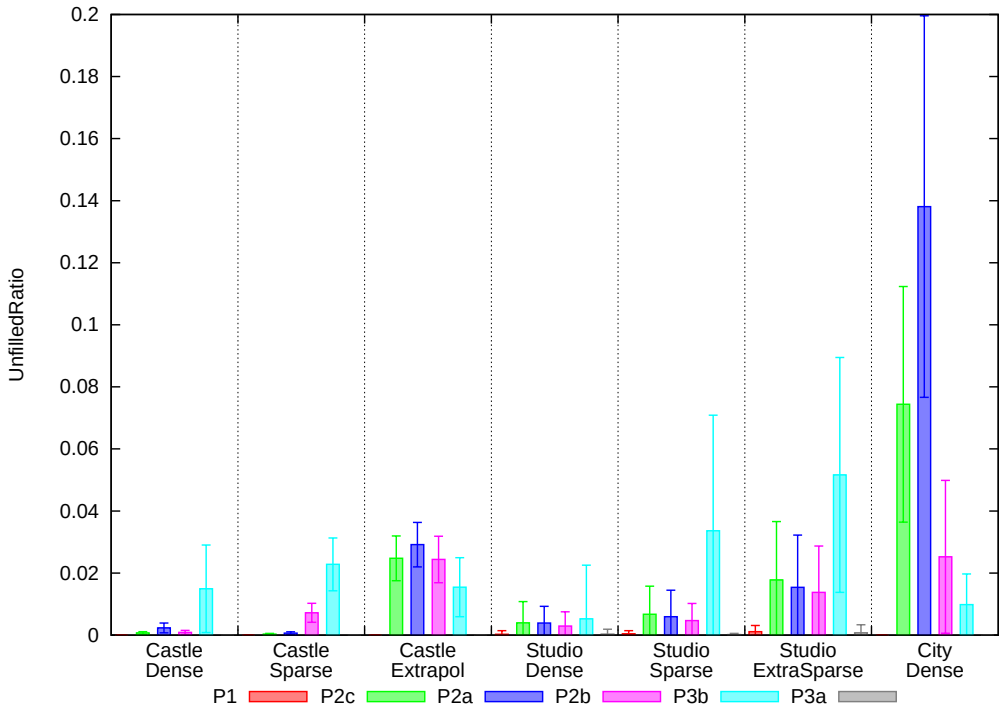


Figure 6.6: Unfilled Pixel Ratio e_{nf} for all data sets and setups

The first observation here is that prototype $P1$ and $P3_a$ produce the smallest number of unfilled pixels regardless of the quality of the input data. This is an inherent property of these algorithms. The View-Dependent Geometry ($P1$) fuses depth information from several real views in screen space and can close remaining holes with interpolation. This reduces the number of unreconstructed pixels but can increase other errors. Plane-Sweep view interpolation ($P3$) accepts the most probable colour for each pixel, even if the absolute probability for a particular pixel is very low compared with other pixels.

6.5.1 The Castle Data Set

When using ground truth data with perfect depth maps (Castle) and a dense sampling, all prototypes except the $P3_b$ achieve a rate of non-filled pixels near 0. Even with sparse sampling these results can be achieved. If we change from inter- to extrapolation, the amount of unreconstructed image regions increases. This is expected because all geometry-based methods cannot reconstruct regions which have not been seen in any real view. In figure 6.7(a) the unreconstructed regions are visualised with yellow colour. $P1$ and $P3_a$ both fill these regions with randomly chosen colours.

6.5.2 The Studio Data Set

On the Studio scene with fairly dense depth maps the Multiple Local Models-based protoypes $P2_{a-c}$ give nearly identical results. When we reduce the plenoptic sampling density from dense via sparse to extra-sparse, all three produce an increased amount of unfilled regions.

In comparison, $P3_b$ produces much more unfilled pixels when the plenoptic sampling density is decreased. Starting with a ratio below 0.01 (dense sampling) this increases to 0.03 (sparse) and to 0.05 (extra sparse). This is caused by the fact that the geometry-based sweep is optimal for the next best real camera and with increasing distance between the virtual camera and the next best real camera the reconstruction quality decreases[1]. In addition, to reconstruct the colour of a pixel, at least two real views have to agree about the colour. For sparse sampling, the number of pixels only seen by one camera is significantly higher than for dense sampling.

6.5.3 The City Data Set

The City data set with its unreliable depth maps reveals differences between the algorithms. All prototypes $P2_{a-c}$ can fuse depth information from several cameras, regions seen by only one camera can be reconstructed. But remaining holes are not filled by interpolation. This becomes visible when we proceed to extrapolation, very sparse sampling or noisy depth maps. Whereever depth information is missing or discarded as unreliable, unfilled regions occur (See figure 6.7(b) for example). On the City data set differences between the $P2$ prototypes can be shown. $P2_b$, using points as primitives, reaches a fill-rate near 100% because every depth information available is used, nothing is discarded. The modelling based on triangles ($P2_a$) and quadrilaterals ($P2_c$) does some depth analysis and rejects noisy

[1] The local geometry approximation is swept in viewing direction of the associated real camera.

(a) (b)

Figure 6.7: Yellow colour indicates unfilled regions. For the castle scene, these result from extrapolation with Multiple Local Models methods. Unfilled regions using the City data-set result from holes in the depth maps.

regions. The adaptive quad generation performs slightly better than the surface reconstruction with the triangle mesh.

Both prototypes $P3_a$ and $P3_b$ can handle improper depth information by design, if the plenoptic sampling density is sufficient. On the City data set they produce significantly fewer unreconstructed pixels than the $P2$.

6.6 Results for Cat. II (Wrong Colour)

The most dominant errors in view generation are wrong coloured pixels. This happens if, for example, incorrect depth or disparity information is used. Colour errors range from small and invisible deviations to severe ghosting artefacts where scene parts appear at wrong image positions. The evaluation of this class of errors is done with the Peak-Signal-to-Noise-Ratio (PSNR) as explained in section 6.3. Two different measurements are presented here. Figure 6.8 shows the resulting PSNR for all data sets, setups and methods evaluating only reconstructed pixels unreconstructed regions are not taken into account.

For some combinations like extrapolation or problematic input depth maps, it does not suffice to evaluate the colour deviation in reconstructed regions because unreconstructed regions disturb the visual quality. For each unreconstructed pixel a penalty of $\delta = 127$ is used to compute the PSNR as shown in figure 6.9. This adapted metric favour algorithms which can fill unreconstructed pixels with reasonable colour values over those algorithms leaving gaps.

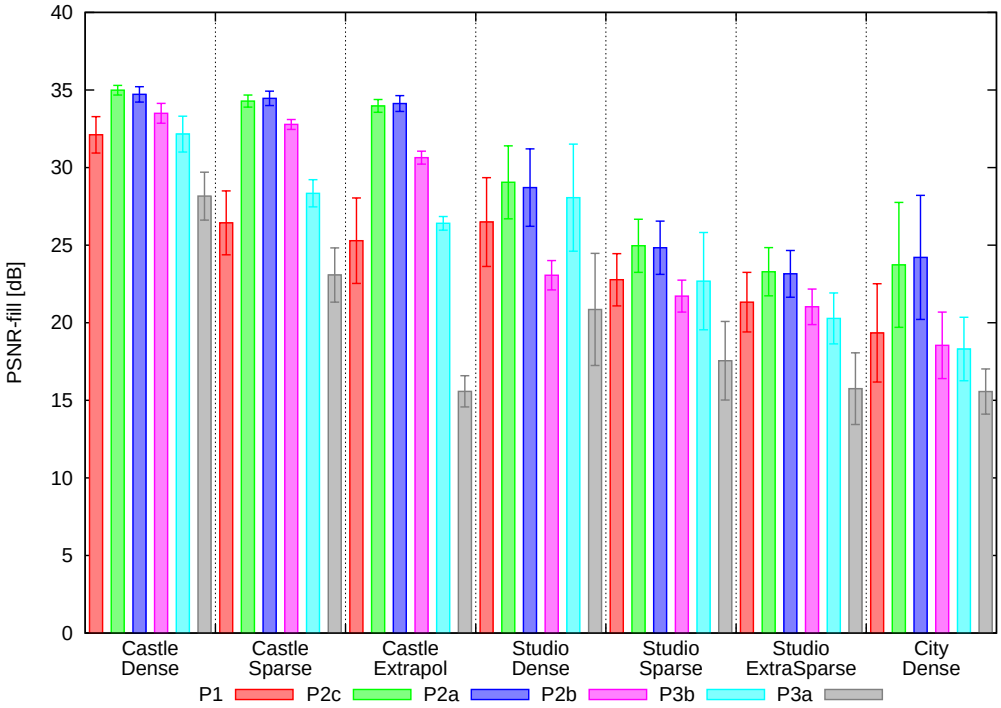


Figure 6.8: Peak-Signal-to-Noise Ratio $PSNR_{filled}$ for all data sets and setups taking only filled regions into account.

6.6.1 The Castle Data Set

Prototype $P2$ Having perfect depth, $P2_c$ and $P2_a$ produce the best results of approximately 34 dB for dense, sparse and extrapolation when evaluating only the filled pixels. The current implementation of point based modelling $P2_b$ has problems reproducing the texture. Adding penalty for unfilled pixels, the PSNR for the extrapolation setup decreases to approximately 21 dB.

Protoype $P1$ The View-Dependent Geometry can handle extrapolation setups better. It produces images with an average PSNR of 25 dB. One problem of $P1$ is revealed when we interpolate from sparse sampled setups. The texture blending produces ghosting artefacts around depth discontinuities. This problem has been explained in section 5.2.4. Figure 6.10 shows an example from the Castle data set. However, using reasonable dense-sampled data sets, ghosting is not a problem.

Prototype $P3$ The Plane-Sweep-based approaches show very divergent results on the castle data setup. The first thing to mention is that $P3_a$ cannot cope with extrapolation setups, it only reaches 15 dB PSNR. Adding geometry guidance

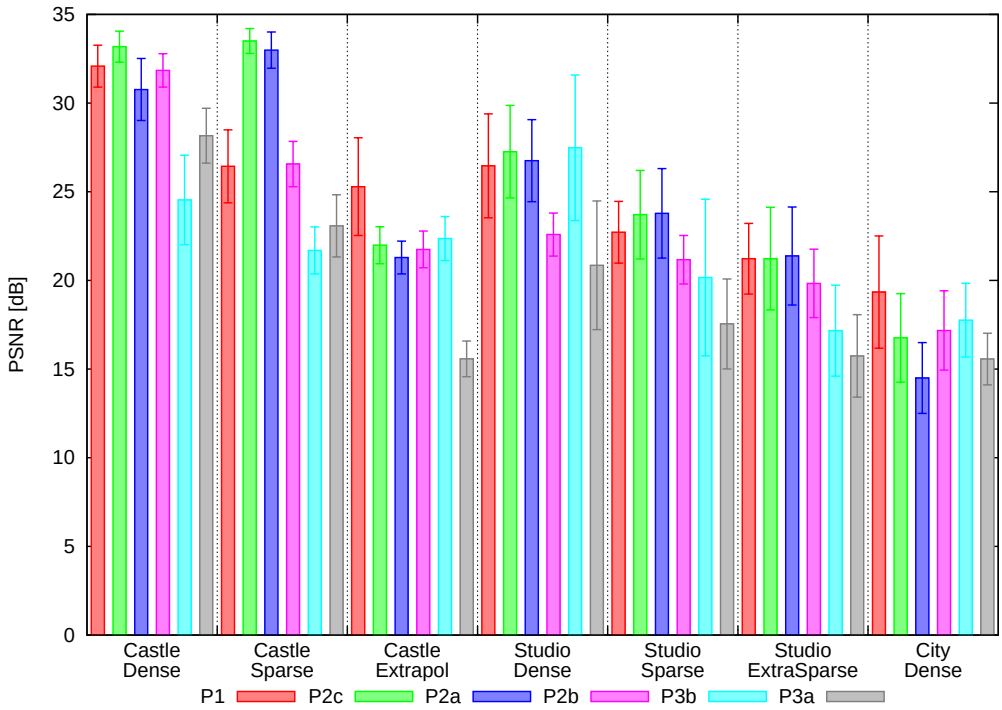


Figure 6.9: PSNR_{all} for all data sets and setups including unfilled regions.

($P3_b$) helps to reach 22 dB including unfilled pixel penalty which is even better than the results for prototype $P2$. For interpolation scenarios, $P3_b$ suffers from unfilled regions. Counting only filled pixels it produces a good PSNR of 32 dB (dense) and 28 dB (sparse), but adding penalty for the unfilled pixels, only 24 dB (dense) and 21 dB (sparse) can be reached. The algorithm is surpassed by all others.

6.6.2 The Studio Data Set

On the studio data set with real depth maps with some error, the results are different. Using a dense setup, $P1$, $P2_c$, $P2_a$ and the $P3_b$ all give comparable results. Without counting unfilled pixels the Multiple Local Models methods ($P2_c$, $P2_a$) are slightly better (28-29 dB), but with all pixels the mentioned algorithms produce results of about 27 dB. If we decrease the sampling density to sparse and extra-sparse setups, the PSNR for $P2_a$ and $P2_c$ decreases to 24 dB and 22 dB respectively. $P3_b$, the geometry-guided Plane-Sweep, provides more problems with sparse data, it can only achieve a PSNR of 20 dB (sparse) and 17 dB (extra-sparse).

