MULTIPERSPECTIVE IMAGING FOR AUTOMATED URBAN VISUALIZATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL
ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Augusto Román
September 2006

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Marc Levoy Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Pat Hanrahan

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Sebastian Thrun

Approved for the University Committee on Graduate Studies.

Abstract

Traditional perspective images (such as those produced by a typical camera) cannot consistently display detail across all parts of an entire city block. It is inevitable that detail is lost in areas of the scene most distant from the camera. A multiperspective image generated from a collection of photographs or a video stream can be used to effectively summarize long, roughly planar scenes such as city streets. For example, we have generated a single continuous image of a street spanning approximately 10 city blocks. This image is over 300,000 pixels wide.

This single-image representation has several advantages over other possible representations (such as 360-degree panoramas, individual photographs, 3D models, or satellite maps) in that it is continuous, compact, high resolution, and requires no special viewing software. However, multiperspective images also suffer distortions caused by the deviation from the familiar perspective image.

Constructing multiperspective images with minimum distortion is typically done manually by an artist, however this is not practical for large-scale projects such as creating images along every street in an entire city. We describe how these images can be automatically constructed, including a technique to evaluate and minimize the distortion without requiring user intervention.

This thesis presents three contributions toward the use of multiperspective images in urban visualization. The first is a method of constructing images from serially blended crossed-slits mosaics that makes it possible to along significantly reduce the distortion in the final output. Second, an efficient method of rendering high-quality multiperspective images is described, along with a user-driven GUI program that allows a user to quickly manipulate the perspective structure of a multiperspective image and gain an intuition about parameters of such images. Finally, we present a metric for quantifying the distortion in these images, along with an optimization for automatically minimizing these distortions.

Acknowledgements

There are many people without whom this work would not be possible. In general, I'd like to thank the Stanford faculty for their insight and enthusiasm and my friends and family for their support.

I'd like to thank my adviser, Marc Levoy, for his bottomless well of research ideas and enthusiasm. Marc seized hold of my interest in computer graphics when teaching his Introduction to Computer Graphics course and the related computer game competition. His inspiring teaching is mirrored in his inspiration in research.

I am also grateful for the teachings of my committee chair, Brad Osgood. He taught one of the first classes I took here at Stanford and his enthusiasm set a high bar for my expectations.

Special thanks to my reading committee members Pat Hanrahan and Sebastian Thrun, again both professors that have an inspiring enthusiasm for their subjects.

Also, I am extremely grateful for the collaborations in this work with Gaurav Garg and Hendrik Lensch. Gaurav is both a great friend and a great researcher and instrumental to the start of this research. Similarly, Hendrik provided key insights into the design of the optimization and a joy to work with.

Much of this work depends heavily on having accurate camera pose and would not have been possible without the excellent pose optimization by James Diebel.

In addition to the guidance from the faculty, the environment sustained by fellow members of the computer graphics lab helped keep research both interesting and enjoyable.

My friends at Stanford, in particular Hrefna & Finnur, helped keep me sane at the busiest of times and relaxed at the best of times.

Last, I owe a tremendous debt to Sarah Harriman for her companionship in all aspects

of life.

This research was funded by the National Physical Science Consortium, the Stanford Dean's Doctoral Diversity fellowship, the Alfred P. Sloan Foundation, and Google Inc.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Visualization of cities and urban landscapes has been a theme in art for many centuries. The key problem in making these visualizations successful is summarizing in a single image the extended linear architectural fabric seen at eye level along a street, and doing so without introducing excessive distortions. In this thesis we address the problem of creating these visualizations easily and automatically. Possible applications include in-car navigation, augmentation to online route mapping applications, and web-based tourism information.

1.1 3D Visualizations

The Google Earth software [Goo06b] combines stitched, geo-referenced aerial and satellite imagery with terrain elevation information to allow the user to virtually fly over the earth. Screenshots are shown in Figure 1.1. By streaming the display data so that only the resolution necessary for the user's current view is transmitted, they can combine both a massive back-end database of imagery that spans the entire earth and still allow the user to seamlessly and interactively browse through the dataset. This geographic visualization is excellent for getting an overhead view of places; however, because the terrain information is coarse and the imagery is only from overhead views, when the user zooms into a particular location, the visualization does not scale well and buildings and urban structures are flattened out. Recent updates have allowed 3D models of buildings to be inserted into the visualization, however they are inserted over the 2D textured-mapped ground that still has the aerial views mapped onto it, creating a confusing hybrid where the image data does not match the 3D data.

The Berkeley Urbanscan [FZ04] project approached this problem in a different way. They attempted to directly create 3D textured models of an urban environment. Once these models are created, they can be viewed through an interface similar to the Google Earth software. These models were created by combining data from multiple sources. One source was aerial imagery along with aerial LIDAR that provides high-resolution 3D information about the scene at a much higher resolution than the terrain data included in Google Earth. Once again, this allows only overhead views. To augment that information, they additionally captured video and LIDAR data from a vehicle traveling through the urban area. From this data, they extracted texture and geometry information that is subsequently registered and combined with the aerial views. From all of this data, they directly reconstruct textured 3D models of the buildings and the ground.

There are a number of drawbacks to this approach that encouraged us to investigate other representations. In particular, the process of automatically extracting high quality 3D geometry and texture is still an open research area, one in which the Urbanscan project made substantial progress. A particularly difficult problem is extracting the texture for a part of a building that is never seen by the video. For example, part of the wall that is hidden by a tree on the sidewalk. Often in these cases, a texture synthesis approach is taken, literally making up a plausible stand-in. Another issues that commonly arises is the level of detail for the extracted geometry. The real world has detail far beyond what is possible to extract, and so reconstructing the leaves on trees (for example) interferes with both the tree's reconstruction and the reconstruction of anything behind it.

1.2 2D Visualizations

One possible approach to depicting the eye level urban fabric using 2D images is using wide angle or omnidirectional views around a single viewpoint, typically captured at street corners. Omnidirectional cameras [Nay97] provide a possible optical solution for capturing such views. Photo-mosaicing (the alignment and blending of multiple overlapping

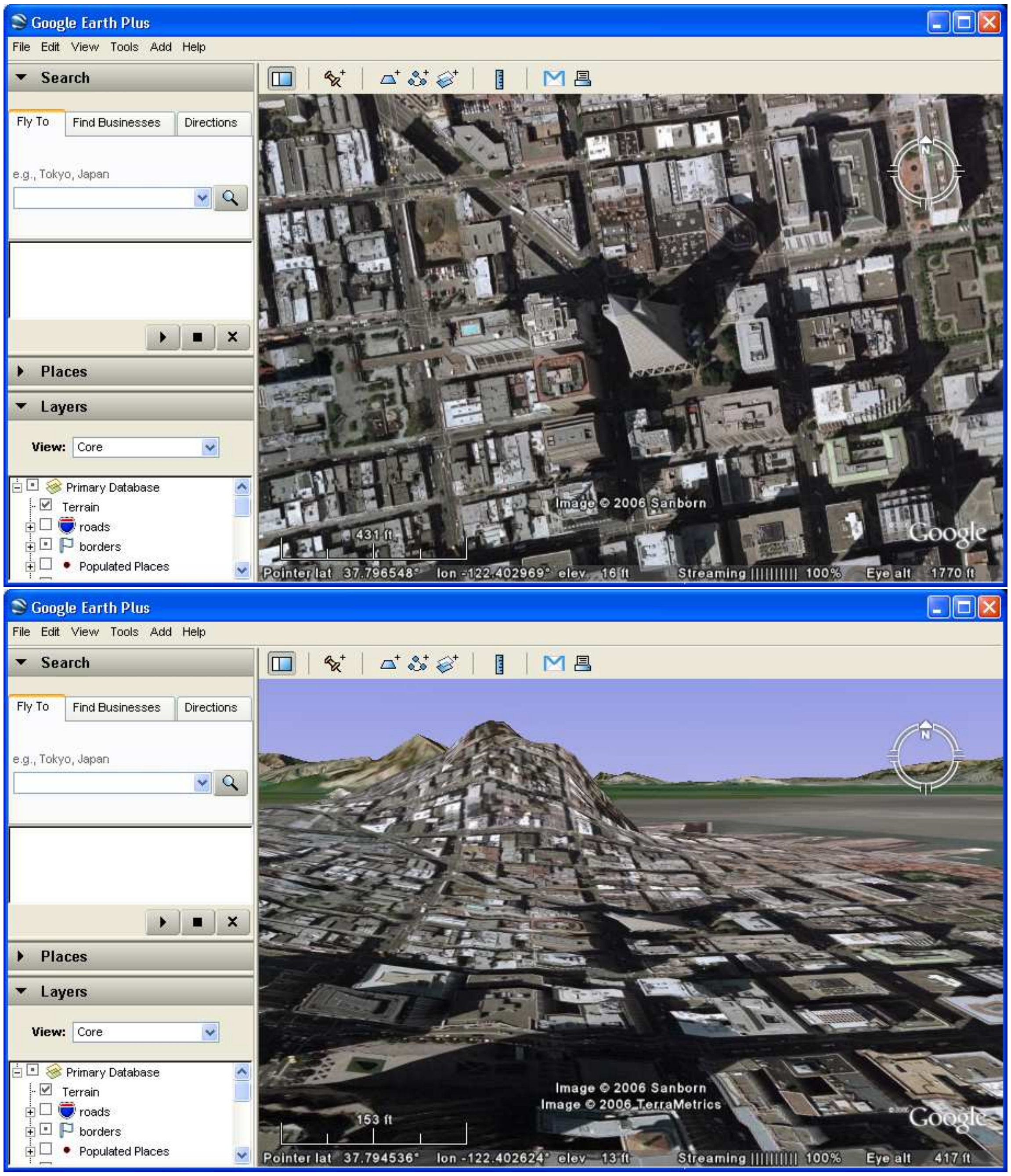


Figure 1.1: Screenshots of the Google Earth software. The top image shows a top view of downtown San Francisco created from geo-referenced aerial imagery. Because this viewpoint closely approximates the actual aerial view, the images look natural. The bottom image shows the same region viewed from an angle. The flattened out buildings look unnatural when viewed from the side.

photographs) is an alternative approach for creating wide field of view images. These mosaics can be made by capturing a part of the scene surrounding a single point by panning a camera around its optical center [Che95, SS00]. Such omnidirectional views, however, are still perspective projections and therefore, objects at any considerable distance from the camera become too small to be recognizable.

A set of georeferenced 360-degree panoramas (such as QuickTime VR images) is a possible urban visualization. The Virtually-Vancouver website [Van06] is an example of this – they have a 360-degree panorama at every street intersection within the targeted region. This allows a potential visitor to visit every street corner and look all around, almost as if they were standing there! Unfortunately, however, the visitor is restricted to visiting only street corners, or in the more general sense, only the locations for which the 360 panorama is available. Even if 360 panoramas were available at any desirable location, there is still the issue of transmitting the image information. Because of parallax, the entire image can potentially change from one location to the next, and this essentially becomes equivalently to transmitting an entire video stream to allow the user to move around in the environment.

Similarly, a number of 2D photographs can be taken along streets and, if they are georeferenced, can allow a potential visitor to virtually walk along the street by moving from photograph to photograph. Unfortunately, there are again the two issues of sampling density (the spacing between photographs) and the bandwidth (amount of information that needs to be sent for every new photograph). The lower the sampling density, the more disconnected each photo will be from the next, requiring a higher cognitive load on the user to keep track of the location. On the other hand, increasing the sampling density so that the user feels as though they are watching a video moving down the street again significantly increases the required bandwidth.

Most images we are accustomed to viewing are formed via perspective projection, that is, they correspond to rays passing through a single center of projection. While perspective projection is an accurate model of image formation on the retina and on camera films/sensors, it has some limitations.

Multiperspective images, offer a promising alternative because they are not bound by these constraints. A multiperspective image is a 2D image where every region can have a

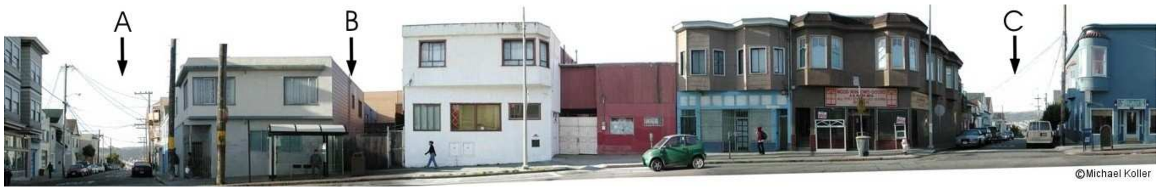


Figure 1.2: A hand-crafted multiperspective image excerpted from Michael Koller's Seamless City. Notice the distinct vanishing points at A and C, even though these two streets are parallel, and the distortions in the depiction of building B. This image was constructed by aligning and inter-cutting several ordinary perspective images. Artful placement of the cuts between images yields a composite image without evident seams. However, this process undoubtedly requires great care and labor. This thesis introduces a novel user interface for semi-automatically constructing images like this one.

different center of projection. An example of this can be seen in Figure 5.5, which shows a small portion of a continuous multiperspective image that spans approximately 2000m along the street. Multiperspective images retain the simplicity of a single image, avoiding the space, time, and viewing complexities of videos or collections of individual images. They can efficiently summarize extended scenes.