The current implementation of prototype $P2_b$ (point-based modelling) brings

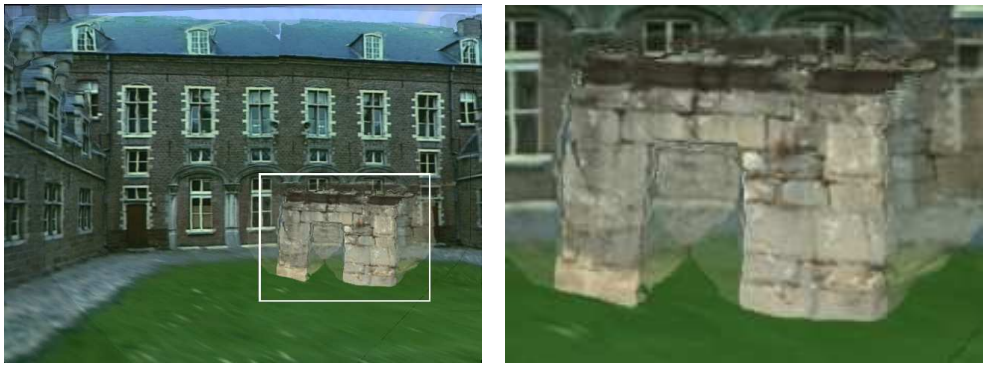
about severe problems with the Studio scene. Using simple OpenGL points with a size of 1 pixel leaves gaps so that the point size has to be increased to 2 pixel. But this results in small block artefacts reducing the image quality. Even for dense sampling a PSNR of only 22 dB is reached and this degrades to 21 dB for sparse sampling and 19.8 dB for the extra-sparse setup. An alternative rendering approach using point-sprites does not solve the problem but introduces even more gaps in the images. Anti-aliased points as defined by OpenGL cannot be used with the proposed blending algorithm because both techniques interfere with the use of the alpha buffer.

Prototype $P3_a$, the standard Plane-Sweep, produces some mismatches in the background leading to a PSNR of 20 dB (dense), 17 dB (sparse) and 15 dB (extra-sparse). The visual quality as demonstrated in figure B.4(k) is not as bad as the numbers suggest.

6.6.3 The City Data Set

The biggest challenge for all prototypes is the City data set.

Prototype $P1$ Due to the noisy depth maps, View-Dependent Geometry produces spikes in the fused geometry from small regions or pixels with a too small distance to the camera. When texture is applied, this results in visual artefacts and distortions as shown in figure B.6(a).



(a) Ghosting artefacts (b) Close-up

Figure 6.10: View-Dependent Geometry based on a sparse sampling applies false textures to the geometry. The regions around the foreground object receive inappropriate colour information leading to ghosting artefacts.

Prototype $P2$ The Multiple Local Models methods can produce reasonable image quality in regions where depth information is available, but the quality of the depth maps of the city sequence violates the prerequisites. The $\text{PSNR}_{\text{fill}}$ of 24 dB in contrast to the overall PSNR of only 16 dB ($P2_c$) and 14 dB ($P2_a$) shows this. Point-based modelling ($P2_b$) does not verify or reject the depth information and the resulting PSNR of 17.2 dB is very close to the $\text{PSNR}_{\text{fill}}$ of 18.5 dB.

Prototype $P3$ Both Plane-Sweep algorithms provide problems with the plenoptic sampling density of the scene leading to mismatches. The numerical result for $P3_b$ (geometry guided) is 18 dB, roughly. This suggests a better image quality than the result of $P3_a$ (no geometry) with only 15 dB. But when we compare the images, the visual difference is small. What can be observed is that at some positions the reconstruction is far better than at others.

6.7 Results for Cat. III (Wrong Depth)

Because for the data sets Studio and City no ground truth geometry is available, these cannot be used for evaluation. Only a comparison between the depth maps used as input data and the resulting depth from the rendering could be done. This would only reveal that the algorithms do not corrupt the information they receive. Due to the fusion of different view, holes in the original depth map can be filled, potentially. This fill-rate has already been covered by e_{nf} in section 6.5 and is not incorporated into e_{depth}.

Prototype $P2$ Having precise depth information as input data, all Multiple Local Models approaches can assemble geometry from several views and reproduce the original depth map with high quality. As for the image generation, the resulting quality mainly depends on the quality of the input data. If depth information is disturbed or incomplete, this not only reduces the image quality but also the interpolated geometry in comparison with the real geometry.

Prototype $P1$ The View-Dependent Geometry approach shows nearly the same behaviour as prototype $P2$ with one exception. In extrapolation mode the mean relative depth error e_{depth} increases significantly. When we extrapolate a scene with occlusion, parts of the scene become visible which have not been seen by any real camera. The prototype $P2$ has to leave these regions unfilled increasing the e_{nf} without impact on e_{depth}. But $P1$ closes all unseen regions by interpolation. This reduces the e_{nf} but directly increases e_{depth}. Comparing the figures 6.6 and 6.11 for the “Castle Extrapolation” reveals this coherence.

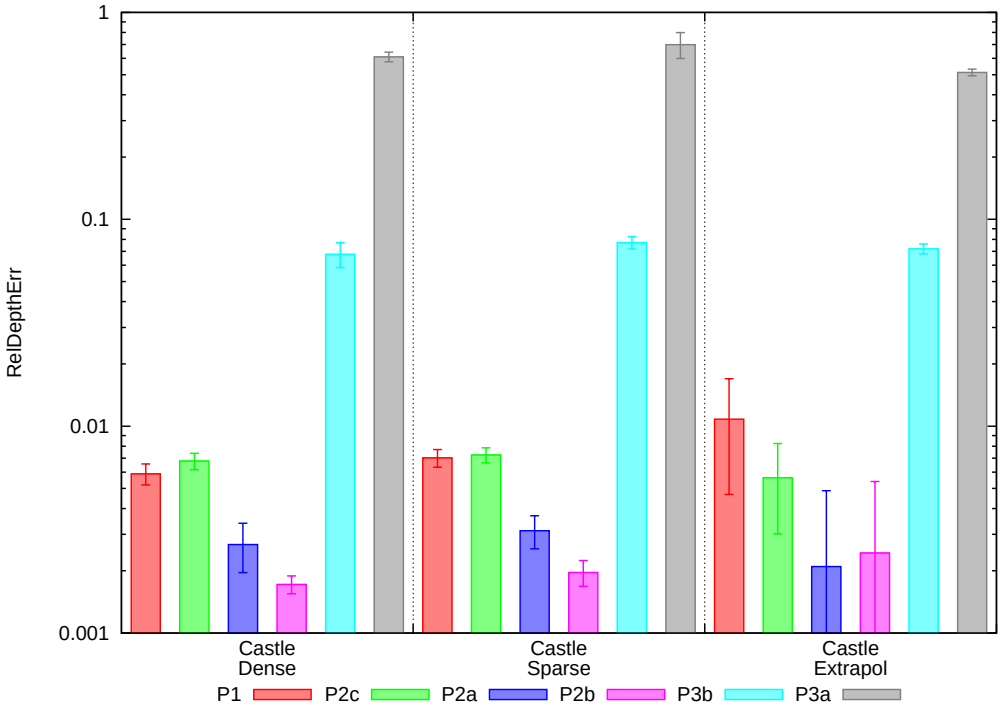
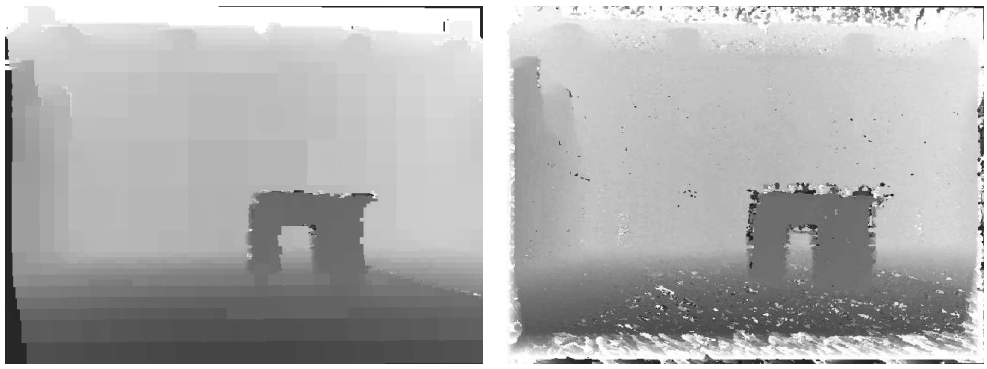


Figure 6.11: The mean relative depth error for the Castle data set. A relative depth error of 0 means no deviation at any pixel.

Prototyp $P3_a$ Both Plane-Sweep rendering methods aim at photo-consistency and therefore differ in their depth reproduction quality from $P1$ and $P2$. The plane-sweep approach without geometry guidance ($P3_a$) obviously is not influenced by the quality of the input depth data. For data sets with precisely known or estimated depth this can be seen as a major drawback. Searching for the most probable depth for each pixel to achieve colour consistency between all active real views the algorithm is a depth estimator itself. Prototype $P3$ reveals its power when dealing with data sets without any depth information or if depth information is unreliable as for the City data set. It cannot only generate colour images but also a rough depth estimation. Figure 6.12(b) shows an example for the castle scene. The quality is not comparable to more sophisticated depth estimation algorithms but it can suffice for compositing at least in a pre-visualisation context. This depth estimation comes for free, no extra CPU costs have to be spent and if we use a more complex fragment shader as matching functions the quality can be increased.

Prototype $P3_b$ The plane-sweep with geometry guidance is a mixture between the $P2$ and the $P3_a$. Using available depth information and trying to achieve



(a) (b)

Figure 6.12: (a): In the depth maps from geometry-guided Plane-Sweep the tiling becomes visible. (b): Without depth guidance, the Plane-Sweep produces many mismatches in homogeneous regions.

photo-consistency it has the potential to combine pre-computed depth with an on-line refinement. In the current implementation it does this in a very simple way. The depth reconstruction in all regions covered by approximate geometry (quads) is the front-most position of each primitive. The per-pixel decision used to find the best matching colour does not influence the depth. Image 6.12(a) shows these planar regions. Compared with figure 6.12(b) the result is much smoother and large depth estimation errors are avoided. But also subtle variations in depth are not reconstructed.

In regions not covered by the input depth map the algorithm of the pure plane-sweep is used. This can be seen in figure 6.12(a) around the arc where a per-pixel depth decision creates the same artefacts as visible in figure 6.12(b).

6.8 Results for Cat. IV (Time Coherence)

The similarity of adjacent frames in a reconstructed dense sequence is measured by the compression factor of P-frames emitted by a MPEG-1 video coder. The higher the compression factor, the more similar are the frames. Evaluation has been done only on the Castle and the Studio sequence. The quality of the reconstruction for the City data set varies too much so that it does not make sense to measure time coherence. The first impression from figure 6.13 is that on the Castle data set, prototype $P2_b$ performs best, while on the Studio data set its performance is the worst by far. The main reason for this phenomenon is that all errors in estimated depth are converted into pixels of wrong depth. When we move the virtual camera, these pixels are not static relative to their neighbours.

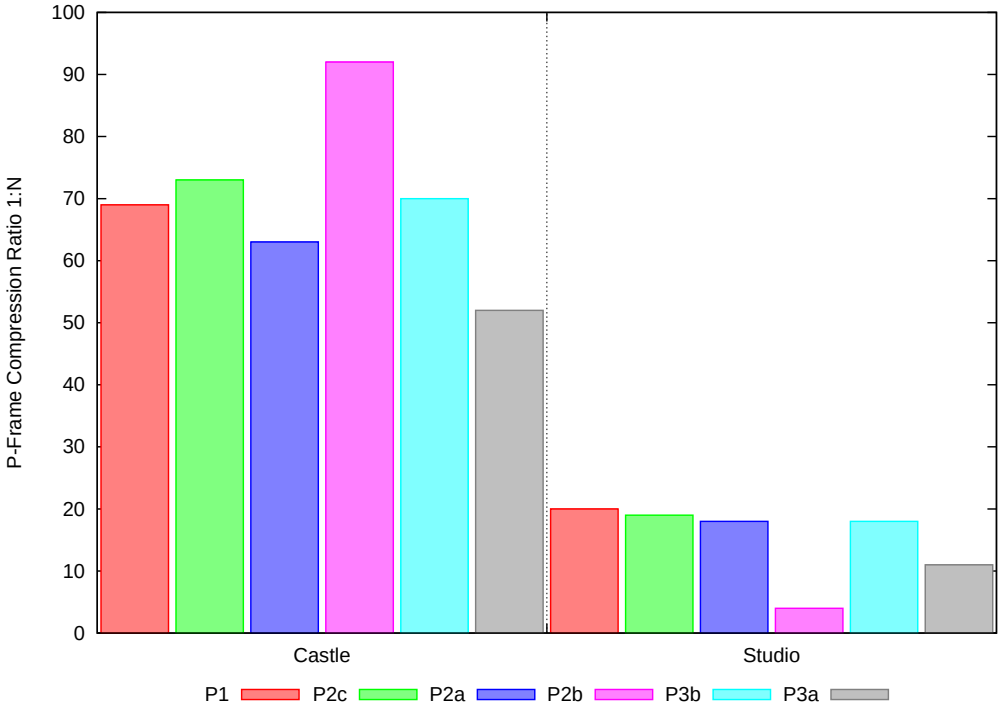


Figure 6.13: Compression ratio of P-frames

These independently moving objects reduce the time coherence dramatically.