An example of a high quality multiperspective image can be found in the work of artist Michael Koller [Kol04]. An excerpt from his work is shown in Figure 1.2. Koller synthesizes a continuous visual image of the city of San Francisco made by sequential photos of a walk through the city. In a manual process he aligns, cuts and pastes these images next to each other to form a final image. By making his cuts follow architectural features in the scene, he produces perspectives that look correct along the alleys and street intersections. The resulting photomosaic is a high-quality, high-resolution continuous single image that effectively summarizes a large geographic area without objectionable distortions.

One of the major disadvantages of the Koller's approach is that the process is manual and time-consuming. Creating a continuous visual representation of a 30-mile route through San Francisco has taken several years of manually aligning and stitching images. Our work can be thought of as an attempt to automate his approach.

A possible approach to automatically create similar images is to use pushbroom [HG94, PRRAZ00] or crossed-slits imaging [ZFPW03a, ZFPW03b]. A pushbroom image is defined as an image that is perspective in one direction (e.g., vertically) and orthographic in the other while a crossed-slits image is an image which is perspective in one direction but is perspective from a different location in the other direction. The perspective structure of crossed-slits cameras are the set of all rays intersecting two fixed lines (slits) in space. For pushbroom cameras, one of the slits is at infinity. In both cases, one is free to select the placement of the slits. Changing these placements strongly affects the visualization and the associated distortions as we show later in our results. In the context of visualizing eye level urban landscapes, we show that we can combine multiple crossed-slits images seamlessly to reduce distortions.

A common problem inherent to these linear multiperspective projections is the distortion introduced when the perspective in the horizontal and vertical dimensions are not the same. Depending on the depth variation in the scene these distortions can be severe and make portions of the image unusable (see Figure 5.6(a)). These distortions can be avoided to some extend by carefully adjusting the perspective for each image region. In Chapter 3, we present a framework which allows an artist to *manually* specify different perspectives for individual scene segments and to compute smooth transitions between them. Unfortunately, as in other approaches, the selection of the perspectives is done manually and thus tedious, error-prone, and cannot scale to larger datasets. We have therefore additionally developed an optimization framework that can automatically define the structure of a multiperspective image that minimizes the output distortion, described in Chapter 5.

1.3 Computer generated multiperspective images

More recently there has been an interest in computer generated multiperspective imaging. The synthesis of multiperspective images has been explored in Wood et al. [WFH$^+$97] and Rademacher and Bishop [RB98]. Wood et al create a multiperspective image where the perspective varies slowly across the image. This has the effect that when a cropped region of the image is viewed, it appears to be entirely perspective. However, as the crop is moved across the image, it appears as though the viewpoint is shifting. With a similar goal, Rademacher et al create a multiperspective image by extracting the central column from a video sequence of camera swept around a 3D model and abutting these columns together. When viewed directly, this image has little meaning to a human. Together with the knowledge of the original camera trajectory, however, they use the multiperspective image to reconstruct perspective views of the original model.

Glassner [Gla00] explores the use of multiperspective imaging as an effective tool for illustration or story telling. He includes a plugin for a 3D modeling program that allows creating a multiperspective image through a two-surface parameterization. Rather than the two-surface parameterization, we use use crossed-slits images as the fundamental modeling primitives for creating multiperspective images. We have also found it important to be able to specify the parameterization of the picture surface. Vallance and Calder [VC01] provide an in-depth analysis of the previous literature in multiperspective imaging. They also describe an API to facilitate rendering of multiperspective images.

1.3.1 Camera Models

In addition to making multiperspective imaging practical, there has also been much theoretical work on multiperspective imaging. Gupta and Hartley [GH97] derive a projection model for pushbroom cameras. Zomet et al. [ZFPW03a] extend this in their work to model crossed-slits cameras. More recently, Yu and McMillan [YM04] provide a General Linear Camera (GLC) model, which unifies the perspective, pushbroom and crossed-slits cameras along with five other camera models under one framework. Although GLC's encompass eight cameras, we currently restrict our system to these three, which seem most useful for our task. Specifically, the subset of GLCs we allow is that which can be created from a camera traveling in a path, since the camera path naturally defines one slit of a crossed-slits camera.

1.4 Applications

Multi-perspective images are nothing new in the art world. 10th century Chinese paintings used multiple perspectives to depict many religious sites in a single image without noticeable distortions [CT01]. More recently, the work of the cubists and M. C. Escher explored combining multiple perspectives.

Artists frequently distort perspectives to emphasize features in the scene, in particular in maps or other urban depictions. Prominent buildings or landmarks are drawn disproportionately large, while minor streets or alleyways are sometimes suppressed alltogether. Distances are often adjusted to make more efficient use of space.

Much of the early work in computer-generated multiperspective imaging used synthetic 3D data for illustrating results. More recently, multiperspective images have been used for visualizing scenes that are large compared to the maximum available field of view of any individual input image. In particular, stitching several perspective photos together to form a single multiperspective image that describes the scene. There are many possible applicable scenarios:

Digital Route Panoramas [Zhe03] was one of the first examples of using a particular type of multiperspective images called *pushbroom panoramas* (described in detail in Section 2.1.2). Zheng generates route panoramas from a moving video by taking the central column of pixels from each frame and abutting them together. In later work [Zhe04], he demonstrated that less objectionable distortions can be obtained by selecting non-central columns for the pushbroom projection. His choice of column is fixed for an entire acquisition, however, and does not allow the flexibility of choosing columns based on scene content as we do.

Seitz and Kim [SK03] investigate how to generate multiperspective images from a moving video camera. They treat the captured video as a stack of frames forming a 3D volume and then allow arbitrary 2D slices through this volume. While this method allows generation of almost any multiperspective image that is possible given the video volume, it is not clear what perspectives the resulting images represent except in special cases. For example, a slice through the volume parallel to the first frame (essentially extracting a frame from the volume) is a perspective image (Section 2.1.1), a slice straight down the volume is a pushbroom image (Section 2.1.2), and a diagonal slice is a crossed-slits image (Section 2.1.3). A non-linear slice through the volume will create a multiperspective image such as we create in this thesis. However, it is difficult to associate any general non-linear slice with its perspective structure in 3D. This in turn makes it hard to design a slice to accomplish a particular task, such as displaying city blocks with their varying facade depths.

Another possible approach is that described by Agarwala et al [AAC+06]. Using a semi-automatic approach, they construct a multiperspective image by reprojecting input camera images onto a single picture surface and extracting arbitrary regions that have minimum distortion along the seams. This approach obtains results similar to those of Koller provided the seam is sufficiently planar.

There are many other possible uses for multiperspective images of extended, roughly planar scenes. For example, one can imagine generating an image that spans rows of books in a library to allow virtual browsing of books online. Similarly, warehouses could quickly inventory items by creating multiperspective images of the storage, allowing workers to quickly and visually identify relevant sections of the warehouse.

It could also be useful for artistic purposes. For example, an entire walking route through a museum could be assembled as a single continuous multiperspective image, similar to the scene shown in Figure 5.10.

Another interesting artistic example is visualizing coral reefs. One could imagine a single, continuous image spanning the Great Barrier Reef in Australia. In this case, imaging a scene underwater introduces an additional limitation of visibility. Even if it were possible to obtain an unoccluded view of the entire reef, it would be much too far away to see anything through the water. Instead, by synthesizing an image from video frames taken very close to the coral, a vivid and colorful panoramic image can be obtained. A similar argument follows for obtaining images of shipwrecks that allow the wreck to be viewed as if the viewer were thousands of feet away from the wreck itself but with the visibility as if the viewer were only a few feet away.

1.5 Our contributions

There are three primary contributions of this research:

1. We demonstrate that multiperspective images can be created by abutting multiple crossed-slits images and we describe the constraints necessary to seamlessly blend between adjacent crossed-slits images. This is described in Chapter 4.

2. We describe an interactive user interface that allows a user to easily create a multiperspective image comprised of abutted crossed-slits images. This software automatically enforces our constraints and helps the user gain an intuition of how the

parameters of the individual crossed-slits images affect the final multiperspective image. This software is described in Chapter 3.

3. We describe an optimization framework that can automatically determine the parameters for a set of crossed-slits images that minimize the distortions in the final output image, described in Chapter 5.

Chapter 2

Multiperspective Imaging Paradigm

Multiperspective images are considered in this thesis as a potentially nonlinear projection from 3D to 2D. This mapping is used to sample from an input dataset of 2D perspective images directly to the final 2D multiperspective image (see Chapter 4).

Although the final multiperspective image may be nonlinear, it is constructed from a set of linear projections. These linear projections are described in Section 2.1. Constraints on combining these are described in Sections 2.3 and 2.2.

2.1 Camera Models

The relationship between the 3D world and the image of a projection of the world is described by a camera model. Two common camera models are the perspective and orthographic projections. These are both used extensively in computer graphics, in particular in 3D rendering and computer graphics; for example, the OpenGL graphics API [Ope92] allows specifying only perspective or orthographic projections.

2.1.1 Perspective

The perspective projection camera model is a 3D to 2D projection based on the camera obscura and is also known as the *pinhole projection*. In this case, an image is formed by projecting all 3D points onto a 2D image plane through a single point in space called the *center of projection.* In reality, rays pass through a small, finite region of space (the aperture) that is approximated as a single point. A detailed discussion of the perspective projection camera model can be found in [HZ04].

Traditionally, the projection is defined for a camera with a center of projection at the origin and oriented facing along the $-z$ axis. In addition, the image plane is typically assumed to lie on the plane $z = -1$.

This 3D-2D projection is typically using homogeneous coordinates with

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.1}$$

or

$$\mathbf{x}_{\text{2D}} = \mathbf{P}\,\mathbf{x}_{\text{3D}} \tag{2.2}$$

where $\mathbf{x}_{\text{2D}}$ is a 3×1 homogeneous 2D point, $\mathbf{x}_{\text{3D}}$ is a 4×1 homogeneous 3D point, and $\mathbf{P}$ is a 3×4 projection matrix.

Note that for a homogeneous point, we have

$$\mathbf{x}_{\text{2D}} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \tag{2.3}$$

The projection matrix is composed of both the internal camera parameters (those that depend only on the camera, for example the lens focal length and the image plane resolution) and the external camera parameters (those that aren't affected by the particular camera, for example the position and orientation). These are referred to as the intrinsics and extrinsics respectively. The intrinsic parameters (in units of pixels) are

f_x, f_y	The focal length of the camera along the x and y directions
c_x, c_y	The optical center of the 2D projection plane
θ	Pixel skew (typically zero and ignored in this discussion)

and the extrinsics are

$\mathbf{t}$	The position of the camera in the world coordinate system
	Described by a 3×1 position vector
$\mathbf{R}$	The orientation of the camera in the world coordinate system
	Described by a 3×3 rotation matrix

The intrinsics are gathered the matrix **K** as

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Using these parameters, the projection can be described with

$$\mathbf{x}_{\text{2D}} = \mathbf{K}\left[\mathbf{R}|\mathbf{t}\right]\mathbf{x}_{\text{3D}} \tag{2.5}$$

or equivalently

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc|c} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \end{array}\right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.6}$$

Multiperspective images are any image in which the image rays do not all pass through a single center of projection, making the space of all possible multiperspective images very large. In this thesis, we instead limit ourselves to multiperspective projections that can be described linearly. In particular, the pushbroom and crossed-slits projections.

2.1.2 Pushbroom

The linear pushbroom camera is a non-perspective projection mathematically described in [GH97]. This projection is a combination of pushbroom and orthographic projections. In this thesis, it describes an image that is perspective vertically and orthographic horizontally.

Consider the case where a video camera is facing down the z-axis and translating along the x-axis at a constant velocity, as shown in Figure 2.1(top). If the central column from each video frame is extracted and abutted into a single image, the result will be a pushbroom image. Notice that because each column of the final output image is taken from a single input video frame, each column of pixels is perspective. However, because each column of pixels is taken from a camera image at a different location, neighboring columns do not share a single center of projection. In fact, because each column of the output pushbroom image is taken from the same column from each input video frame, they are parallel. That is, projection rays for any pixels in different columns of the final output image are parallel. The image, therefore is vertically perspective and horizontally orthographic.

2.1.3 Crossed-slits

The crossed-slits camera is a non-perspective linear projection and is described in detail in [ZFPW03a]. This projection is a superset of the perspective and pushbroom projections. It describes both of those projections and can also smoothly transition between the two.

Consider the case of a translating video camera as described above for pushbrooms. Instead of taking the center column out of each frame, instead take the left-most column of the first frame and the right-most column of the last frame, as depicted in Figure 2.2. By linearly varying the column that is taken according to the camera position, the result is a crossed-slits projection.