All other methods show a similar distribution on both data sets. Prototyp $P2_c$ (quad-based) produces the best time coherence followed by $P1$ and $P3_b$ which both are comparable. The geometry approximation of adaptive quads smoothes out small depth variations similar to the adaptive quad-tree of the geometry Plane-Sweep and the View-Dependent Geometry. Prototype $P2_a$ (triangle mesh) uses a higher non-adaptive sampling of the depth maps resulting in less smoothing.

6.9 Results: Performance

The run-time performance of the proposed algorithms has been measured as the time required to render one image. This has been evaluated on the castle and the Studio data set with an output image size of 640x480 and 976x576 respectively. 50 new images have been generated per cycle for the Castle and 42 for the Studio. The Plane-Sweep based prototypes $P3_a$ and $P3_b$ are configured to use 200 planes, and 20 different distances for regions with approximate geometry. Prototype $P2$ and $P1$ use 3 active real views while $P3$ uses 4 views.

Because most algorithms make use of the graphics hardware, five different

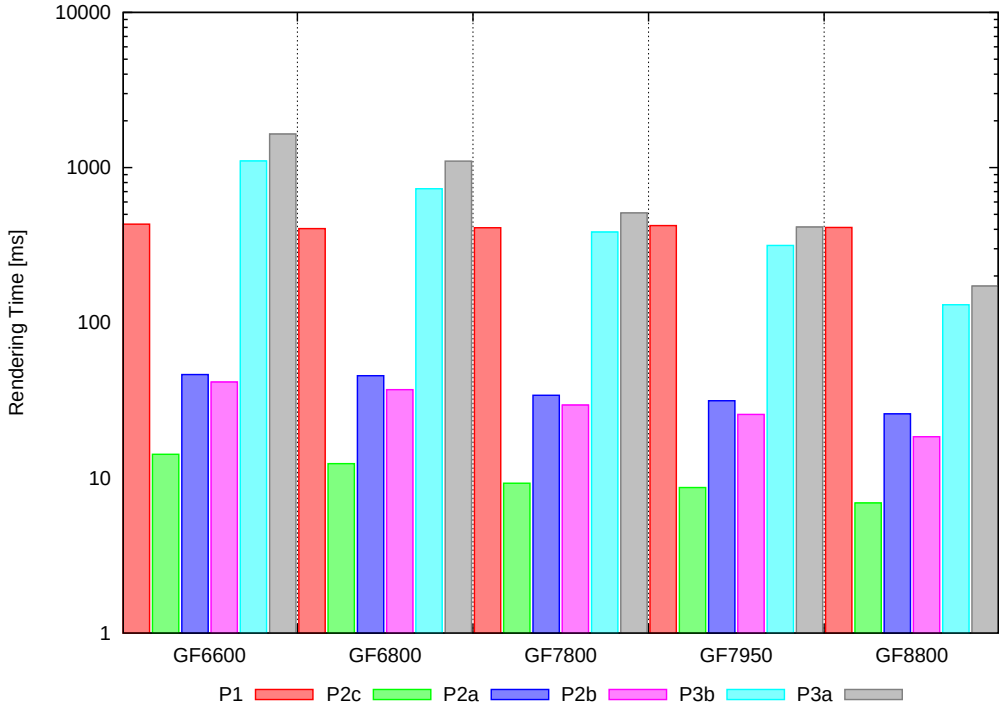


Figure 6.14: Rendering times of all methods on different graphics cards using the Studio data set. The plot for the Castle data set is nearly identical.

graphics cards with different GPUs from three generations have been used. All runs were timed on a 4-way PC workstation with 2 Intel Xeon 5140 dual-core CPUs running 2.33 GHz and 3 GB of main memory. The implementations are all single-threaded and do not make use of more than one CPU core. Care has been taken to load all relevant data into memory so that disk I/O does not influence measurements.

Each method has to process a sequence of 6 cycles, one warm-up cycle which has not been timed to load all data into memory and then 5 cycles with each one timed and logged. So for each method 250 (Castle) and 210 (Studio) values have been recorded. The average time required to generate an image has been calculated and plotted in figure 6.14. But before, outliers induced by operating system activity have been removed manually. For detailed reference, the absolute numbers are given in table 6.1.

Prototype *P*1 The View-Dependent Geometry uses the GPU only for small parts of the process, the main load is computed on the CPU. Across all GPUs, the rendering time is constant. The implementation has no specific optimisations

Dataset Setup	$P1$	$P2_c$	$P2_a$	$P2_b$	$P3_b$	$P3_a$
X300-Castle-Dense	482.2	32.7	261.7	180.6	6568.8	8490.5
X300-Studio-Dense	634.1	95.0	334.3	238.7	6638.3	8506.9
GF6600-Castle-Dense	324.5	6.1	36.4	32.9	1087.5	1645.6
GF6600-Studio-Dense	432.0	14.2	46.3	41.5	1103.2	1645.5
GF6800-Castle-Dense	313.6	4.8	36.5	29.6	721.0	1107.1
GF6800-Studio-Dense	404.2	12.4	45.6	37.0	730.7	1102.0
GF7800-Castle-Dense	326.2	4.0	27.7	23.9	380.9	510.8
GF7800-Studio-Dense	409.8	9.2	34.1	29.5	383.7	510.2
GF7950-Castle-Dense	303.1	3.8	25.5	21.4	313.6	415.4
GF7950-Studio-Dense	422.6	8.7	31.4	25.7	314.6	414.1
GF8800-Castle-Dense	309.3	2.4	43.8	15.2	130.2	173.0
GF8800-Studio-Dense	411.2	6.9	25.9	18.4	130.5	172.3

Table 6.1: rendering time in [ms] for the different algorithms and different GPUs.

and has been compiled with standard optimisation flags for the particular platform. A rendering time of approximatly 300 ms (Castle) and 410 ms (Studio) provides a frame-rate of 2-3 fps, which can be seen as a lower boundary for interactive response.

Prototype $P2$ The Multiple Local Models methods have the lowest rendering times in this comparison, by far. Most of the relevant computations are done by the GPU and the performance is limited by the vertex processing mainly. The more vertices to transform, the slower is the rendering. $P2_c$ (adaptive quads) has the fewest number of vertices, which results in rendering times of 2.4-6.1ms (Castle) and 6.9-14.2ms (Studio). $P2_b$ (point-based) is the next best, it has 2-3 times more vertices to render than $P2_c$ reaching approximatly 15-30ms. $P2_a$ (triangle mesh) is not adaptive. This results in even more vertex processing than for $P2_b$ because each vertex belonging to N triangles has to be processed N times resulting in rendering times of 25.9-36ms.

The fragment processing time needed for blending depends on the number of fragments to be processed. This is constant over all three Multiple Local Models approaches and linear to the number of real images and the output image size. This can be observed for the quad-based Multiple Local Models method when the Castle data set with an image size of 640x480 is compared with the Studio data set using images (input and output) of 976x576 pixels. Without blending as described in section 5.3.6 the rendering is approximately two times faster.

Prototype $P3$ The Plane-Sweep methods are both GPU-limited due to their expensive fragment shader programs and texture look-ups. The number of vertices N_v for the standard Plane-Sweep is linear to the number of planes N_p: $N_v = 12N_p + 4$. 200 planes have been evaluated for both data sets giving 2404 vertices to be used for each new view. The vertex count to be processed for the adaptive quad-tree depends on the input depth maps and the number of depth levels used. In both cases tiles without stable depth information used 200 levels, stable tiles used only 1/10 of that, namely 20 different depth levels. This results in approximatly 600,000 vertices to render for the Castle data set and 1,100,000 vertices for the Studio data set. So even if the overall number of vertices to process is higher for $P3_b$ than for $P3_a$, the geometry guidance is faster when rendering. This demonstrates that the bottleneck of both methods is the fragment processing, not the vertex processing. If we compare the resulting rendering times with the fill-rate and the memory bandwidth of the GPUs as given in table A.1, it is obvious that nearly all computation time is spent in the fragment processing.

The implementation of the Plane-Sweep algorithm uses two frame-buffer objects to render to and read from. The size of these intermediate images determines the number of fragments to be processed as well as the number of planes. The given rendering times have been achieved using intermediate images of size 1024^2. This has been chosen according to the input and output image size to avoid sub-sampling. When we use intermediate images of size 512^2, the rendering time is exactly 1/4 of the above[2]. It has to be mentioned that especially the timing results for the Plane-Sweep algorithms strongly depend on the GPU performance.

The results also show what performance boost each new GPU generation can give. While different GPUs of one generation only vary slightly (around 20 % speed-up), the next generation of GPU typically achieves a speed-up of factor 2-3.

6.10 Summary

In this chapter, the prototypes developed in chapter 5 have been analysed in great detail.

In the beginning, the expected reconstruction errors have been identified and appropriate error metrics have been introduced:

- Unreconstructed regions (ratio of not filled pixels),
- wrong colour (Peak-Signal-to-Noise Ratio),
- wrong depth (relative depth error),

[2]The size of the intermediate images is limited to 2^nx2^m for technical reasons.

- time coherence (MPEG compression ration).

Three data sets with different properties have been used as input data: Castle (perfect depth maps), Studio (good quality depth maps), and City (moderate quality depth maps).

Different setups of real views have been defined: Interpolation with dense, sparse and extra sparse sampling as well as extrapolation.

The following discussion of the results has been structured according to the error metrics. Finally, the rendering times of all prototypes have been analysed depending of the graphics processing unit used.

Chapter 7

Summary

After the analysis of all presented prototypes in chapter 6, it will be concluded which algorithm can be used for what purpose in this chapter. General conclusions followed by an outlook for possible future research are given.

7.1 Final Discussion of the Prototypes

If the input depth maps are sufficiently dense and reliable, the rendering systems using Multiple Local Models ($P2$) are best suited for interactive view generation. Rendering performance as measured in frames-per-second is very good, the blending can adapt to the special needs of interpolation and extrapolation, even very sparse spatial sampling (large distance between real views) allows reasonable reconstructions. If depth maps have holes, these may be closed during rendering by contribution from other views. Foreground objects are not connected to the background or other objects, so distorted texture on stretched polygons cannot appear. Only regions which have never been seen by any real camera can produce unreconstructed regions in the generated view. Depth reconstruction fuses the available structure neatly so that these systems serve well for occlusion computation in mixed reality applications.

The main advantage of the View-Dependent Geometry ($P1$) system is that it avoids holes in the reconstruction. But when occluded scene parts are revealed, distorted textured triangles are generated instead of black regions. Compared with the Multiple Local Models, the geometry fusion is very expensive because it is done vertex by vertex on the CPU. To achieve near-interactive performance, the depth maps have to be sub-sampled carefully to avoid aliasing at depth discontinuities. When the vertices are fused, the spacing of the grid in screen space has to be adapted to the sub-sampling to ensure proper fusion. When the geometry proxy is textured, it has to be decided which texture to use on a per-triangle base.

Otherwise parts of the scene receive inappropriate texture.

If depth is not available or unreliable and incomplete, the best suited system is obviously the Plane-Sweep ($P3$). It can work without depth, it can generate approximate depth maps quickly and it can incorporate depth to enhance quality and speed. The rendering quality can be influenced by the amount of time spent for correspondence search or by the quality of the geometrical proxy used. A trade-off between quality and speed can supply fast interactive preview or high-quality image generation. The Plane-Sweep is accelerated by GPU-updates more than all other proposed systems . During rendering the implicit depth estimation results in a rough approximation of the scene geometry. This geometry proxy is most often too rough and unreliable for occlusion computation in high-quality mixed reality applications. If the computation time is compared with that of multi-view-stereo systems, it can be used for interactive pre-visualisation.

7.2 Finding the Right Method

The final question for the best Image-Based Rendering method cannot be answered easily since none of the presented approaches can serve all needs. Many constraints influence the choice and in the following, some guidelines are given to find the best suited algorithm for each purpose. First, the diagram in figure 7.1 helps to find the method according to the input data.

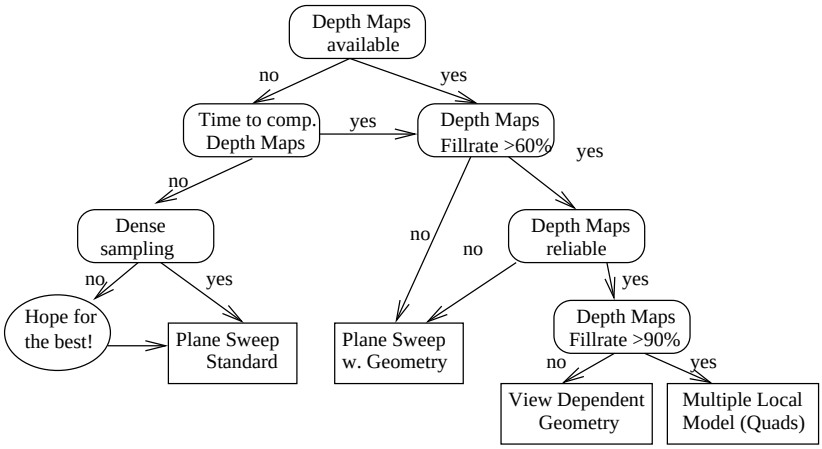


Figure 7.1: Diagram to find the best suited rendering method. All thresholds are only suggestions.

Second, there are some general recommendations:

- If the quality of the depth maps is sufficently high, the Multiple Local Models methods are the best choice.

- Multiple Local Models is the method with the smallest requirements concerning CPU and GPU.
- If the GPU is less powerful, Plane-Sweep cannot achieve interactive framerates.
- Dynamic content with multiple cameras (no depth estimation possible) can only be handled with Plane-Sweep without geometry.
- View-Dependent Geometry needs a powerful CPU to achieve interactivity.

7.3 Contribution of this Thesis

The main contribution of this thesis is the development of three different Image-Based Rendering algorithms with additional subtypes. To evaluate all proposed algorithms, six prototypes have been realised. These six prototypes have been studied in great detail. All aspects like image quality, depth reconstruction, time coherence and rendering time have been analysed. The complete work includes the following phases:

Taxonomy: In the last decade, many different Image-Based Rendering systems have been proposed. To understand this evolutionary process and the relations between different approaches, a taxonomy has been introduced and many known systems have been classified according to this taxonomy.

Requirements and Constraints: After defining the goals for an Image-Based Rendering system, its requirements with regards to the input data have been identified. This leads to the developement of basically three different prototypes.

Rendering Methods: In this phase, the basic algorithms have been developed and implemented as prototypes. To achieve usable systems, necessary additional parts like camera selection and blending have been added.

- Prototype $P1$ is called View-Dependent Geometry because for each new view to render, an approximate geometry proxy is constructed on-the-fly and real images are warped into the new view via this surface.
- Prototype $P2$ is called Multiple Local Models. For each real view, a local geometry model is prepared and multiple models are blended for a new image. Three modelling strategies have been proposed.

- Prototype *P*3 uses a Plane-Sweep algorithm to find photo-consistency between real views. It has been extended to use geometrical information where available to enhance the quality and performance.

Detailed Analysis: An evaluation of the prototypes has been conducted for several reasons. Firstly it has to be proved that the algorithms fulfil the requirements. Secondly, differences between the prototypes have to be identified. Thirdly, it has to be verified which image quality can be expected and what quality of input data has to be matched.

Appropriate error metrics have been defined to analyse:

- image quality,
- depth reconstruction,
- unreconstructed regions,
- coherence between adjacent views,
- rendering time.

Three different data sets and experiments have been used to vary the following aspects:

- spatial resolution (number of real views),
- layout of the real views (1-/2-dimensional),
- quality of depth maps,
- interpolation vs. extrapolation,

7.4 Conclusions and Future Research

Three new methods have been presented to generate novel views from existing images. For the given requirements and quality of input data all methods can interpolate images with reasonable quality. If these methods are put in the context of all the systems presented in chapter 3 (see figure 7.2), it shows that the new methods are more general approaches than most of the others. The three prototypes developed in this thesis have been designed to be as universal as possible (refer to section 4.1 for the design goals).

Nevertheless, none of the new approaches has solved the problem of Image-Based Rendering once and for all. The experiments have shown that reducing the spatial sampling density also reduces the quality of the visual reconstruction. An evaluation of the result from the synthetic data (Castle) reveals that the relation between sampling density and visual quality is not a direct dependency in

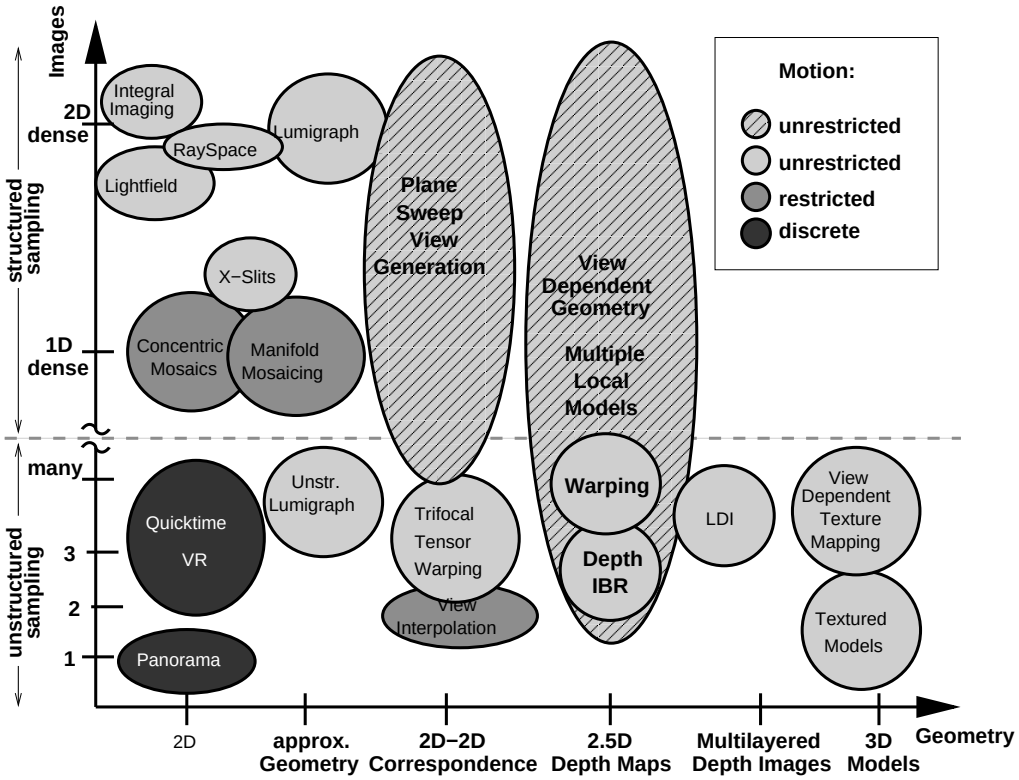


Figure 7.2: The prototypes developed in this thesis (striped) are designed to be more general than most other approaches. View-Dependent Geometry and Multiple Local Models share most properties.

most cases. It is an indirect dependency introduced by the geometrical reconstruction quality. Errors in the depth maps become more visible with increasing distances and the depth estimation itself becomes more challenging. Image-Based Rendering without geometry is restricted to very dense sampling. The better the geometry data, the better results can be achieved for sparse sampling. Further on this assumption leads to a more or less complete geometrical model with view-dependent texture mapping.

Considering the results of the image quality analysis, it is obvious that the ideal Image-Based Rendering system, if at all possible, has not yet been developed. Different aspects of the presented approaches leave room for improvements.

- The first and most dominant source of reconstruction errors are incomplete and incorrect depth maps. So enhancing the geometrical reconstruction will also enhance the visual reconstruction. This is even more important for decreased spatial sampling density (fewer real views).

- Instead of better geometrical reconstruction, algorithms like the Plane-Sweep can circumvent unreliable input data. Enhancing Plane-Sweep rendering could start by increasing the depth resolution and reducing mismatches [1].
- Image-Based Rendering is by its definition not limited to isolated objects. It is not even limited to perspective images. A great extension would be to use spherical real views to capture a more complete environment and then generate perspective images as required by most applications. This would allow to virtualize complete rooms with only a few plenoptic samples.

[1] This directly leads to better depth estimation.

Appendix A

Additional Background Information

A.1 MPEG-1 Compression

MPEG-1 is a standardised video compression method defined by the *Motion Picture Expert Group* and released in 1991. The main idea to compress motion picture sequences efficiently is to reduce the redundancies between consecutive frames by coding differences to already processed frames. The standard defines four different types of frames: I, P, B, D (D is not discussed here). The relevant three types of frames are:

I (Intra) Intra frames are compressed for themselves without information dependencies from their neighbours. They can be decoded easily without any dependencies.

P (Predicted) P-frames are encoded in dependence on their predecessor (I-or P-frames only). Thus, to decode a P-frame, the previous I- or P-frame has to be decoded before.

B (Bidirectional) B-frames depend on their predecessor and their successor, equally. To decode them, both neighbour frames have to be decoded completely.

Sequences of I-, P- and B-frames are assembled to so-called *Group of pictures (GOP)*. A GOP always starts with an I-frame followed by B- and P-frames. A GOP of size 16 can for example be: IBBPBBPBBPBBPBBP.

I-frames are encoded in a way similar to the JPEG-compression standard for still images. Basically, the image is split into macro blocks (16x16 pixels) and each block is coded individually. A frequency analysis (DCT) is applied and frequency components are quantised according to a quantisation table.

P-frames are encoded in reference to their preceding frame. The basic principle works as follows (see figure A.1): To compress a new frame I_t, this frame is

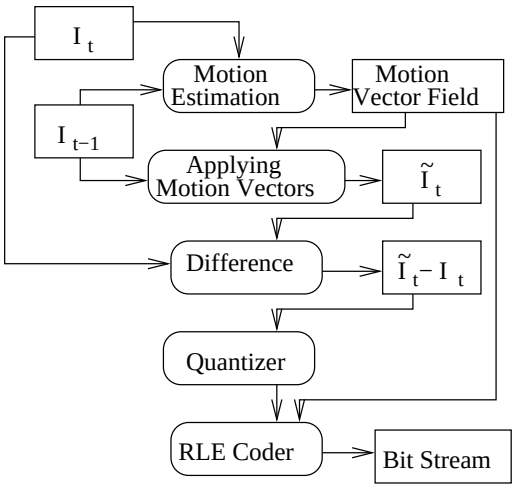


Figure A.1: Compression scheme for MPEG1 P-frames.

divided into macro blocks and for each macro block, its most probable position in the previous frame I_{t-1} is determined. This is called *motion estimation* and it can be done by computing the correlation of the macro block with the region in the I_{t-1} within a given search range. The result of this motion estimation is a motion vector field with a two-dimensional vector for each macro block. This vector is the difference of the macro block position in I_t and in I_{t-1}.

Then this motion vector field is applied to I_{t-1} giving $\tilde{I}_t$, a prediction of I_t from I_{t-1}. In the next step, the difference image between the prediction and the real image $\tilde{I}_t - I_{t-1}$ is computed. This contains the residual error, parts which cannot be predicted from I_{t-1}. The difference image is quantised in a way similar to the I-frames and, together with the motion vector field, compressed by a run-length encoder.

A.2 Graphics Processing Units

The performance of all proposed algorithms has been measured on 6 different GPUs. Table A.1 gives an overview of the technical details. The ATI [www.ati.com, ati.amd.com] Radeon X300 have been taken as reference only. In section 6.9 the results of three different generations of NVidia [www.nvidia.com] Geforce GPUs are evaluated.

Vendor	NVidia	NVidia	NVidia	NVidia	NVidia	ATI
Model	8800GTX	7950GT	7800GTX	6800GT	6600GT	X300
Codename	G80	G71	G70	NV45	NV43	M22
Released	11/06	06/05	06/05	04/07	04/04	01/05
Shader Units	128U	8V,24P	8V,24P	6V,16P	3V,8P	
Core Clock (MHz)	575	550	430	350	500	350
Memory Clock (MHz)	900	1400	1200	1000	1000	300
Memory Amount	768	512	512	512	128	64
Memory Interface (bit)	384	256	256	256	128	64
Memory Bandwidth (GB/sec)	86,4	44,8	54,4	35,2	26	4,8
Texture Fill Rate (billion/sec)	36.8	13,2	13,2	5,6	4	1,3
Vertices / sec (Million)		1100	1100	600	380	500?

Table A.1: Details of all used graphics processing units (GPU). The number of shader units is given for vertex- (V), pixel- (P) or unified (U) shaders. The core clock of the 8800 is 575 MHz and the shader-units work at 1350 MHz.