All of the rays in this crossed-slits projection can be described by the set of rays intersecting two lines in space: the camera path and a second, perpendicular line. Intuitively, it makes sense that all of the rays in this resulting image must pass through the camera path.

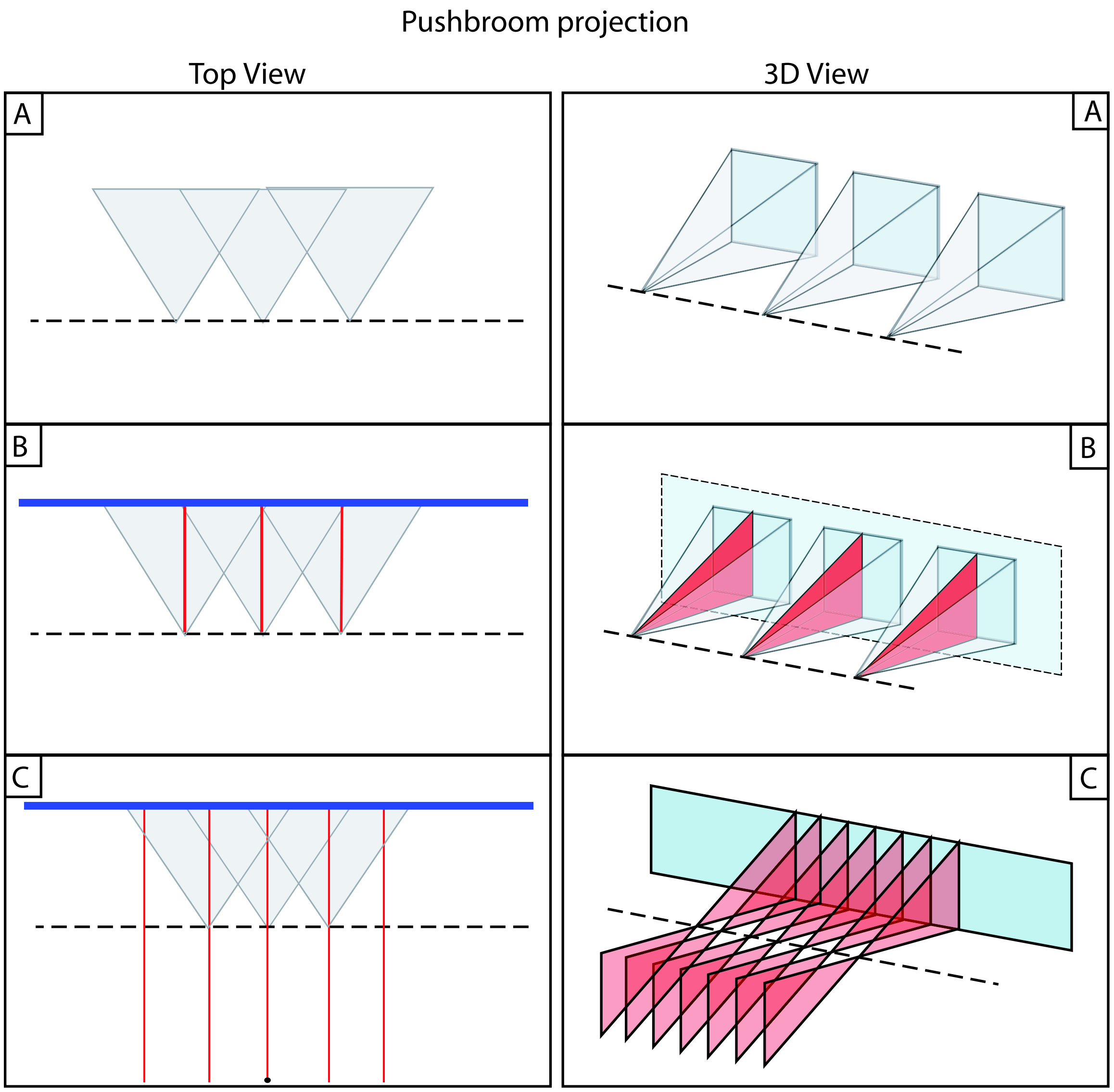


Figure 2.1: Example of the acquisition and construction of a pushbroom image. The right images show the 3D setup of a translating camera oriented along the *z*-axis and moving along the *x*-axis at a constant velocity while the left images show a top view. (A) shows several example camera frames in blue along the camera's trajectory (dotted black line). (B) A picture surface is specified (dark blue on the left, the large plane on the right) and parallel columns from each camera frame (shown in red) are indicated. (C) By extracting parallel columns out of each input image and abutting them together, we create a pushbroom image.

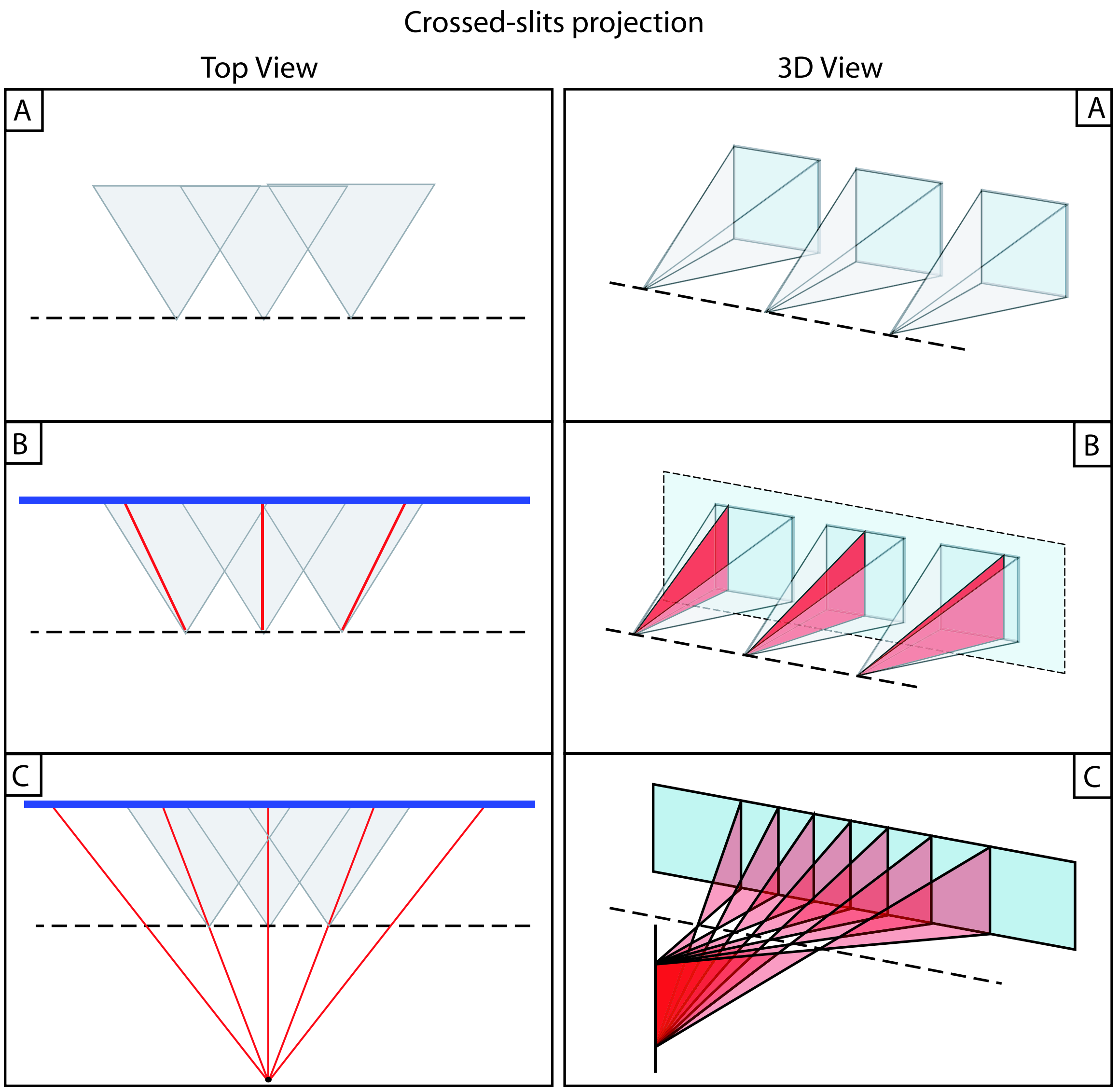


Figure 2.2: Example of the acquisition and construction of a crossed-slits image. The right images show the 3D setup of a translating camera oriented along the *z*-axis and moving along the *x*-axis at a constant velocity while the left images show a top view. (A) shows several example camera frames in blue along the camera's trajectory (dotted black line). (B) A picture surface is specified (dark blue on the left, the large plane on the right) and columns from each camera frame (shown in red) are indicated. (C) By choosing the columns from each input frame so that the direction of rays intersect at a point independent of the camera trajectory, we create a crossed-slits image.

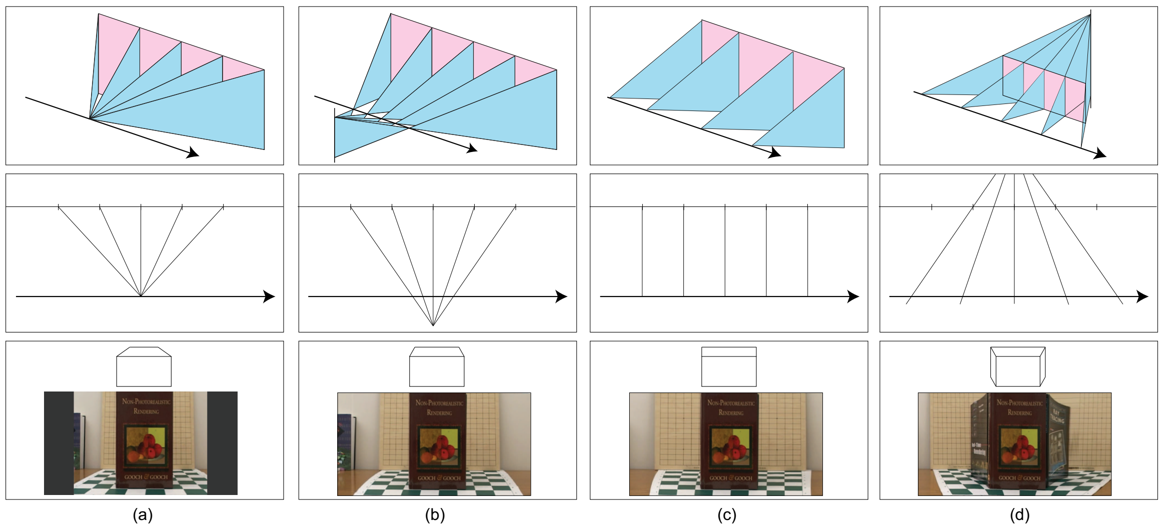


Figure 2.3: Effect of varying perspective in x and y. The top row shows a 3D view of the picture surface (in pink), the camera path and the ray manifold for different kinds of projections. The second row provides a birds-eye view and the last the resulting images when the picture surface is aligned with the front book cover. These are four types of perspectives our system can generate. For all examples the perspective in y is given by the input images. (a) A perspective projection simply corresponds to the original perspective of the camera. Notice the limited field of view. (b) A crossed-slits perspective can be used to extend the field of view in x by moving the horizontal center of projection off the camera path. (c) Moving the horizontal center of projection to infinity results in a pushbroom which is horizontally orthographic as can be seen in the checkerboard pattern. (d) In an inverse perspective the center of projection is behind the picture surface. The effect is that objects get larger with increasing distance and both sides of a cube are visible.

This is because the image is constructed by taking pixels from a set of perspective images. Each of those perspective images is a video frame corresponding to a location along the trajectory of the camera and therefore a corresponding center of projection. These centers of projection form the line of the camera path. The second, perpendicular line in space defines the intersection of the fans of rays described by each column extracted from the video frames. By requiring that these fans intersect in this second line, we define the direction of those rays and therefore the column from the input video frame.

This provides some interesting flexibility. All of the rays in a crossed-slits projection are defined by these two lines (or *slits*). Of course, the camera path is determined by the original trajectory of the input video camera, but we are free to choose the location of the second slit.

Consider the case in which this second slit is positioned so that it intersects the camera path. Since all of the rays of the resulting image must pass through both slits, then the only place that is satisfied is at the point of intersection. Since all of the rays are passing through a single point, the result is a perspective image.

Consider the case where the second slit is placed infinitely far away. In this case, the fans of rays corresponding to each column of the final image become parallel and we are left with a pushbroom image.

These effects of varying the slit location in a crossed-slits projection are illustrated in Figure 2.3. In general, the slits in crossed-slits cameras need not be perpendicular to each other although they are constrained to be so in our application.

Perspective, pushbroom, crossed-slits, and orthographic cameras are all subsets of a larger family of linear projection cameras called *general linear cameras*[YM04]. We restrict ourselves to using crossed-slits projections because that corresponds most naturally to the useful perspective variations from a translating camera.