Appendix B

Colour Images

This chapter contains colour images from all analysed data sets. First, overview pictures of the scenes are given. The Castle data set has been synthesised with the

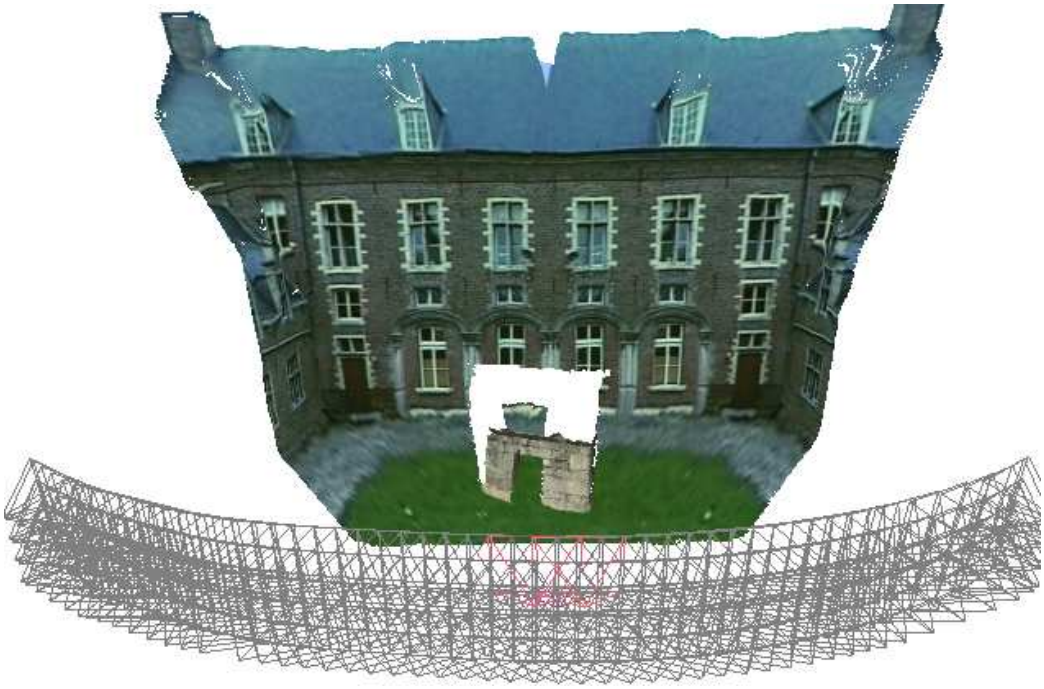


Figure B.1: The Castle data set consists of 206 views in 4 rows.

use of a modified 3D model of the Leuven-Castle (Belgium) and a 3D model of an arch in the foreground. This introduces occlusions and depth discontinuities. Images have been taken in 4 rows as shown in figure B.1. An example image and the corresponding depth map is shown in figure 6.2.

Figure B.2: The Studio data set consists of 50 views in 4 rows.

The Studio data set has been captured at the BBC Research Studio near London using a TV-Camera mounted on a dolly. Scans in four different heights have been performed and the calibration has been done with structure-from-motion algorithms. An overview is given in figure B.2 and an example image together with the corresponding depth map is shown in figure 6.3.

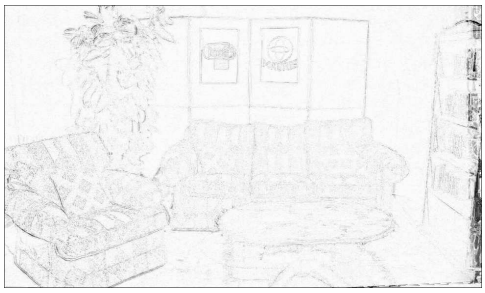
The City data set has been captured with two cameras mounted on the roof of a car driving through the city centre of Chapel Hill, North Carolina, USA. Calibration and depth estimation has been done with a real-time structure-from-motion and modelling system developed at the University of North Carolina (UNC). A complete overview cannot be given, a small part of the sequence is shown in figure B.3.

B.1 Studio Results

B.2 City Results

Figure B.3: The City data set consists of 150 views in one row.

(a) View-Dependent Geometry

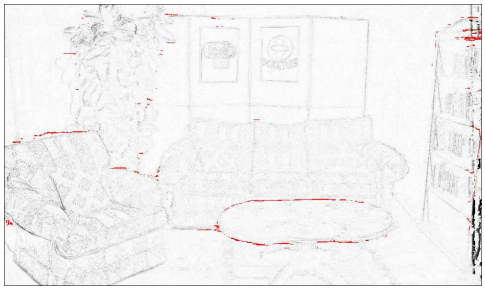


(b) View-Dependent Geometry

(c) Multiple Local Models: Quads

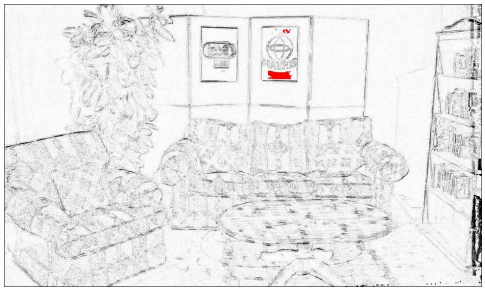
(d) Multiple Local Models: Quads

(e) Multiple Local Models: Triangles



(f) Multiple Local Models: Triangles

(g) Multiple Local Models: Points



(h) Multiple Local Models: Points

(i) Plane-Sweep: Geometry

(j) Plane-Sweep: Geometry

(k) Plane-Sweep: Pure

(l) Plane-Sweep: Pure

Figure B.4: Rendered images of the Studio data set with dense setup and the results of the analysis (difference image). Dark pixels indicate deviations from the reference image.

(a) View-Dependent Geometry

(b) View-Dependent Geometry

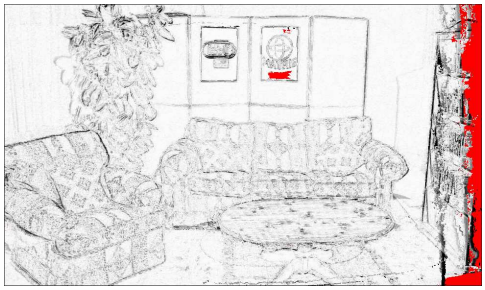
(c) Multiple Local Models: Quads

(d) Multiple Local Models: Quads

(e) Multiple Local Models: Triangles

(f) Multiple Local Models: Triangles

(g) Multiple Local Models: Points



(h) Multiple Local Models: Points

(i) Plane-Sweep: Geometry

(j) Plane-Sweep: Geometry

(k) Plane-Sweep: Pure

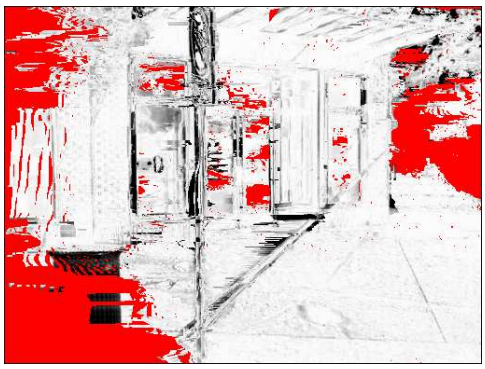
(l) Plane-Sweep: Pure

Figure B.5: The Studio data set with extra sparse setup. Left: Rendered Images, right: Mean absolute error, red pixels are unfilled.

(a) View-Dependent Geometry

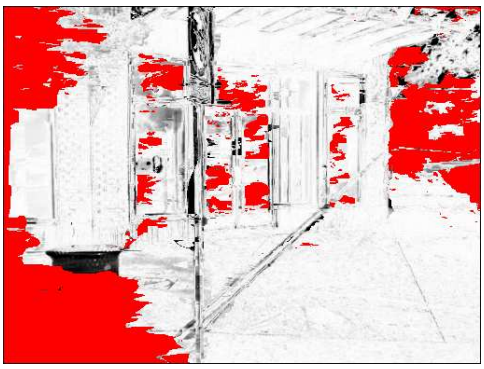
(b) View-Dependent Geometry

(c) Multiple Local Models: Quads



(d) Multiple Local Models: Quads

(e) Multiple Local Models: Triangles



(f) Multiple Local Models: Triangles

Figure B.6: Rendered images and mean absolute error (magnified) of the City data set.

(a) Multiple Local Models: Points

(b) Multiple Local Models: Points

(c) Plane-Sweep: Geometry

(d) Plane-Sweep: Geometry

(e) Plane-Sweep: Pure

(f) Plane-Sweep: Pure

Figure B.7: Analysed Images of the City data set.

Appendix C

All Results

In sections 6.5 - 6.8 all results are presented in a condensed form plotting only the mean and the standard deviation. In this section, the results for all analysed data sets and setups are shown as plots over the image number. This gives an impression of the distribution of results. These graphs are not discussed in detail.