2.2 Multiperspective Images

Multi-perspective images can be specified as a 2D manifold of rays and the mapping from this manifold to a rectangular, regularly sampled image as described in Figure 2.4. Note that contrary to perspective images, objects in a scene may appear more than once in a

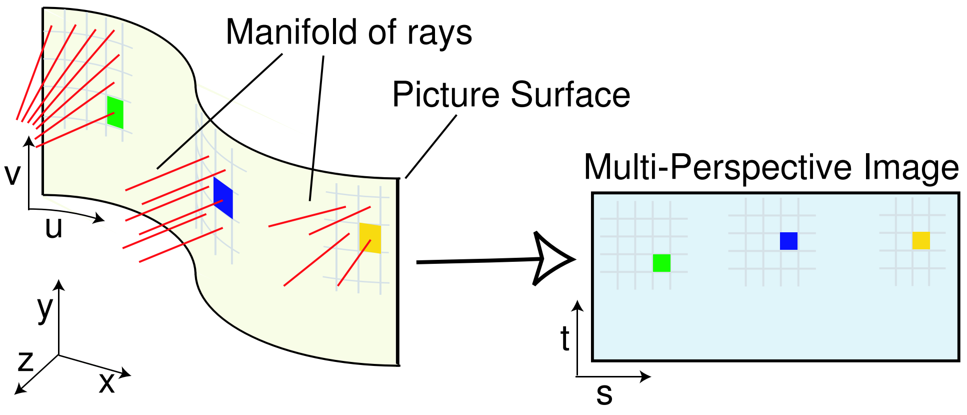


Figure 2.4: The ray manifold is the set of rays in space. The sampling of these rays and the mapping to the output image can be specified by a picture surface with a 2D parameterization of its 3D surface.

single multiperspective image because the direction of rays is less constrained. The user specifies the manifold of rays as a sequence of crossed-slits cameras in the 3D scene, and specifies the mapping to an image by placing a regularly sampled picture surface in the scene. We find that distinguishing clearly in our user interface between specifying the ray manifold and specifying the mapping to the output image improves the intuitiveness of our system.

The manifold of rays can be specified in several ways. For a pinhole camera, the manifold is the set of all rays passing through a point. In an orthographic camera, the manifold describes all rays in a single direction. Similarly, for a crossed-slits camera the manifold is described by all rays passing through two lines (or slits). These are three of the eight 2D linear manifolds described by Yu and McMillan [YM04].

For our application, we choose to constrain the allowable set of manifolds based on their applicability to urban landscapes and the ease of specification for the user. Specifically, we restrict the space of ray manifolds to crossed-slits images. Doing so allows us to include perspective and pushbroom images while at the same time enabling the interpolation scheme described in section 2.3. The result is that our final image can be represented as a mosaic of multiple crossed-slits images.

The picture surface defines the sampling of the manifold of rays as well as their mapping to the final image, as shown in Figure 4.1. Traditionally, in a single perspective image,

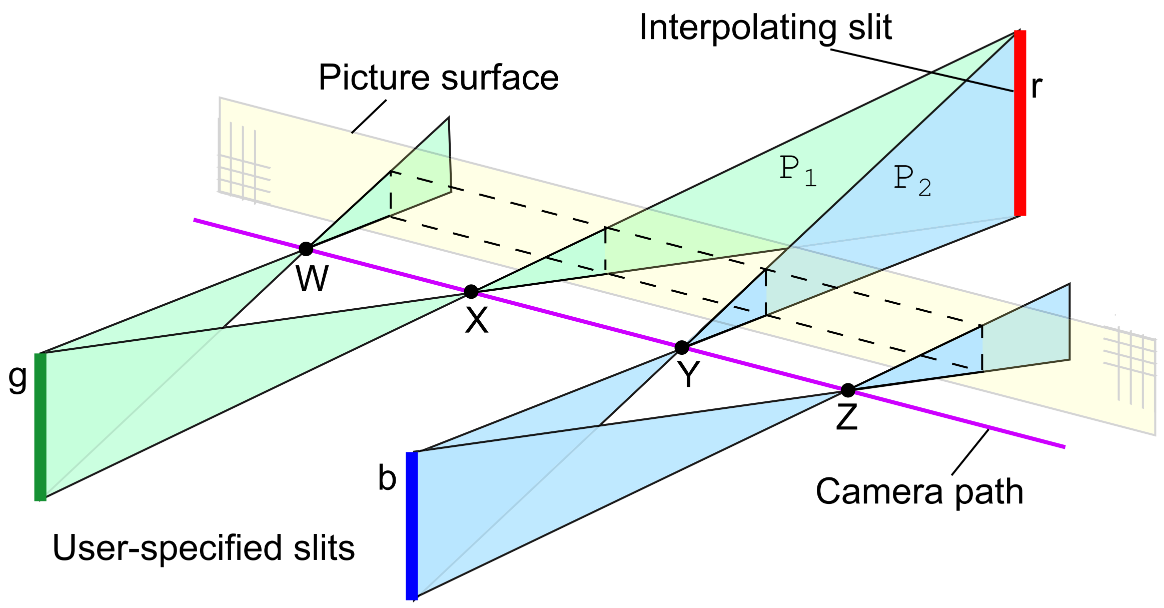


Figure 2.5: We visualize urban landscapes using a blend of adjacent crossed-slits images. The figure shows two user-specified crossed-slits cameras, represented by slit pairs WX-*g* and YZ-*b*. This partitions the camera path WXYZ into three sections. The plane P_1, formed by the slit *g* and point X, represents the rightmost column of pixels in crossed-slits camera WX-*g* and their associated ray directions. Similarly, P_2 is the plane formed by slit *b* and point Y. These two planes P_1 and P_2 intersect in line *r*, which becomes our interpolating slit. The XY-*r* crossed-slits pair becomes our interpolating camera. Note that the interpolating camera has the same ray directions on its edges as its neighboring cameras. This ensures that the generated image contains no discontinuities.

the picture surface is a plane. However, we can reduce distortion in the output image by allowing the picture surface to change orientation to accommodate the storefronts. In our application, we constrain the picture surface to be vertical with respect to the ground.

Lastly, every point on the picture surface must be associated with a ray from exactly one ray manifold. This ensures that the resulting final output image has no missing regions.

Examples of the types of multiperspective images that we allow in our images are shown in Figure 2.3.

2.3 Interpolating Two Crossed-Slits Images

Associating regions of the picture surface with crossed-slits cameras naturally leads to the problem of how to handle unassigned regions between them. Our system adds an additional crossed-slits camera between every two adjacent user-specified crossed-slits cameras, resulting in smooth interpolation. The process of computing the location of the interpolating slit is described in Figure 2.3. As shown in the figure, the location of the interpolating slit is uniquely defined by the geometry of the adjacent slits.

It should be clear from the figure that the insertion of these interpolating slits is independent of the picture surface chosen by the user. Also, the camera path must be continuous but need not be straight—the interpolating slit will depend only on the points X and Y along the camera path. Note that this interpolation scheme is not dependent on using the camera path as one of the slits. In fact, this scheme will correctly interpolate any pair of crossed-slits cameras provided that the slit joining them is continuous.

Chapter 3

Interactive Specification of Multiperspective Images

While most people generally have a good idea of what a photograph will look like when shown a diagram of the camera position and orientation relative to a scene, this intuition does not exist for multiperspective images. The choice of 2D manifold of rays, the placement of the picture surface and the sampling of the surface constitute a design problem. We have designed a user interface which helps develop an intuition for the perspective structure of multiperspective images as well as generates effective visualizations of urban landscapes.

We describe an interactive system that can be used for constructing multiperspective images from sideways-looking video captured from a moving vehicle. The input to our system is a set of video frames with known camera pose. The interface then provides a set of tools that allow the user to define the picture surface and place crossed-slits cameras. Our system then automatically computes an additional crossed-slits camera between every pair of adjacent user-specified cameras leading to a smooth interpolation of viewpoint in the final multiperspective image. Our system provides the tools necessary to minimize distortions and discontinuities for creating good multiperspective images for urban landscapes. Using our system, a person can create a multiperspective image of a whole city block in a few minutes. The process is summarized in Figure 3.1.

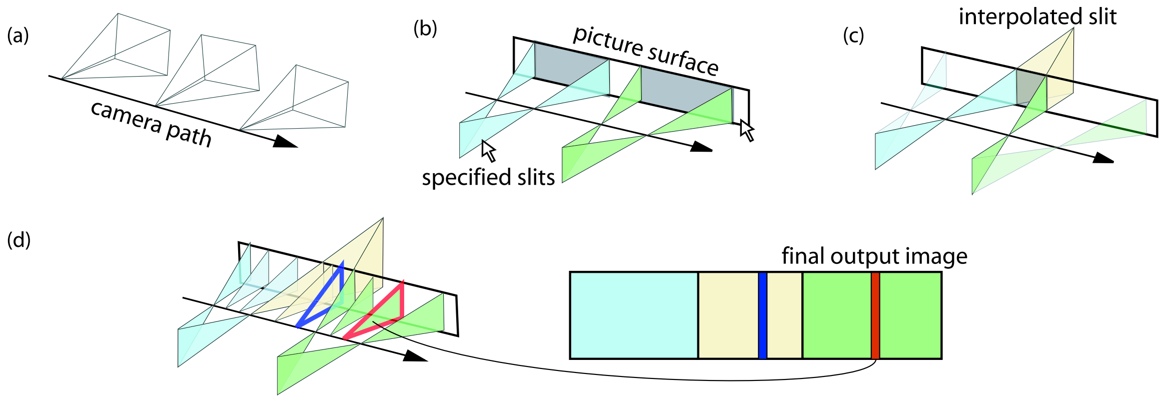


Figure 3.1: This figure summarizes our algorithm for generating multiperspective images. (a) First, we process each input video frame to estimate the corresponding position and orientation of the camera. (b) Second, the user specifies the picture surface and any number of crossed-slits camera locations (the green and blue regions), thereby defining valid region (gray shading) on the picture surface. (c) For the remaining regions (gray shaded), we automatically compute the interpolating crossed-slits camera (yellow). (d) Within each camera, each planar fan of rays (blue or green triangles) denote one line of pixels (typically vertical) in the final output image. To produce this image, these pixels must be extracted from the appropriate frame of video, as described in section 4.4.

In this chapter, we describe our system for interactive design and rendering of multiperspective images. Our system takes a set of video frames (captured with a sideways-looking video camera) with known camera pose as input and produces one composite multiperspective image as output. The system consists of a user interface and uses the rendering engine described in Chapter 4.

3.1 Design Choices

There are a number of restrictions that we place on our system. We assume that the camera path in the input video lies on a plane parallel to the ground plane. For each crossed-slits image, the user specifies one slit, and the camera path is assumed to be the second slit. We assume that the user-specified slit is vertical (perpendicular to the ground plane). This allows a simplified plan view to be used in our user interface where the vertical slits are projected as points.

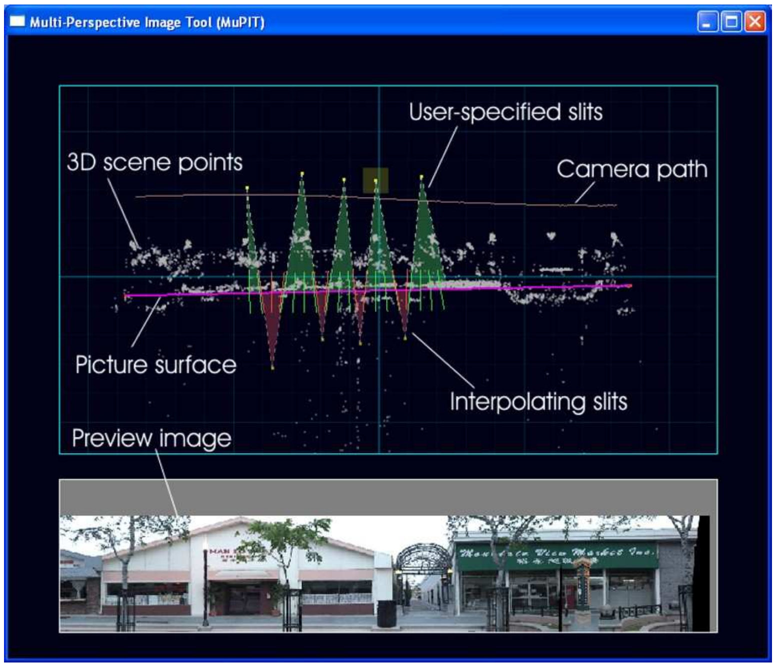


Figure 3.2: This is a snapshot of our interface. The diagram at top is a plan view of the scene. You can see the partial 3D scene structure, picture surface, camera path, user-specified slits, interpolating slits and a low-resolution preview image.

As shown in Figure 4.1, the picture surface is a parametric surface in 3D. We choose our picture surface to have vertical sides. The restriction of vertical sides on the picture surface applies naturally to urban facades on flat terrain. To aid the user in specifying non-vertical sides of the picture surface, we provide piecewise-continuous lines and quadratic splines. Quadratic splines are approximated as piecewise-linear segments to simplify rendering. This allows our surface to be represented as a series of planar facets. We constrain the sampling of the picture surface to be regular.