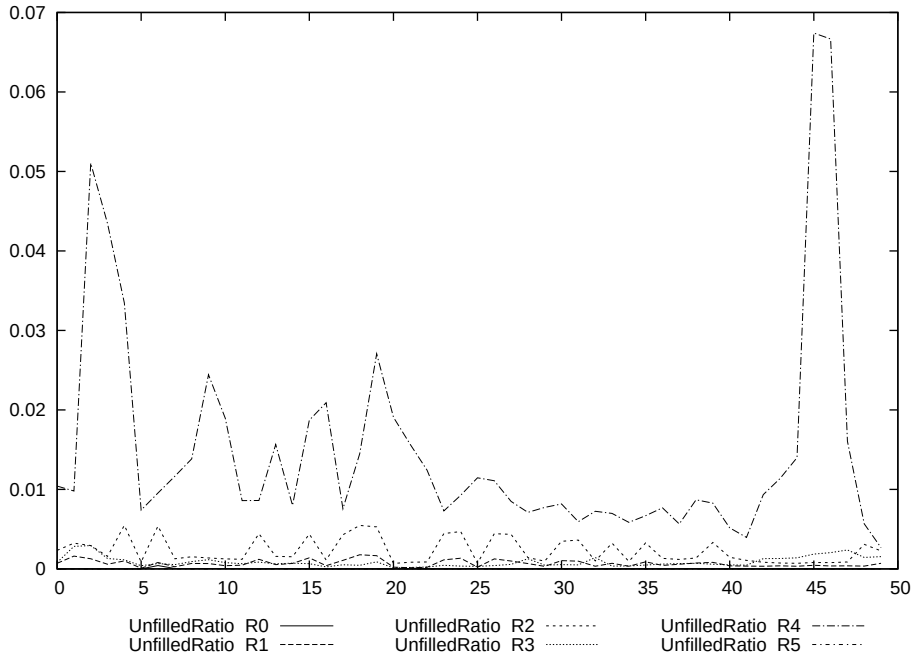


Figure C.1: Castle-Dense UnfilledRatio

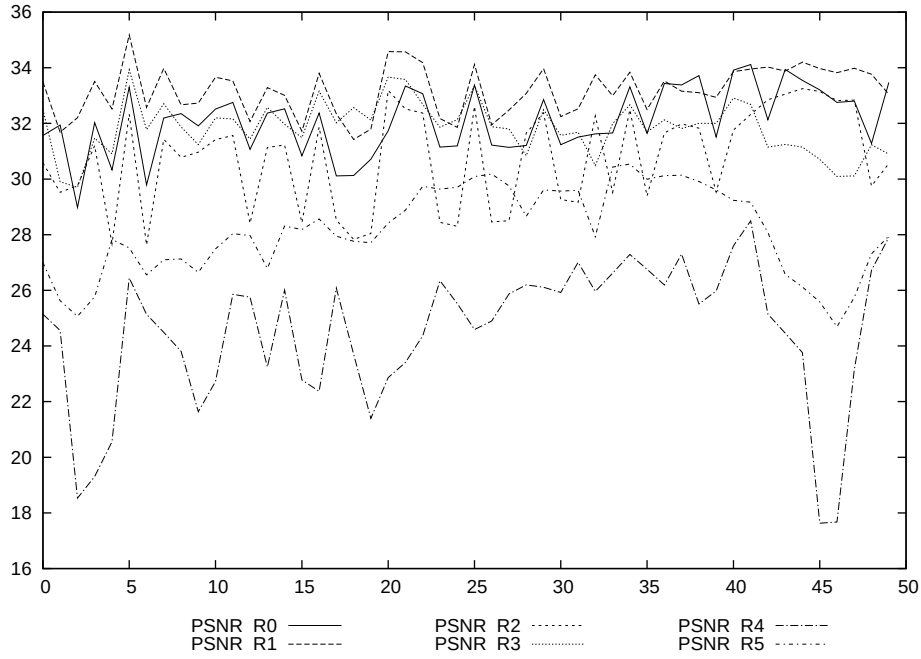


Figure C.2: Castle-Dense PSNR

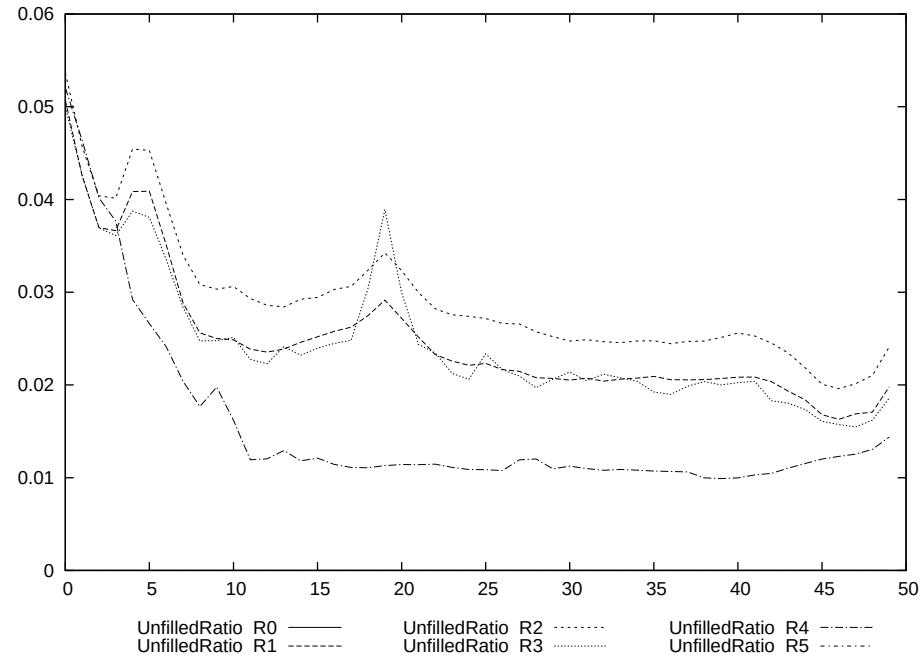


Figure C.3: Castle-Extrapol UnfilledRatio

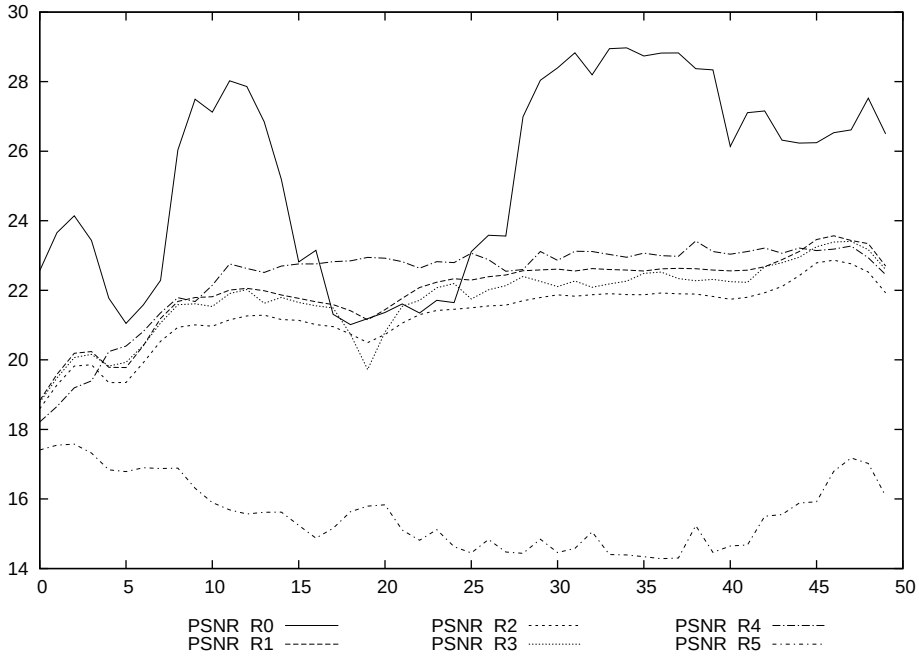


Figure C.4: Castle-Extrapol PSNR

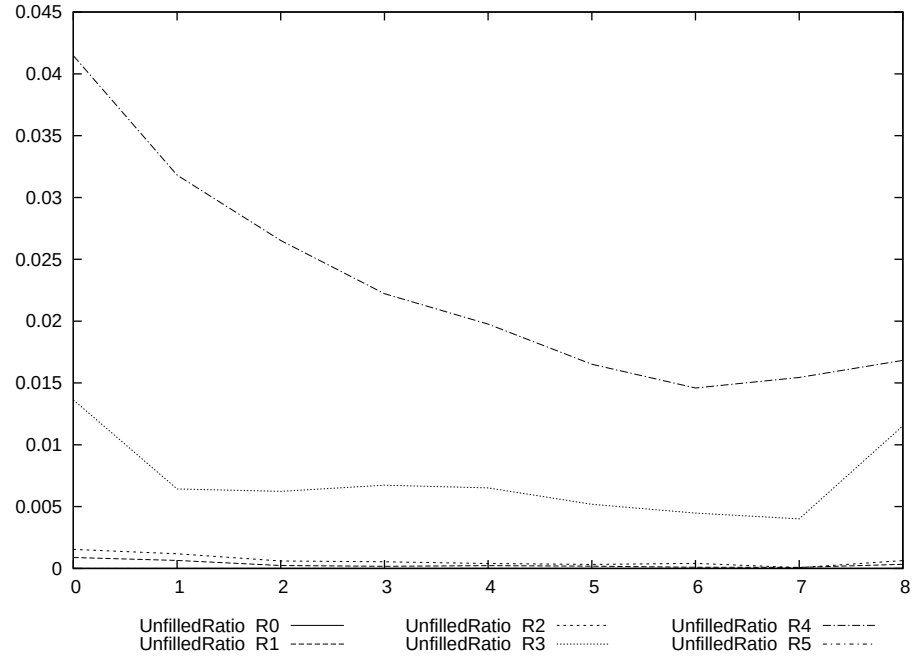


Figure C.5: Castle-Sparse UnfilledRatio

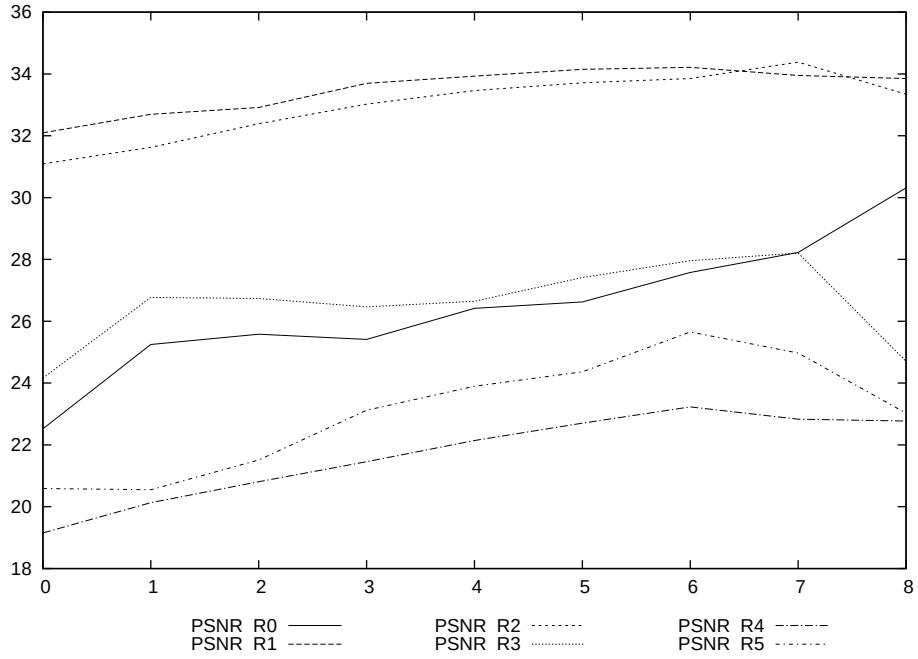


Figure C.6: Castle-Sparse PSNR

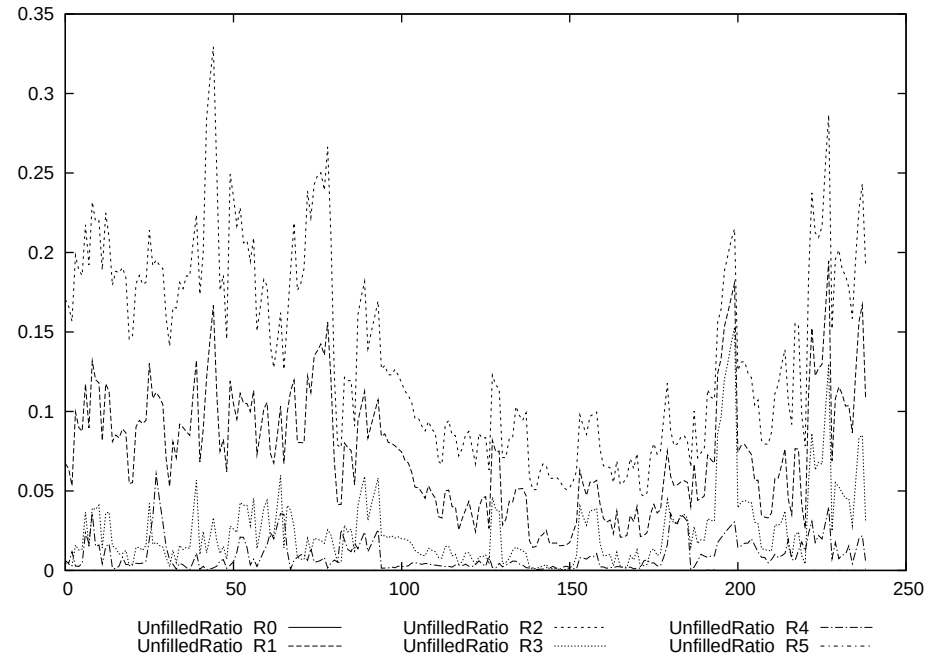


Figure C.7: City-Dense UnfilledRatio

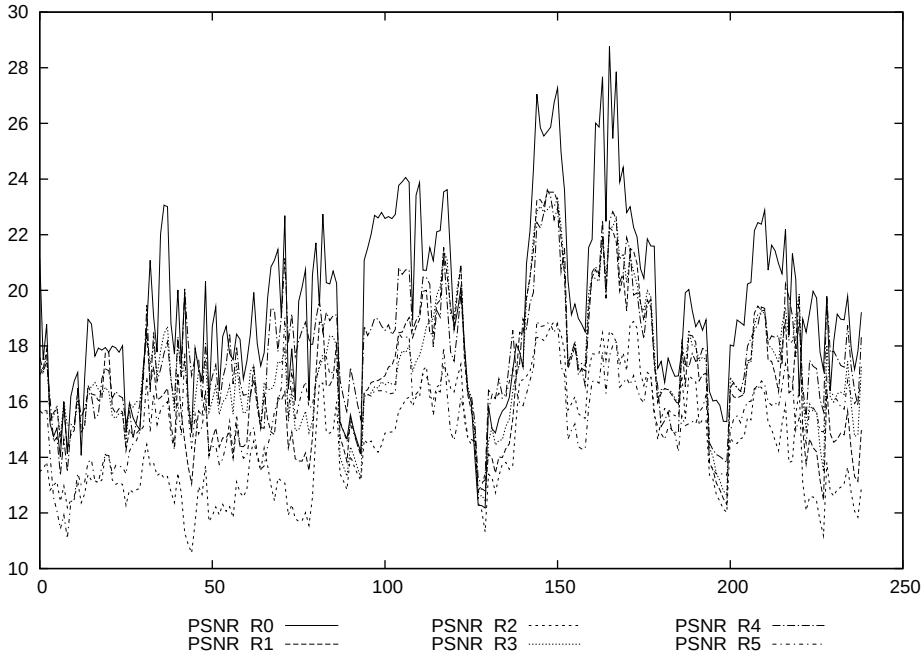


Figure C.8: City-Dense PSNR

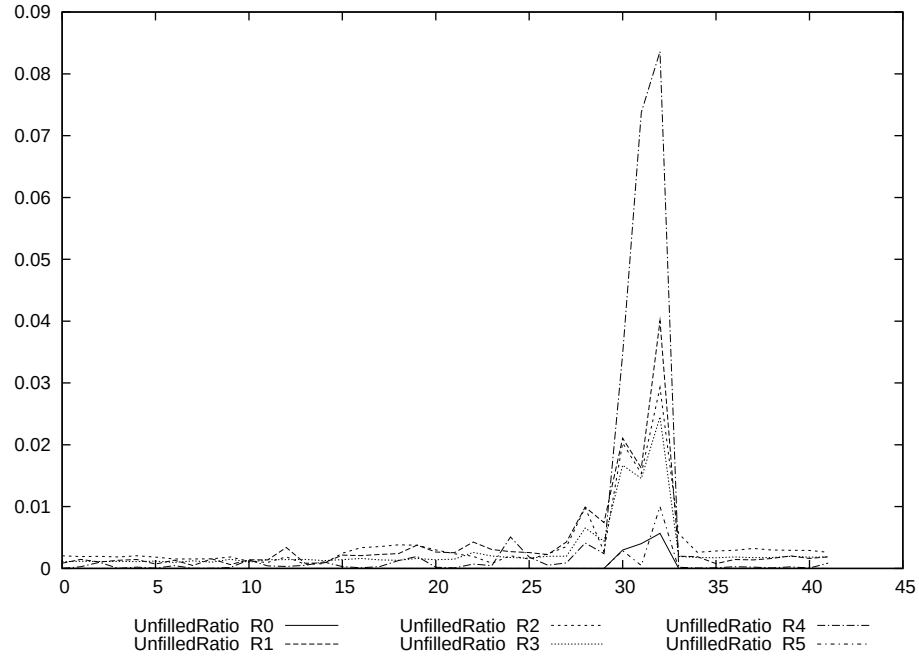


Figure C.9: Studio-Dense UnfilledRatio

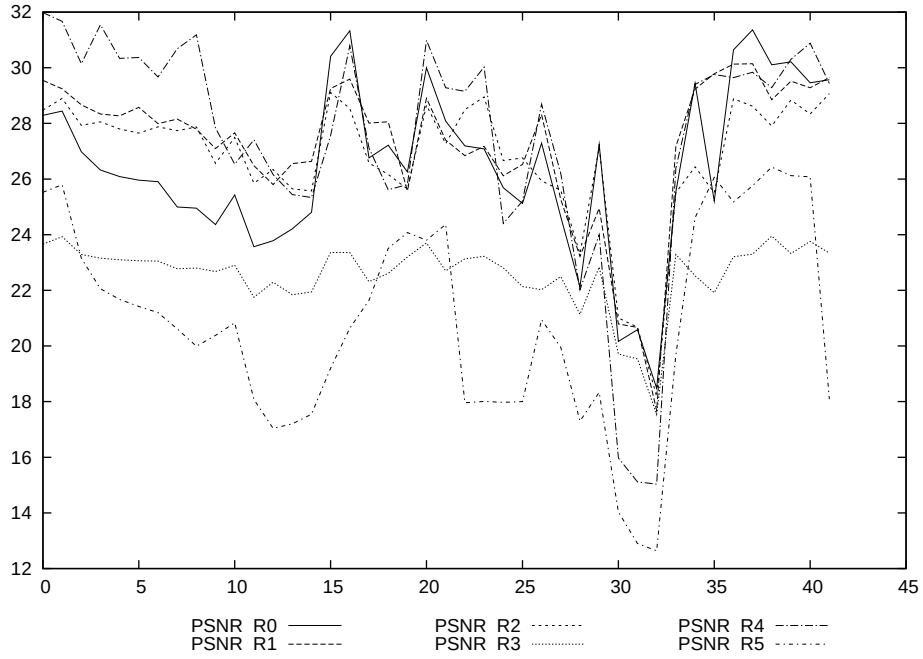


Figure C.10: Studio-Dense PSNR

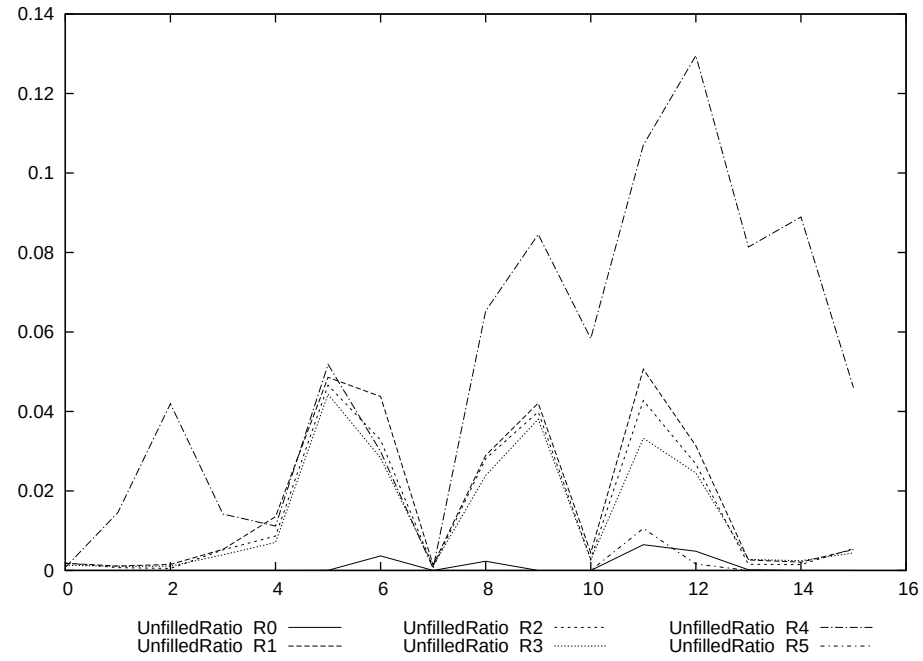


Figure C.11: Studio-ExtraSparse UnfilledRatio

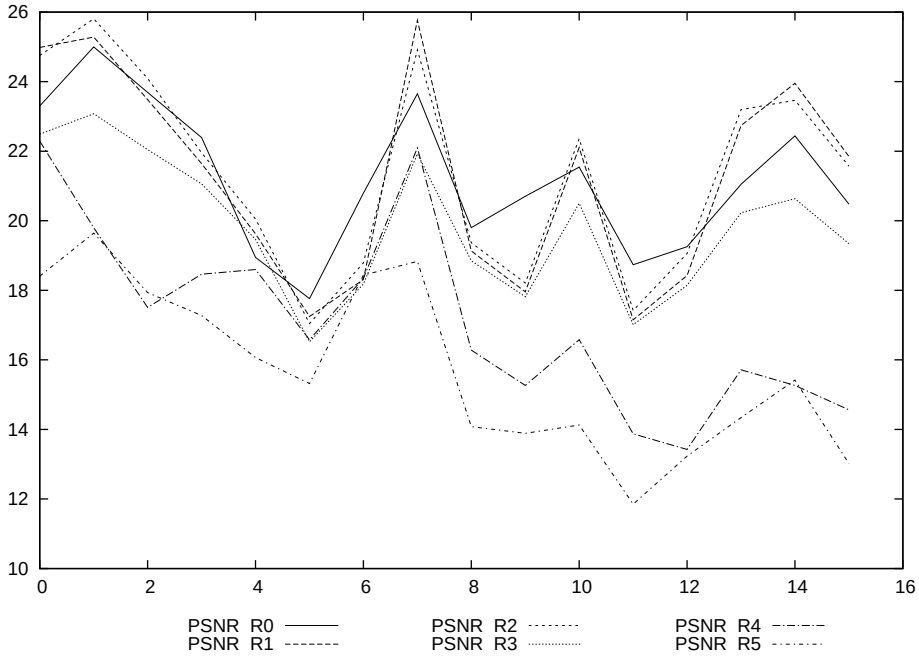


Figure C.12: Studio-ExtraSparse PSNR

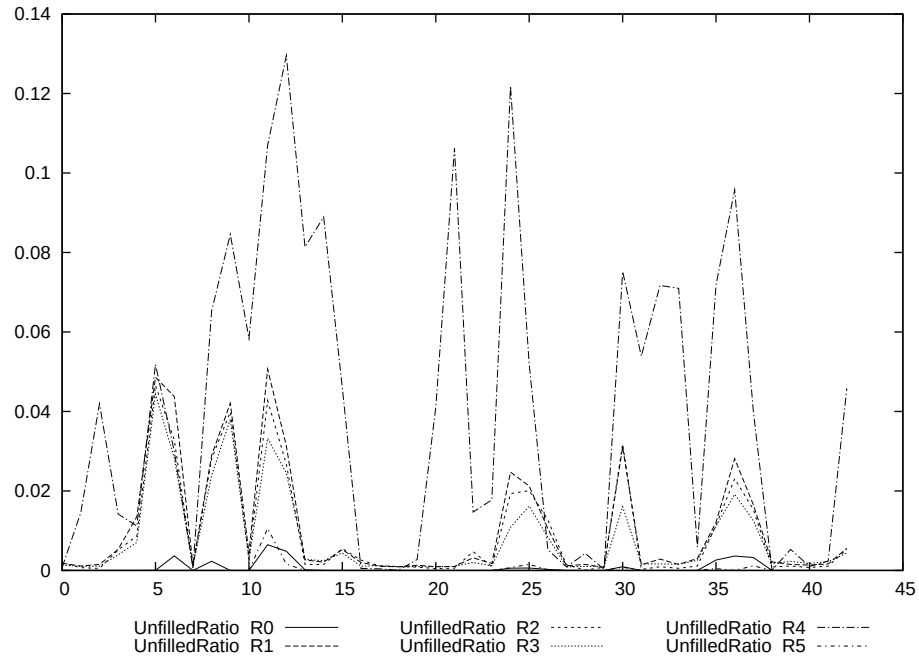


Figure C.13: Studio-Sparse UnfilledRatio

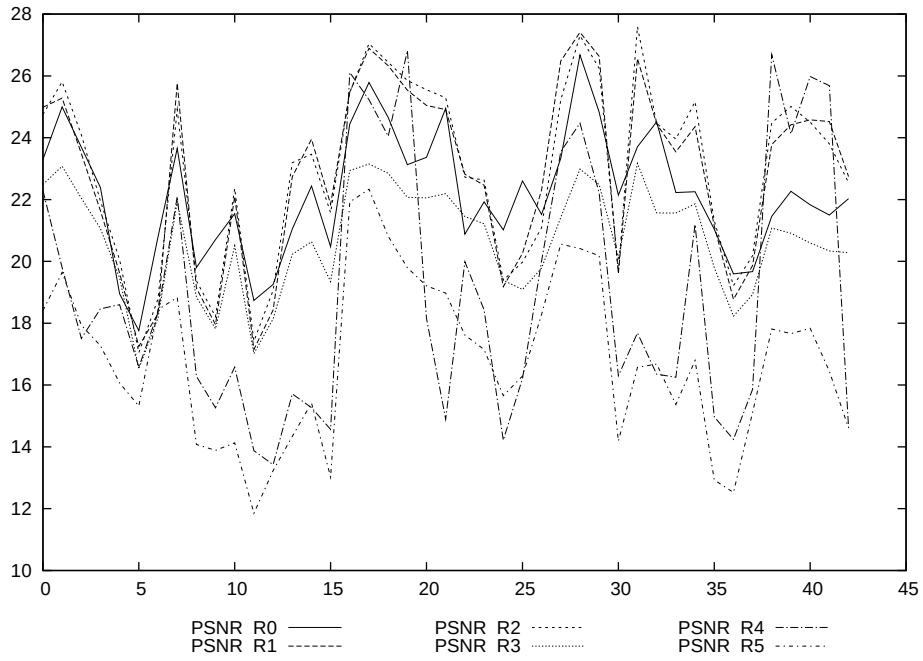


Figure C.14: Studio-Sparse PSNR

Bibliography

E. Adelson and J. Bergen. The Plenoptic Function and the Elements of Early Vision. In M. Landy and J. Movshon, editors, *Computation models of visual processing*, pages 3–20. MIT Press, 1991.

M. Antone and S. Teller. Extrinsic Calibration of Omni-Directional Image Networks. *International Journal of Computer Vision*, 2002.

S. Avidan and A. Shashua. Novel view synthesis in tensor space. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 1034. IEEE Computer Society, 1997. ISBN 0-8186-7822-4.

S. Baker and S. K. Nayar. A Theory of Catadioptric Image Formation. In *Proc. of IEEE International Conference on Computer Vision (ICCV)*, pages 35–42, Bombay, January 1998.

H. Bakstein and T. Pajdla. Non-central cameras for 3D reconstruction. In *Proceedings of Workshop 2003*, pages 240–241, Faculty of Architecture of CTU, Prague, Czech republic, February 2003a. Czech Technical University in Prague, CTU Publishing House. ISBN 80-01-02708-2.