3.2 User Interface

Shown in Figure 3.2, our interface provides both a design section in which the user specifies the multiperspective image and a preview section that can provide rapid, low-resolution previews of the final image. Once satisfied with the design, the system can output the full resolution image.

To create a multiperspective image, the user must define the picture surface, place all desired crossed-slits cameras, and associate the cameras with regions of the picture surface. Because of the lack of natural intuition concerning these types of images, the interface strives to present the design in terms of familiar concepts. With this in mind, the user is shown the camera trajectory in plan view along with any estimated scene structure in the form of a point cloud as output by our structure-from-motion algorithm. The camera trajectory, as explained in section 3.1, defines one of the two slits required for any crossed-slits image.

To define the picture surface in our interface, the user needs to only draw a set of connected line segments in plan view. This is possible because we restrict the picture surface to be vertical. To help fit the picture surface to curved facades, segments of the picture surface can also be toggled between straight lines and quadratic splines.

In plan view, the task of positioning user-specified slits involves simply placing the slits as points and specifying their field of view. The intersection of the field of view with the picture surface defines the region of the picture surface associated with that slit. If any segment of the picture surface is associated with more than one user-specified slit, such as if two fields of view overlap, there no longer exists a unique ray direction for points in that segment, and therefore that is not a valid specification for a multiperspective image. As long as the fields of view do not overlap on the picture surface, however, any number of user slits may be described. The specified camera slits can also be toggled between slits located at finite positions and slits located at infinity. Slits at infinity are represented by a directional line next to the selected point. Placing a slit at infinity produces a pushbroom image. Similarly, placing the slit directly on the camera path (thus intersecting both slits) produces an ordinary perspective image.

Once any valid multiperspective image is specified, the interface immediately shows a set of example ray directions at several points along the picture surface. The program also automatically displays the interpolating crossed-slits camera between any two adjacent user-specified cameras as explained in section 3.2.

3.3 Results

We tested our system on several videos of city blocks taken under different challenging scenarios. The input data for this system is described in Section 4.1 and the rendering algorithm described in Chapter 4. We first examine a scene with a relatively flat facade and straight camera path. Figure 3.3(a) shows a short section of a pushbroom representation of this scene. A pushbroom image is perspective in one direction (vertical in our case) and orthographic in the other direction (horizontal). Under this projection, only objects lying on the picture surface can be rendered faithfully. In our example, the picture surface is placed at the store facade. As expected with such a parameterization, trees (which are closer than the facade) are horizontally compressed while the view down the alleyway (which is farther) is expanded. By interactively manipulating the perspective structure of the image, we can reduce these distortions as shown in Figure 3.3(b). Using our user interface, we achieve this by specifying ordinary perspective cameras near regions of significant depth variation such as the trees and the alleyway. This creates an image that is more natural looking in these areas. To keep the image continuous, the system inserts interpolating cameras between the user-specified cameras.

Another example that illustrates the benefit of manipulating ray directions based on scene geometry is shown in Figure 3.4. The scene consists of a building with flat facade on the left and a very deep plaza on the right. By choosing multiple crossed-slits as shown in Figure 3.4(b) we can get a more recognizable image than Figure 3.4(a).

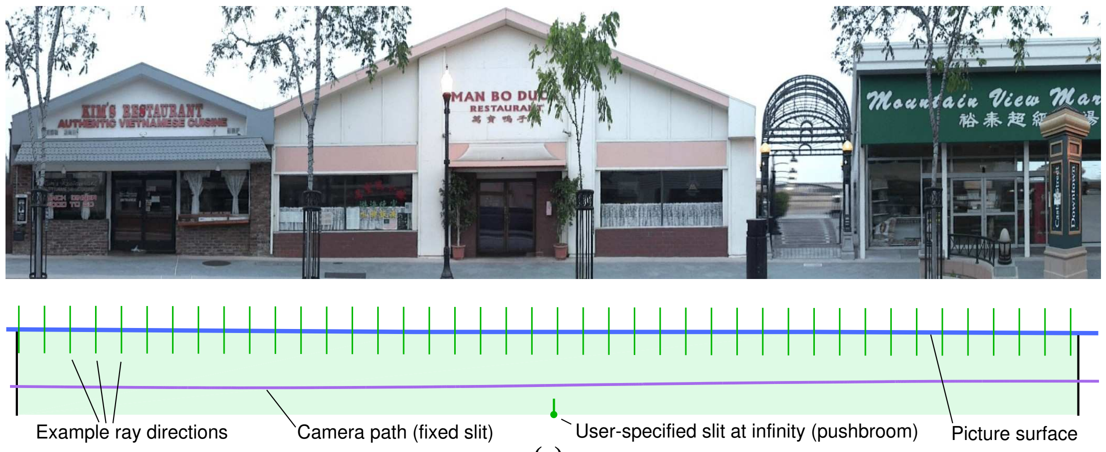
The ability to specify curved picture surfaces allows us to conform the picture surface to the natural architecture of a corner. An example of this is shown in Figure 3.5(a). This type of image is impossible to create with a traditional single, planar picture surface.

Although this allows us a summary view of both sides of the corner, the apparent size of facade is constant throughout the image. We can more naturally represent what a motorist or pedestrian would see if we stretch the image near the corner, producing a pinching effect as shown in Figure 3.5(b). This is a nonlinear effect and would require sampling of the picture surface as shown in Figure 3.6(c). Instead, we achieve this effect by curving the picture surface as shown in Figure 3.5(d) and adjusting the horizontal sample density as shown in Figure 3.6(d) by using the control polygon. This is a departure from the earlier imposed regular sampling on the picture surface. This sampling adjustment is performed automatically by our program and is explained in Figure 3.3.

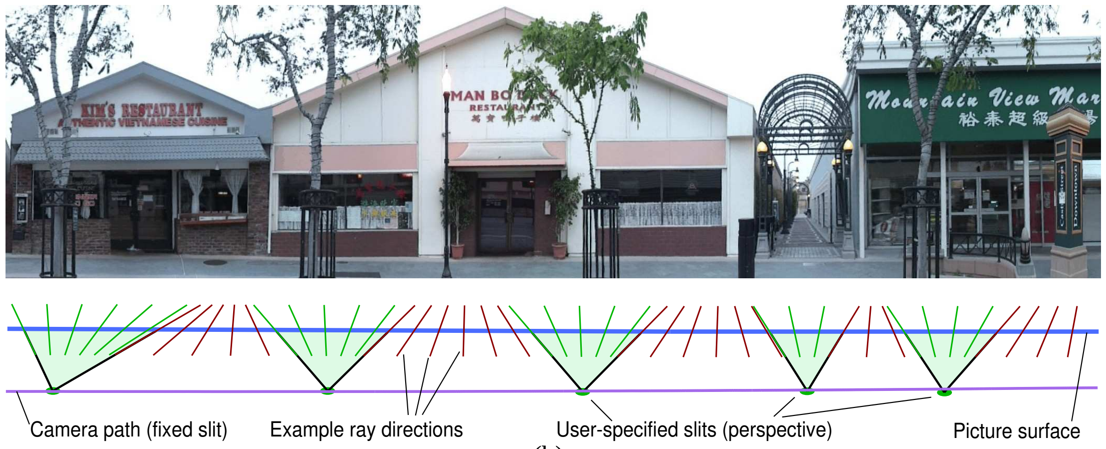
3.4 Discussion

There are infinitely many possible multiperspective images that visualize a city block. Our tool helps manage the creation of such an image in three ways. First, by restricting the space of multiperspective images to those most likely to be useful for visualizing urban landscapes, we reduce the number variables involved in creating these images. Second, our tool abstracts the away the ray manifold to a set of more intuitive *virtual viewpoints*. Finally, by providing rapid feedback to the user, our tool helps develop an intuition about the effects of the design parameters on the resulting image. This last point is key to the effectiveness of our tool and is enabled only by the simple rendering system described in Chapter 4.

One limitation of our system is that we allow only regular sampling on the picture surface (and the automatic adjustment of horizontal sampling density illustrated in Figure 3.3). We do not allow any user-specified sampling strategies. One can imagine sampling the image more densely in the center than toward the edges, resulting in a fish eye like effect. Also, the enforcement of vertical slit orientation, a limitation imposed by our user interface, implies that the user interface cannot accurately depict changes in elevation such as hills. Similarly, the user interface does not permit altering the orientation of the picture plane from vertical. Finally, we choose to allow representing only three of the eight GLCs from Yu and McMillan [YM04]. It would be interesting to incorporate these other cameras into our design tool.

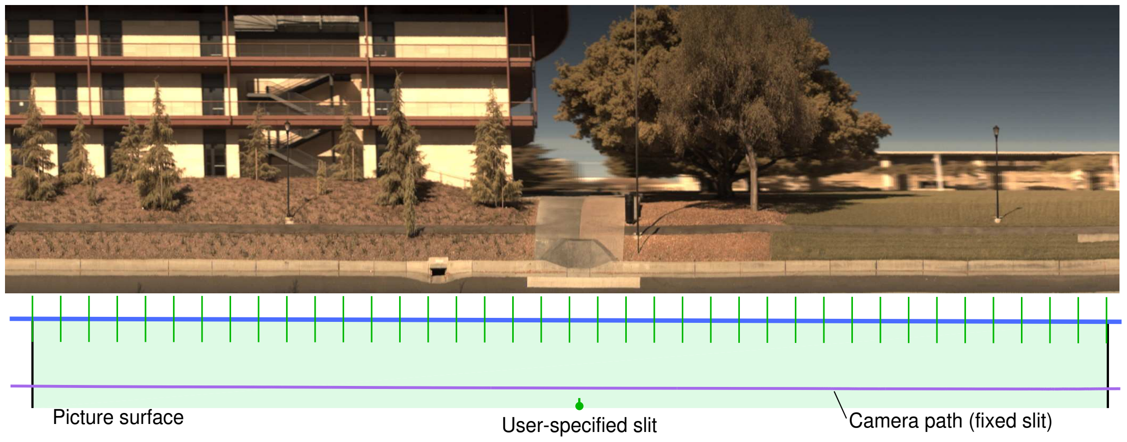


(a)

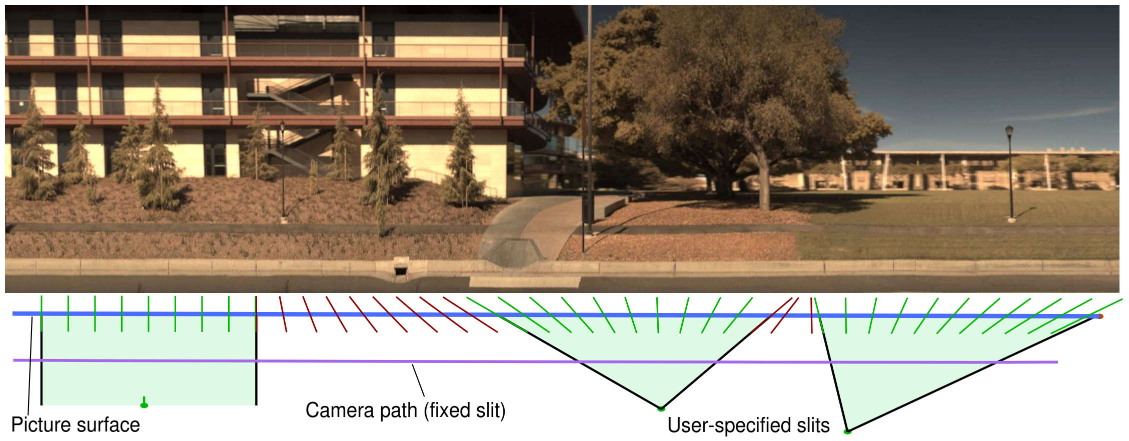


(b)

Figure 3.3: This example shows how manipulation of the perspective structure of the image can be used to generate a multi-perspective image with reduced distortion. The diagram below each is a plan view of the scene, with the input video camera moving along the indicated path and looking upward. The picture surface in both (a) and (b) is fixed at the facade of the storefronts. (a) is a traditional pushbroom image generated by specifying a vertical slit at infinity with all the ray directions being parallel. The resulting image has the familiar distortions associated with a pushbroom image: objects in front of the picture surface (e.g. trees) are compressed horizontally, and objects behind the picture surface are expanded (the view down the alleyway). (b) has been generated using multiple crossed-slits. By placing selected user-specified slits atop the camera path, ordinary perspective views are generated in the vicinity of the trees and alleyway. This enhances the realism of the image while still maintaining the extended field of view of the pushbroom. This does not come for free, however. Notice that the sidewalk in the center of the image appears curved and that one of the columns of the distant building appears tilted. These are artifacts induced by the changing perspective structure across the image.