H. Bakstein and T. Pajdla. Ray space volume of omnidirectional180x360 deg. images. In O. Drbohlav, editor, *Computer Vision — CVWW'03 : Proceedings of the8th Computer Vision Winter Workshop*, pages 39–44, Prague, Czech Republic, February 2003b. Czech Pattern Recognition Society. ISBN 80-238-9967-8.

H. Bakstein and T. Pajdla. Rendering novel views from a set ofomnidirectional mosaic images. In *Proceedings of Omnivis 2003: Workshop onOmnidirectional Vision and Camera Networks*, page 6, Los Alamitos, USA, June 2003c. IEEE Computer Society Press. ISBN 0-7695-1900-8.

R. C. Bolles and H. H. Baker. Epipolar-plane image analysis: A technique for analyzing motion sequences. Technical Report 377, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Feb 1986.

C. Buehler, M. Bosse, L. McMillan, S. J. Gortler, and M. F. Cohen. Unstructured lumigraph rendering. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 425–432. ACM Press / ACM SIGGRAPH, 2001.

J.-X. Chai, S.-C. Chan, H.-Y. Shum, and X. Tong. Plenoptic sampling. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 307–318. ACM Press/Addison-Wesley Publishing Co., 2000. ISBN 1-58113-208-5.

N. L. Chang and A. Zakhor. A Multivalued Representation For View Synthesis. In *Proceedings of the International Conference on Image Processing (ICIP)*, pages 505–509, Kobe, Japan, October 1999.

S. Chen and L. Williams. View interpolation for image synthesis. In *Siggraph 1993, Computer Graphics Proceedings*, pages 279 – 288, 1993.

S. E. Chen. QuickTime VR — an image-based approach to virtual environment navigation. *Computer Graphics*, 29(Annual Conference Series):29–38, 1995.

W.-C. Chen, J.-Y. Bouguet, M. H. Chu, and R. Grzeszczuk. Light field mapping: Efficient representation andhardware rendering of surface light fields. In J. Hughes, editor, *SIGGRAPH 2002 Conference Proceedings*, Annual Conference Series, pages 447–456. ACM Press/ACM SIGGRAPH, 2002.

R. Collins. A space-sweep approach to true multi-image matching. In *Proc. Computer Vision and Pattern Recognition Conf*, pages 358–363, 1996.

R. T. Collins. A space-sweep approach to true multi-image matching. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 358–363, 1996.

E. Cooke, P. Kauff, and O. Schreer. Image-based rendering for tele-conference systems. In *Proc. of WSCG 2002, 10th Int. Conference on Computer Graphics, Visualization and Computer Vision*, page 119, Plzen, Czech Republic, Feb. 2002.

P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. *Computer Graphics*, 32(Annual Conference Series):189–198, 1998.

P. Debevec, Y. Yu, and G. Boshokov. Efficient view-dependent image-based rendering with projective texture-mapping. Technical Report CSD-98-1003, University of California, Berkeley, 1998.

J.-F. Evers-Senne and R. Koch. Image Based Interactive Rendering with View Dependent Geometry . In *Eurographics 2003*, Computer Graphics Forum, pages 573–582. Eurographics Association, 2003.

C. Fehn. Depth-Image-Based Rendering (DIBR), Compression and Transmission for a New Approach on 3D-TV. In *Proc. Stereoscopic Displays and Applications*, San Jose, CA, USA, January 2004.

R. Fernando and M. J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, 2003.

R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

M. Fischler and R. Bolles. *RANdom SAmpling Consensus: a paradigm for model fittingwith application to image analysis and automated cartography*, pages 381–395. Commun. Assoc. Comp. Mach. 24, 1981.

T. Fujii. *A Basic Study in the Integrated 3-D Visual Communication*. PhD thesis, University of Tokyo, 1994.

T. Fujii and M. Tanimoto. Free viewpoint TV system based on ray-space representation. *Three-Dimensional TV, Video, and Display*, 4864:175–189, 2002.

C. Geyer and K. Daniilidis. Omnidirectional Video. *The Visual Computer*, pages 405 – 416, 2003.

B. Goldluecke and M. Magnor. Real-time, free-viewpoint video rendering from volumetric geometry. In T. Ebrahimi and T. Sikora, editors, *Visual Communications and Image Processing 2003*, volume 5150 of *Proceedings of SPIE*, pages 1152–1158, Lugano, Switzerland, June 2003. The International Society for Optical Engineering (SPIE), SPIE. ISBN 0-8194-5023-5.

S. J. Gortler, R. Grzeszczuk, and R. S. M. F. Cohen. The lumigraph. *Proceedings SIGGRAPH '96*, 30(Annual Conference Series):43–54, 1996.

R. Hartley and A. Zissermann. *Multiple View Geometry in Computer Vision*. Cambridge university press, second edition, 2004.

B. K. P. Horn. Tsai's camera caliration method revisited. http://people.csail.mit.edu/bkph/articles/Tsai_Revisited.pdf, 2000.

R. W. Hunt. *The Reproduction of Color*. Wiley, 2004.

R. W. Hunt. *Measuring colour*. Fountain Press, England, 3rd edition, 1998.

A. Isaksen, L. McMillan, and S. J. Gortler. Dynamically reparameterized light fields. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 297–306. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

O. S. J. Heikkilae. A four-step camera calibration procedure with implicit imagecorrection. In *Proceedings CVPR*, 1997.

J.C.McGlone, editor. *Manual of Photogrammetry*, chapter 2, pages 98–102. ASPRS, 5th edition, 2004.

S. B. Kang. A Survey of Image-based Rendering Techniques. Technical report, DEC Cambridge Research Lab, August 1997.

S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High dynamic range video. *ACM Trans. Graph.*, 22(3):319–325, 2003. ISSN 0730-0301.

H. Kawasaki, K. Ikeuchi, and M. Sakauchi. Light Field Rendering for Large-Scale Scenes. In *Computer Vision and Pattern Recognition (CVPR)*, volume 2, page 2, Hawaii, USA, Dec 2001.

R. Klette, G. Gimel'farb, S. Wei, F. Huang, K. Scheibe, M. Scheele, A. Börner, and R. Reulke. On design and applications of cylindrical panoramas. In *Proc. Computer Analysis of Images and Patterns*, pages 1–8, Groningen, The Netherlands, August 2003.

L. Kobbelt and M. Botsch. A survey of pointbased techniques in computer graphics. *Computers and Graphics 28, 6 (2004). SEIDEL H.-P.: Feature-sensitive surfaceextraction from volume data. In Proceedings of ACM SIGGRAPH (2001).*, 2004.

R. Koch. *Automatische Oberflächenmodellierung starrer dreidimensionalerObjekte aus stereoskopischen Rundum-Ansichten*. PhD thesis, Universität Hannover, 1996.

R. Koch and J.-F. Evers-Senne. View synthesis and rendering methods. In *3D Video Communications*. Wiley, 2005.

R. Koch, M. Pollefeys, and L. V. Gool. Multi viewpoint stereo from uncalibrated video sequences. In *Proc. ECCV'98*, volume 1 of *LNCS 1406*, pages 55 – 71, Freiburg, 1998. Springer-Verlag.

R. Koch, M. Pollefeys, B. Heigl, L. Van Gool, and H. Niemann. Calibration of handheld camera sequences for plenoptic modeling. In *Proceedings ICCV 99*, Korfu, Greece, 1999.

C. Kurashima, R. Yang, and A. Lastra. Combinig Approximate Geometry with View-Dependent Texture Mapping. In *XV Brazilian Symposium on Computer Graphics and Image Processing*, pages 112–120, Fortaleza, CE, Brazil, October 2002.

S. Laveau and O. Faugeras. 3-d representation as a collection of images. In *Proc.ofthe IEEE Int. Conf. on Pattern Recognition (CVPR'97)*, pages 689–691. IEEE Publishers, 1997.

M. Levoy and P. Hanrahan. Light field rendering. *Proceedings SIGGRAPH '96*, 30(Annual Conference Series):31–42, 1996.

M. Li, M. Magnor, and H.-P. Seidel. Improved hardware-accelerated visual hull rendering. *Proc. Vision, Modeling, and Visualization(VMV-2003)*, Munich, Germany, pages 151–158, Nov. 2003.

A. Lippman. Movie-Maps: An Application of the Optical Videodisc to Computer Graphics. In *Proc. ACM SIGGRAPH*, pages 32–42, 1980.

L. McMillan and S. Gortler. Image-Based Rendering: A New Interface Between Computer Vision and Computer Graphics. In *ACM SIGGRAPH Computer Graphics 33:4 (Applications of Computer Vision to Computer Graphics)*, pages 61–64, November 1999.

M. Meyer. Generalized barycentric coordinates on irregular polygons. *Journal of Graphics Tools*, 7:1086–7651, 2002.

T. Naemura, T. Yoshida, , and H. Harashima. 3-D Computer Graphics Based on Integral Photography. *Optics Express*, 8:255 – 262, Feb. 2001.

V. Nozick, S. Michelin, and A. D. Real-time plane-sweep with local startegy. In *Proceedings of WSCG 2006*, Plzen, Czech Republic, 2006.

S. Peleg and J. Herman. Panoramic mosaics by manifold projection. In *CVPR97*, pages 338–343, 1997.

H. Pfister, M. Zwicker, and J. van Baarand Markus Gross. Surfels: Surface elements as rendering primitives. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 335–342. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

M. Pollefeys, R. Koch, and L. J. VanGool. Self-calibration and metric reconstruction in spiteof varying and unknown internal camera parameters. *International Journal of Computer Vision*, 32(1):7–25, 1999.

P. Rademacher and G. Bishop. Multiple-center-of-projection images. In *Proceedings of SIGGRAPH '98*, pages 199–206. ACM Press, 1998.

L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Computer Graphics Forum (Eurographics 2002)*, volume 21, pages 461–470, Sept. 2002.

B. Rousso, S. Peleg, and I. Finci. Mosaicing with generalized strips. In *DARPA97*, pages 255–260, 1997.

B. Rousso, S. Peleg, I. Finci, and A. Rav-Acha. Universal mosaicing using pipe projection. In *ICCV98*, pages 945–952, 1998.

S. Roy and I. J. Cox. A maximum-flow formulation of the n-camera stereo correspondenceproblem. In *ICCV*, pages 492–502, 1998.

H. Saito, S. Baba, M. Kimura, S. Vedula, and T. Kanade. Appearance-Based Virtual View Generation of Temporally-Varying Events from Multi-Camera Images in the 3D Room. In *Proceedings of Second International Conference on 3-D Digital Imaging and Modeling*, pages 516 – 525, October 1999.

G. Schaufler and M. Priglinger. Efficient Displacement Mapping by Image Warping. In *10th Eurographics Workshop on Rendering*, pages 175–186, 1999.

M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast shadows and lighting effects using texture mapping. In *Proceedings of SIGGRAPH 1992*, 1992.

S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring, 1997a.

S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring, 1997b.

S. M. Seitz and C. R. Dyer. View morphing. In *SIGGRAPH 96*, pages 21–30, 1996.

J. Shade, S. Gortler, L. W. He, and R. Szeliski. Layered depth images. In *Proceedings ACM SIGGRAPH*, pages 231–242. ACM Press / ACM SIGGRAPH, 1998.

J. R. Shewchuk. A two-dimensional quality mesh generator and delaunay triangulator. http://www.cs.cmu.edu/∼quake/triangle.htm, 1996.

H. Shum and R. Szeliski. Panoramic image mosaics. Technical report, Microsoft Research, 1997.

H. Shum, L. Wang, J. Chai, and X. Tong. Rendering by Manifold Hopping. *International Journal of Computer Vision (IJCV)*, 50(2):185 – 201, 2002.

H.-Y. Shum and L.-W. He. Rendering with concentric mosaics. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 299–306. ACM Press/Addison-Wesley Publishing Co., 1999. ISBN 0-201-48560-5.

H.-Y. Shum and S. B. Kang. A review of image-based rendering techniques. In *Proceedings Visual Communications and Image Processing*, pages 2–13, 2000.

H.-Y. Shum, S.-C. Chan, and S.-B.Kang. *Image-Based Rendering*. Springer, 2005.

T. Takahashi, H. Kawasaki, K. Ikeuchi, and M. Sakauchi. Arbitrary view position and direction rendering forlarge-scale scenes. In *Proceedings CVPR 2000*, pages 296–303, 2000.

S. Teller, M. Antone, Z. Bodnar, M. Bosse, S. Coorg, M. Jethwa, , and N. Master. Calibrated, Registered Images of an Extended Urban Area. *International Journal of Computer Vision*, pages 93–107, 2003.

X. Tong, J. Chai, and H. Shum. Layered Lumigraph with LOD Control. *Journal of Visualization and Computer Animation*, 13(4):249–261, 2002.

R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv camerasand lenses,an efficient and accurate camera calibration tec. *IEEE Journal of Robotics and Automation*, RA-3 (4):323–344, 1987.

R. Unbehauen. *Systemtheorie 1*. Oldenbourg, 2002.

P. Verlani, A. Goswami, P. J. Narayanan, S. Dwivedi, and S. K. Penta. Depth images: Representations and real-time rendering. In *IEEE Proc. of 3st International Symposium on 3D DataProcessing Visualization and Transmission (3DPVT'06)*, Chapel Hill, NC, USA, June 2006.

C. Vogelgsang and G. Greiner. Interactive Range Map Rendering with Depth Interval Texture Slicing. In *Vision, Modelling and Visualization (VMV)*, Munich, Germany, November 2003.

D. Weinshall, M.-S. Lee, T. Brodsky, M. Trajkovic, and D. Feldman. New view generation with a bi-centric camera. In *Proc. of 7th European Conference of Computer Vision*, pages 614–618, Copenhagen, DK, May 2002.

J. Woetzel and R. Koch. Multi-camera real-time depth estimation with discontinuity handling on pc graphics hardware. In *17th International Conference on Pattern Recognition (ICPR 2004), Cambridge, United Kingdom, August 2004*, August 2004.

S. Yamazaki, R. Sagawa, K. Kawasaki, H.and Ikeuchi, and M. Sakauchi. Microfacet billboarding. In *Rendering techniques 2002 (Eurographics Workshop Proceedings)*, pages 169–179, June 2002.

R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphicshardware. In *Conf. Computer Vision and Pattern Recognition CVPR03*, Madison, WISC., USA, June 2003.

R. Yang, G. Welch, and G. Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Proceedings of Pacific Graphics*, pages 207–214, Beijing, China, October 2002.

C. Zitnick, S. Kang, M. Uyttendaele, S.Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. In *Proc. ACM SIGGRAPH*, pages 600–608, Los Angeles, CA, Aug. 2004.

A. Zomet, D. Feldman, S. Peleg, and D. Weinshall. Mosaicing New Views: The Crossed-Slits Projection. *IEEE Trans. on PAMI*, pages 741–754, June 2003.

M. Zwicker, H. Pfister, and J. van Baarand Markus Gross. Surface splatting. In *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 371–378. ACM Press, 2001.