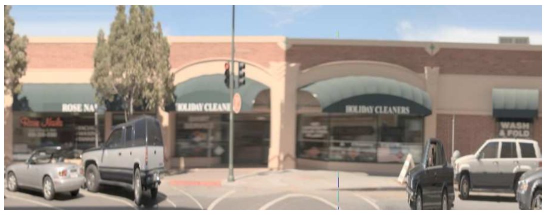


(a)

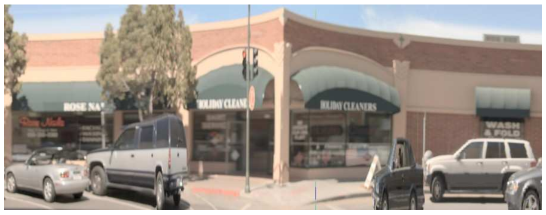


(b)

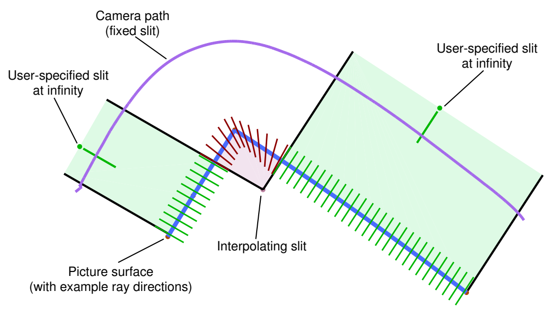
Figure 3.4: This example shows how our system can be used to generate effective multi-perspective images of deep plazas. The picture surface in both (a) and (b) is fixed at the facade of the building on the left. (a) is a traditional pushbroom image generated by specifying a vertical slit at infinity. This causes the deep plaza on the right to stretch horizontally, leading to apparent smearing due to limited resolution. In (b), by blending between two almost perspective views on the right (one for the tree and one for the building) and the same pushbroom view on the left results in a better visualization. In our multi-perspective image, some unwanted curving is introduced into the walkway at the center of the image. This walkway is in fact straight.



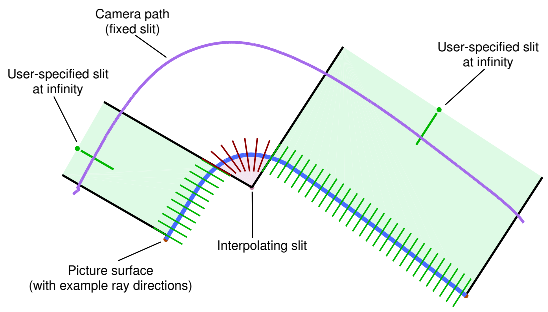
(a)



(b)

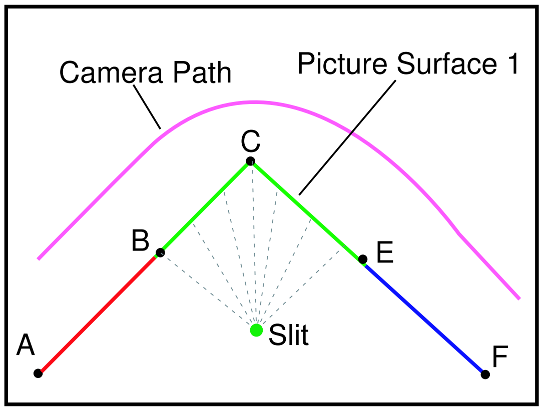


(c)

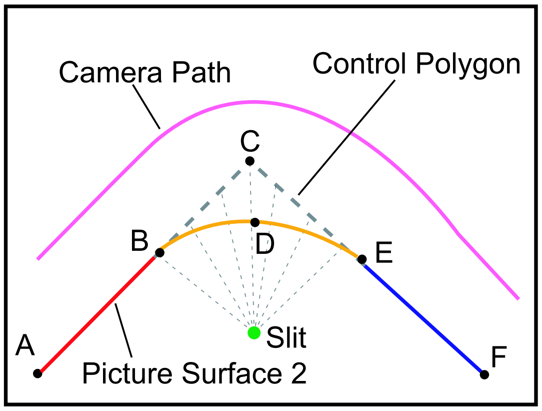


(d)

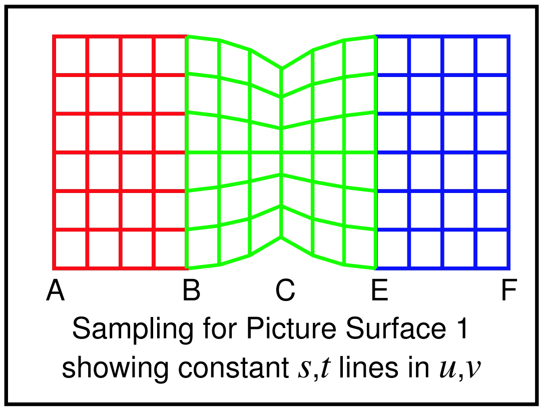
Figure 3.5: Our visualization of a street corner with perpendicular storefronts. (a) shows the multi-perspective image generated by our system for the choice of picture surface shown in (c). (b) shows the multi-perspective image generated by our system for the choice of picture surface shown in (d). The picture surface in (c) conforms to the actual storefront whereas it has been artificially curved in (d) using our interactive system. Note that the slits are at the same location in both the setups. (a) gives the impression that the storefront is continuous and there is no corner. By altering our picture surface we introduce an artificial pinching effect in the multi-perspective image shown in (b). This helps emphasize that it is a corner without causing severe distortions.



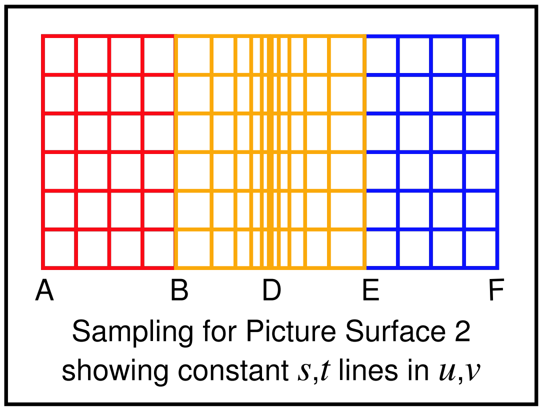
(a)



(b)


A
B
C
E
F
Sampling for Picture Surface 1
showing constant s,t lines in u,v

(c)


A
B
D
E
F
Sampling for Picture Surface 2
showing constant s,t lines in u,v

(d)

Figure 3.6: This figure explains how our choice of a curved surface and a non-uniform sampling strategy achieves a pinching effect at the corners seen in Figure 3.5(b). In (a), picture surface 1 conforms to the storefront. To achieve the pinching effect in this case we would need to stretch the image at C. This would mean non-linearly mapping the picture surface *uv* coordinates to the final output image *st* coordinates as shown in (b) to maintain continuity in the image. We achieve the same effect by instead using a curved surface as shown in (d) but sampling according to the strategy in (c). This solves the problem of nonlinear vertical sampling, but now requires adjusting the horizontal sampling. This can be easily done by using the control polygon to define the horizontal mapping from 3D *xyz* to *u* but still mapping to *s* linearly. If the control polygon corresponds to the storefront, then a brick on the wall will not appear to be stretched horizontally.

Chapter 4

Rendering Multiperspective Images from Real Data

With a user interface that allows a user to easily specify the structure of a multiperspective image, the next step is to render that image. This chapter describes the algorithm used to render our multiperspective images that are composed of several abutting crossed-slits images. First we describe the requirements on the input data and then we describe the basic rendering algorithm. In Section 4.3, we improve upon the rendering method to handle cases where the input data is not sufficiently dense, and in Section 4.4 we describe how we handle the extremely large dynamic range typical in outdoor scenes.

4.1 Data Requirements

There are three basic assumptions about the input data required for rendering. The first is that the input images are from a moving video camera. This results in a 3D video dataset instead of a 4D lightfield, restricting the set of multiperspective images that can be created. Second, we assume that the environment is approximately static.

The last assumption is that the camera trajectory must be known. Structure-from-motion software can be used to extract this directly from the video in some cases. External constraints (such as placing the camera on a track) or auxiliary sensors (GPS, accelerometers, etc.) can also be used to robustly determine the camera path and improve accuracy.

Video data must be captured densely enough so that the sampling on the picture surface is equal both vertically and horizontally. That is, the height of a single pixel from an input camera image (when projected onto the picture surface) should equal the width of a column on the final multiperspective image.

In our current implementation, data is captured using a Basler A504kc high-speed camera which captures 1280×1024 images at 300 frames per second with a 72-degree field of view vertically. With a picture surface that is 10m from the camera path, each pixel is approximately 1.5cm high. The camera should therefore not move more than 1.5cm between video frames. At 300 frames per second, that corresponds to a maximum speed of 10 mph. The high speed capture ensures dense image data while still allowing a reasonable driving speed for urban traffic; higher speeds are possible using view interpolation, as described in Section 4.3.

Camera pose is then estimated with a structure-from-motion (SFM) algorithm. We have used both a commercial software package Boujou [2D306] and a freely available software package Voodoo [Uni03]. SFM also outputs partial 3D scene structure, which helps the user in choosing crossed-slits locations in the interactive UI or helps the optimization algorithm for the automated specification. For some of our datasets, the camera pose is computed using a combination of external sensors. In these cases, additional scene structure is available through the use of laser range scanners. We assume that the X-Z plane that camera pose is estimated in corresponds to the real-world ground plane.

4.2 Algorithm for Rendering MPIs

Using an Intel 2.8 GHz Pentium 4 machine, the low-resolution preview image (up to 1000 × 200) in our user interface can typically be rendered in under a second. An efficient rendering engine is necessary to provide this fast feedback to the user.

Our task is to determine for each pixel in the output image which pixel position (which may fall between captured pixels) in which frame of the input video should be displayed there. This is done in four steps:

1. Compute the projection of each input frame onto the output image.

2. Project the center-of-projection of each from onto the output image.
3. For each column of the output image, choose an input frame.
4. Use the projection of the input frame to extract and transform the appropriate pixels onto the output image.

The constraints imposed by our design choices allow us to simplify and accelerate the rendering, in particular the assumption of a discretized, piecewise-planar picture surface. Since each point on the picture surface is associated with only one crossed-slits camera, the final output image is composed from several distinct, abutting crossed-slits images. Each crossed-slits camera can span multiple planar segments of the picture surface. Also, each planar segment may contain multiple crossed-slits cameras. In both cases, we render only a single crossed-slits image onto a single planar segment at a time. Each planar segment is parameterized by (u,v). Finally, the restriction of vertical sides for the picture surface and of vertical user-specified slits allows us to assign entire columns of the final output image from a single input video frame. For each crossed-slits image on a planar segment, there is a mapping between the (u,v) coordinates of the planar segment to the (s,t) coordinates of the final output image, defining the sampling of the picture surface. This can be seen in Figure 4.1. The sampling on each planar segment is regular in the u and v parameter directions.

A fast algorithm to render a single crossed-slits image onto a single planar surface is therefore the basic building block used in rendering. For each crossed-slits image, we compute the homography between each input frame and the final output image. We then use this transform to compute which pixels will actually contribute to the final output image and then transform only those pixels.

An important point is that the *entire* input image does not need to be transformed. Instead, we first calculate the necessary transformation and then apply it to only a small region from each input image. This speeds up rendering to the point that if all images can be preloaded into RAM, previews can be rendered at interactive rates even for thousands of input images.

We now describe the process of computing the homography transform from an input video frame to the output image for a single planar segment. We use the convention of

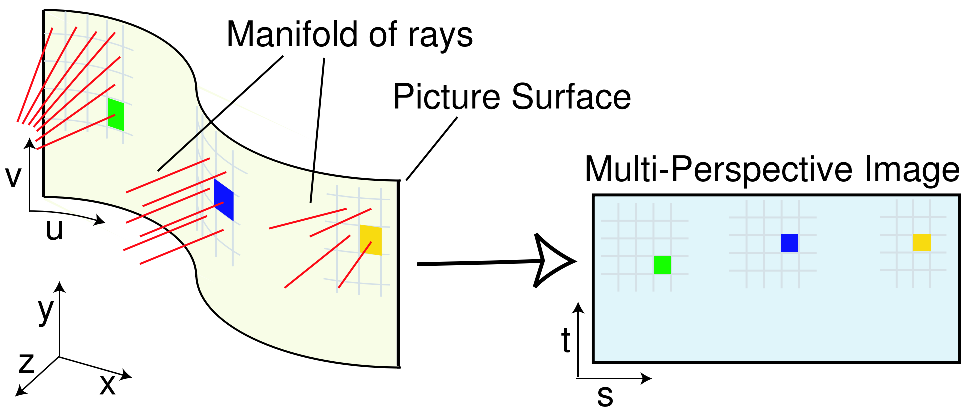


Figure 4.1: The ray manifold is the set of rays in space. The sampling of these rays and the mapping to the output image can be specified by a picture surface with a 2D parameterization of its 3D surface.

using bold for vectors (lowercase) and matrices (uppercase). Points in different coordinates systems are

$\mathbf{x} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$	3D point in world coordinate system in which camera pose is estimated
$\mathbf{u} = \begin{bmatrix} u & v & 1 \end{bmatrix}^T$	2D point in the coordinate system of the planar segment of the picture surface
$\mathbf{p} = \begin{bmatrix} p & q & 1 \end{bmatrix}^T$	2D pixel location of a single input video frame
$\mathbf{s} = \begin{bmatrix} s & t & 1 \end{bmatrix}^T$	2D pixel location of the final multiperspective image

These are also depicted in Figure 4.1.

Let the origin of the planar segment in world 3D homogeneous coordinates be the 4x1 vector $\mathbf{o}$. Let $\mathbf{w}$ and $\mathbf{h}$ be 4x1 vectors defining the width and height extent of the segment. Then, the mapping from a point $\mathbf{u}$ (for all $u, v \in [0,1]$) on the planar segment of the picture surface to a corresponding world point $\mathbf{x}$ is given by

$$\mathbf{x} = u\,\mathbf{w} + v\,\mathbf{h} + \mathbf{o} \tag{4.1}$$

$$= \begin{bmatrix} \mathbf{w} & \mathbf{h} & \mathbf{o} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4.2}$$

$$= \mathbf{Q}\,\mathbf{u} \tag{4.3}$$

In general, a point in the world coordinate system **x** is mapped into a particular input video frame *i* by the relationship:

$$\mathbf{p} = \mathbf{K}\,[\mathbf{R}_i\,|\,\mathbf{t}_i\,]\,\mathbf{x} \tag{4.4}$$

given the camera intrinsics **K**, the rotation matrix $\mathbf{R}_i$, and the translation vector $\mathbf{t}_i$. Substituting for **x**, we obtain the relationship between the planar segment and the input video frame:

$$\mathbf{p} = \mathbf{K}\,[\mathbf{R}_i\,|\,\mathbf{t}_i\,]\,\mathbf{Q}\,\mathbf{u} \tag{4.5}$$

With our restriction of regular sampling on the planar segments, we map the segment to a rectangular region of the final output image using only translation and scaling. Note that this choice of mapping is arbitrary. We can now define the relationship between this planar segment of the picture surface and the final output image as:

$$\mathbf{s} = \begin{bmatrix} a & 0 & c \\ 0 & b & d \\ 0 & 0 & 1 \end{bmatrix} \mathbf{u} = \mathbf{M}\,\mathbf{u} \tag{4.6}$$

or equivalently

$$\mathbf{u} = \mathbf{M}^{-1}\,\mathbf{s} \tag{4.7}$$

where c and d define the origin of the rectangular region on the final output image and a and b define the width and height of the region respectively.

We can then obtain a direct relationship between each input video frame i and the final multiperspective image as

$$\mathbf{p} = \mathbf{K}\,[\mathbf{R}_i\,|\,\mathbf{t}_i\,]\,\mathbf{Q}\,\mathbf{M}^{-1}\,\mathbf{s} \tag{4.8}$$

Using this formulation, all of the intermediate matrices reduce to a single invertible 3x3 matrix that we call $\mathbf{H}$, giving

$$\mathbf{p} = \mathbf{H}\,\mathbf{s} \tag{4.9}$$

$$\mathbf{s} = \mathbf{H}^{-1}\,\mathbf{p} \tag{4.10}$$

The matrix $\mathbf{H}$ defines the homography between an input video frame and a portion of the final multiperspective image corresponding to one planar segment of the picture surface.

With the mapping from each input video frame to the final output image, each pixel on the final output image will have multiple input frames overlapped onto it. We must now choose which input frame to select for each pixel. For our crossed-slits images, one of the slits is fixed—the camera path. The second, vertical slit is either a user-specified or interpolated slit. We project a point on this vertical slit through each input video frame onto the final output image.

To compute this, we take the 3D point on the second slit that intersects the ground plane, $\mathbf{x}_g$, project it onto the input video frame ($\mathbf{p}_g$), and then use the homography derived above to project the point onto the final output image ($\mathbf{s}_g$). For a particular input video frame i, the image of $\mathbf{x}_g$ through that camera's center of projection maps to

$$\mathbf{p}_g = \mathbf{K}\,[\mathbf{R}_i\,|\,\mathbf{t}_i\,]\,\mathbf{x}_g \tag{4.11}$$

$$\mathbf{s}_g = \mathbf{H}^{-1}\,\mathbf{p}_g \tag{4.12}$$

As shown in Figure 4.2, for each column of the final output image, we then select the input frame with the closest projected point $\mathbf{s}_g$ and transform the pixels from that frame that

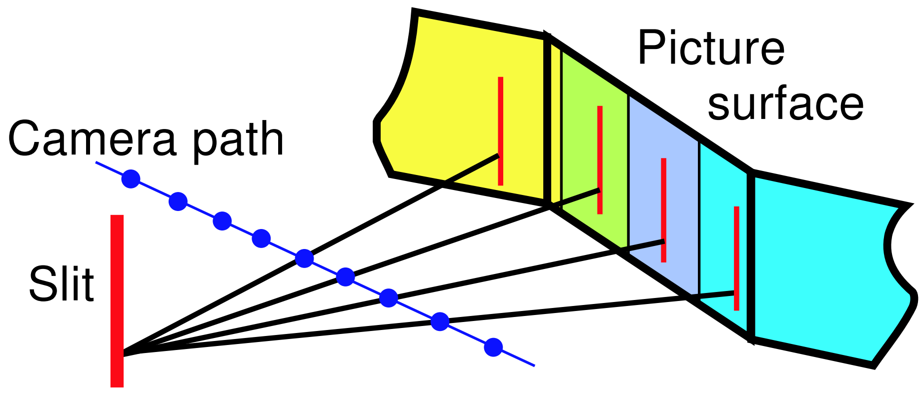


Figure 4.2: The slit is projected through each camera onto the picture surface. The region closest to each projection is assigned to the corresponding camera.

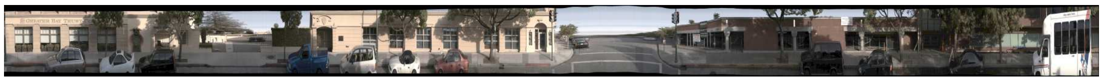
correspond to the output image column.

4.3 View Interpolation

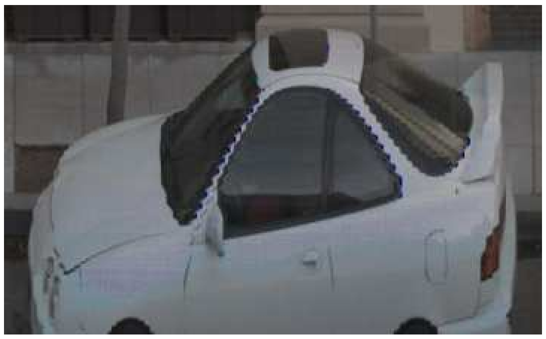
This rendering algorithm relies on having very dense input video, ideally one frame for each column of the output image. Often, however, such densely sampled video is not practically available. In these cases, we use view interpolation to synthesize intermediate views.

For example, the initial acquisition of the Castro street dataset (Figure 3.3) used a 30 frame per second video camera driving down the street at about 2 mph. This low speed is not practical for more than test datasets. In contrast, even though the 18$^{\text{th}}$ street dataset (Figure 5.5) was taken with a high-speed camera operating at approximately 70 frames per second, the average driving speed of 20 mph causes each image to shift nearly 20 pixels when reprojected onto the picture surface. Without view interpolation, artifacts such as those shown in Figure 4.3 are common.

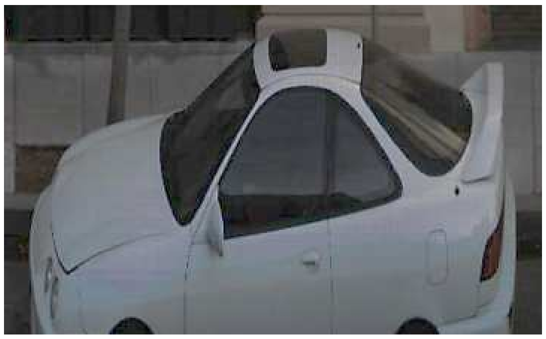
View interpolation fits very simply into the rendering algorithm described above. As each input video frame projects onto the picture surface at location $\mathbf{s_g}$ (equation 4.12), any particular column will lie between two input video frames. If $\alpha \in [0,1]$ is the normalized position of the column between the two input video frames, then we generate an interpolated intermediate view I_α between the two input video frames. If we also generate the intermediate camera matrices $\mathbf{R}_\alpha$ and $\mathbf{t}_\alpha$, we can render from the interpolated frame exactly



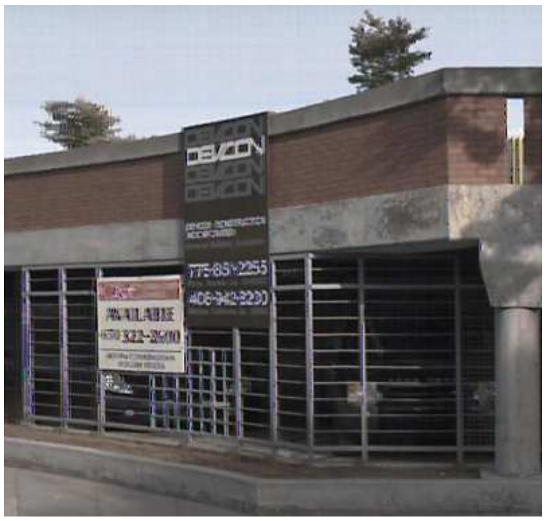
(a)



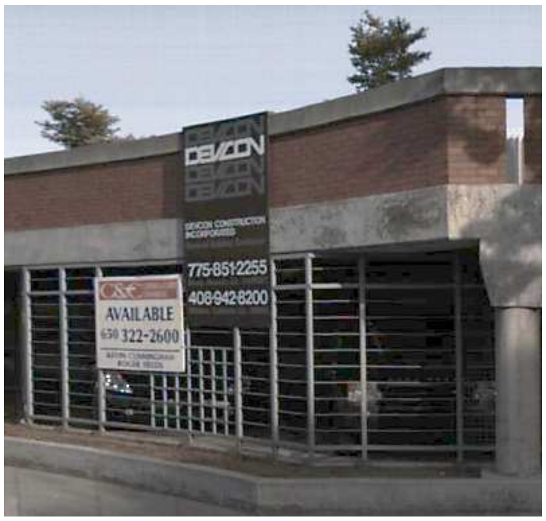
(b)



(c)



(d)



(e)

Figure 4.3: Example of striping artifacts caused by insufficient video framerates. (a) A section of Lytton Street in Palo Alto, CA. At full resolution this image is 7200 pixels wide but is resampled here to only 1800 pixels wide. When displayed at this lower resolutions, the video framerate has little effect on the resulting image. (b) A full-resolution crop of the image around the car without optical flow. This image is 360 pixels wide. (c) The same car rendered from the same video, but using optical flow to interpolate between input video frames. (d) A 400-pixel wide crop of the image showing a sign. Notice that the phone numbers are severely distorted. (e) The same crop rendered using optical flow. The phone numbers are now legible.

as if it were an input video frame using the formulation in (4.8).

Notice that after generating an entire intermediate viewpoint, we extract only a small region of the generated frame. It is desirable to reduce the computation and generate only the necessary intermediate pixels. In our datasets, this typically implies generating an intermediate view for a region only 10 pixels wide. Unfortunately, the view interpolation requires sufficient context to robustly establish correspondence between pixels in both input video frames. We have empirically found that our view interpolation requires 300-500 pixel wide regions to match robustly.

The problem of view interpolation is therefore: Given two images I_1, I_2 along with the associated camera matrices ($\mathbf{K}\left[\mathbf{R_1}|\mathbf{t_1}\right]$ and $\mathbf{K}\left[\mathbf{R_2}|\mathbf{t_2}\right]$ respectively) and a desired intermediate position $\alpha \in [0,1]$, compute an estimated intermediate image I_α and the corresponding camera matrices $\mathbf{K}\left[\mathbf{R}_\alpha|\mathbf{t}_\alpha\right]$.

The camera motion between the two images is typically small and thus the intermediate camera matrices are approximated by linear interpolation for the position and spherical linear interpolation[Sho85] via quaternions for the orientation.

The intermediate image is generated using view interpolation. The two methods we used are described here, however there is a rich literature for view interpolation [CW93, SD96, GKG04, XS04].

1. Compute an unconstrained, dense optical flow field between the two images. This flow field is scaled according to α and used directly to generate the desired intermediate view. A robust optical flow implementation based on [Bla92, BA93] was implemented. This method is simpler and requires no additional information beyond the two images. The disadvantage of this method is that it does not exploit the additional available information of the camera pose.

2. Extend the optical flow to exploit the known camera pose. First rectify the two images so that the epipolar lines are parallel and then constrain the optical flow computation to be only along the epipolar lines. This gives a disparity estimate for each pixel of the rectified images that is used to generate a rectified version of the desired intermediate image. Finally, we apply an inverse homography that maps the rectified

intermediate image to our desired intermediate image. While this method provides improved estimation due to the added constraints, it requires accurate camera pose.

4.3.1 Method 1: Optical Flow

Optical Flow Field Computation

We implemented the robust optical flow described by [Bla92, BA93], briefly summarized here. Given two luminance images, $I_1(x,y)$ and $I_2(x,y)$, of a scene with arbitrary motion, the goal is to compute the motion for each pixel $D_u(x,y)$ and $D_v(x,y)$. We assume that each pixel in I_1 corresponds to a pixel in I_2 and that the intensity of corresponding pixels is equal.

Relying on the intensity constancy assumption, we assume that the image can be represented by

$$I_1(x,y) = I_2(x + D_u(x,y), y + D_v(x,y)) \tag{4.13}$$

for some flow field (D_u, D_v).

These can be estimated by minimizing the robust residual error

$$E_D(u,v) = \rho(I_x D_u + I_y D_v + I_t, \sigma_1) \tag{4.14}$$

where ρ is a robust estimator and σ_1 weights the influence of intensity outliers in the optimization. We use the Lorentzian estimator as in [BA93], so that

$$\rho(x,\sigma) = \log\left(1 + \frac{1}{2}\left(\frac{x}{\sigma}\right)^2\right) \tag{4.15}$$

where σ is the normalization parameter. When estimating a dense flow field, (4.13) is underconstrained, so we add the robust regularization term to favor smooth flow fields. This term is

$$E_S(D_u, D_v) = \sum_{u \in D_u} \sum_{u_n \in \mathfrak{N}} \rho(u - u_n, \sigma_2) + \sum_{v \in D_v} \sum_{v_n \in \mathfrak{N}} \rho(v - v_n, \sigma_2) \tag{4.16}$$

where $\mathfrak{N}$ contains the four neighbors (north, south, east, west) of a given point and σ_2 weights the influence of smoothness outliers.

The final energy term that is minimized is

$$E = E_D + \lambda E_S \tag{4.17}$$

where λ controls the regularization. This is solved using a coarse-to-fine simultaneous over-relation (SOR) implementation as described in [Bla92, BA93].

View Interpolation Using Flow Fields

To generate an intermediate image that is α between the two input images (with $\alpha \in [0,1]$), the images are bilinearly resampled using a fraction of the computed flow fields.

$$I_{\alpha_1}(x,y) = I_1(x - \alpha D_u(x,y), y - \alpha D_v(x,y)) \tag{4.18}$$

$$I_{\alpha_2}(x,y) = I_2(x + (1-\alpha) D_u(x,y), y + (1-\alpha) D_v(x,y)) \tag{4.19}$$

Linearly blending these images creates the desired intermediate image I_α.

$$I_\alpha = (1-\alpha) I_{\alpha_1} + \alpha I_{\alpha_2} \tag{4.20}$$

4.3.2 Method 2: Stereo Flow

Rectification

Stereo algorithms typically assume that the images are *rectified*, that is, the epipolar lines are parallel. In many stereo applications, this is enforced by the physical arrangement of a pair of stereo cameras. In our case, we rectify the images based on their relative positions and orientations that was required input for our system.

As in section 4.2, we assume that the camera intrinsics ($\mathbf{K}$), rotation matrices ($\mathbf{R_1}, \mathbf{R_2}$), and the translation vectors ($\mathbf{t_1}, \mathbf{t_2}$) are known. The goal is to compute a new rotation $\widehat{\mathbf{R}}$ and translations for each camera $\hat{\mathbf{t}}_1, \hat{\mathbf{t}}_2$ that aligns the epipolar lines to scanlines. To center the images, offset are applied to the intrinsics matrix $\mathbf{K}$ to form $\widehat{\mathbf{K}}_1$ and $\widehat{\mathbf{K}}_2$.

We used the the rectification algorithm given by Fusiello et al [FTV00], summarized here. First compute the center of projection for each camera as

$$\begin{aligned} \mathbf{C_1} &= -\mathbf{R_1^T t_1} \\ \mathbf{C_2} &= -\mathbf{R_2^T t_2} \end{aligned} \tag{4.21}$$

These two points define the direction of motion of the camera, and so we rotate the camera image so that the x-axis of the rectified images is aligned with this baseline. A coordinate basis aligned with this baseline is constructed by

$\mathbf{x} = \mathbf{C_2} - \mathbf{C_1}$	Define baseline direction
$\mathbf{z}' = \mathbf{R_1^T} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathbf{T}}$	Extract original z vector from input rotation matrix
$\mathbf{y} = \mathbf{x} \times \mathbf{z}'$	Choose orthogonal y
$\mathbf{z} = \mathbf{x} \times \mathbf{y}$	Choose orthogonal z

With this orthogonal basis, we assemble the rotation matrix simply using the basis vectors as rows.

$$\widehat{\mathbf{R}} = \begin{bmatrix} \mathbf{x^T} & \mathbf{y^T} & \mathbf{z^T} \end{bmatrix}^T \tag{4.22}$$

We also construct new intrinsics matrices. The intrinsics are equal to the original camera's intrinsics matrix except that an offset can be applied to center the images after the rectification. (See [FTV00] for details)

$$\begin{aligned} \widehat{\mathbf{K}}_1 &= \mathbf{K} + \begin{bmatrix} 0 & 0 & dx_1 \\ 0 & 0 & dy_1 \\ 0 & 0 & 0 \end{bmatrix} \\ \widehat{\mathbf{K}}_2 &= \mathbf{K} + \begin{bmatrix} 0 & 0 & dx_2 \\ 0 & 0 & dy_2 \\ 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{4.23}$$

Finally, we can compute the two homographies.

$$\begin{aligned}\mathbf{H_1} &= \widehat{\mathbf{K}}_1\widehat{\mathbf{R}}\left(\mathbf{KR_1}\right)^{-1} \\ \mathbf{H_2} &= \widehat{\mathbf{K}}_2\widehat{\mathbf{R}}\left(\mathbf{KR_2}\right)^{-1}\end{aligned} \tag{4.24}$$

Because the epipolar lines are now aligned with the x-axis, all motion due to parallax will manifest as offsets in the x direction in the rectified images. Apply this homography to the input video frames I_1,I_2 gives us the rectified input frames $\hat{I}_1$ and $\hat{I}_2$.

Stereo

We now constrain the robust optical flow formulation described in section 4.3.1 to only estimate flow in the x direction. When computing the flow for the rectified images, this has the effect of constraining the correspondence search to lie along the epipolar lines. After rectification, this is done simply by constraining the flow field to the x axis, and is accomplished by dropping the v term entirely from (4.13) to become

$$\hat{I}_1(x,y) = \hat{I}_2(x+\hat{D}_u(x,y),y). \tag{4.25}$$

Similarly, (4.14) and (4.16) become

$$E_D(D_u) = \rho(\hat{I}_x\hat{D}_u + \hat{I}_t, \sigma_1) \tag{4.26}$$
$$E_S(D_u) = \sum_{u\in\hat{D}_u}\sum_{u_n\in\mathfrak{N}} \rho(u-u_n, \sigma_2). \tag{4.27}$$

The resulting flow field for the rectified images $\hat{D}_u$ is now considered a disparity map $\hat{D}(x,y)$ such that

$$\hat{I}_1(x,y) = \hat{I}_2(x+\hat{D}(x,y),y) \tag{4.28}$$

View interpolation

Once we have computed the disparity between the rectified images, generating an intermediate rectified image is straightforward. An inverse rectification homography must be applied to the intermediate rectified image before the image can be used as a virtual input video frame in the rendering system.

The rectified intermediate image is computed by resampling both input rectified images by a fraction of the disparity, and then linearly blending those images. To compute an image that is α between the two input images (with $\alpha \in [0,1]$), we first resample both input rectified images by a fraction of the disparity:

$$\hat{I}_{\alpha_1}(x,y) = \hat{I}_1(x - \alpha\hat{D}(x,y), y) \tag{4.29}$$

$$\hat{I}_{\alpha_2}(x,y) = \hat{I}_2(x + (1-\alpha)\hat{D}(x,y), y) \tag{4.30}$$

Linearly blending these images creates an intermediate rectified image $\hat{I}_\alpha$:

$$\hat{I}_\alpha = (1-\alpha)\hat{I}_{\alpha_1} + \alpha\hat{I}_{\alpha_2} \tag{4.31}$$

In order to sample from this intermediate rectified image using the framework described above, it is necessary to find the relationship between an intermediate camera image specified by $\mathbf{K}[\mathbf{R}_\alpha|\mathbf{t}_\alpha]$ and the rectified intermediate camera image $\hat{I}_\alpha$ just computed. The rectification homography $\mathbf{H}_\alpha$ that relates these is computed as follows.

The offset in $\widehat{\mathbf{K}}$ is simply

$$\widehat{\mathbf{K}}_\alpha = (1-\alpha)\widehat{\mathbf{K}}_1 + \alpha\widehat{\mathbf{K}}_2 \tag{4.32}$$

The intermediate rectified image can be described by the camera matrices $\widehat{\mathbf{K}}_\alpha[\widehat{\mathbf{R}}_\alpha|\hat{\mathbf{t}}_\alpha]$. This is related to the desired non-rectified intermediate image by the desired homography $\mathbf{H}_\alpha$.

$$\mathbf{H}_\alpha\left(\mathbf{K}[\mathbf{R}_\alpha|\mathbf{t}_\alpha]\right) = \widehat{\mathbf{K}}_\alpha[\widehat{\mathbf{R}}_\alpha|\hat{\mathbf{t}}_\alpha] \tag{4.33}$$

From this we get

$$\mathbf{H}_\alpha\mathbf{K}\mathbf{R}_\alpha = \mathbf{K}'_\alpha\mathbf{R} \tag{4.34}$$

$$\mathbf{H}_\alpha = \mathbf{K}'_\alpha\mathbf{R}\left(\mathbf{K}\mathbf{R}_\alpha\right)^{-\mathbf{1}} \tag{4.35}$$

Notice the similarity between (4.35) and (4.24). Transforming $\hat{I}_\alpha$ by $\mathbf{H}_\alpha^-\mathbf{1}$ gives us the desired intermediate image I_α which we use directly in the rendering system.

4.4 High Dynamic Range

On a sunny day, the dynamic range between a fully sunlit surface and a fully shadowed surface can span a range of 10,000:1 [XDCW02]. This poses a challenge for most modern digital cameras which typically have a 10- to 12-bit dynamic range capability on the sensors. For our application, our camera (a Basler A504kc) used a linear 8-bit sensor. In order to increase the quality of the output images, we assembled high dynamic range versions of the multiperspective images. This section describes how these images were created, and is divided into three parts. First, we briefly described how the camera was setup to camera the images. Second, we describe how to assemble several low dynamic range images, and finally we explain our decision to render the images before assembling into a high-dynamic range composite.

4.4.1 Capture

The Basler camera is programmed to continuously cycle through a sequence of three preprogrammed exposures. These are chosen according to the approximate average brightness of the day and spaced by factors of 8. For example, 0.125ms, 1ms, and 8ms are typical exposure times.

Because the vehicle is moving continuously, none of the frames are have exactly the same viewpoint. Thus, the images cannot be directly recombined into a high dynamic range composite as described by [DM97]. There are two possible approaches to generating the high dynamic range result:

1. Use view interpolation to combine neighboring frames into a high-dynamic range video and then use that video for rendering, as in [KUWS03].

2. Render the multiperspective image for each exposure separately and then combine the separate images into a high-dynamic range composite.

We used the second approach. It has the advantage that it does not require any view interpolation to function correctly if the input video stream is already sufficiently dense. Otherwise the two methods require approximately the same amount of computation.

4.4.2 Rendering

Because we render each exposure separately, rendering is performed exactly as described in previous sections, once for each video stream.

4.4.3 Assembly

Assembling high-dynamic range images is described in detail in [DM97, XDCW02]. Because we have a restricted input dataset, we can simplify the assembly process. In particular, we have a linear sensor with known, fixed exposure times and a fixed set of exposures.

We define three weighting functions used to normalize the radiance of the input images:

$$w_{low}(c) = 1 - \text{clip}\left(s(K-c), 0, 1\right) \tag{4.36}$$

$$w_{med}(c) = 1 - \text{abs}\left(s(K-c)\right) \tag{4.37}$$

$$w_{high}(c) = 1 + \text{clip}\left(s(K-c), 0, 1\right) \tag{4.38}$$

where K determines the midpoint of the range and s normalizes the values. In our case, we use $K = 110$ and $s = 100$. These functions are plotted in Figure 4.4.

These weights are then used to directly assemble the high-dynamic range image from the low-dynamic range input images. For a set of E exposures, we have a set of E corresponding pixels p_e and exposure times t_e. The high dynamic value r_p can computed by first computing a weighted average of the scaled radiance:

$$r_p = \frac{\sum_e w_e(p_e) t_e^{-1} p_e}{\sum_e w_e(p_e)} \tag{4.39}$$

Here, w_e is used to denote the weighting function appropriate to the exposure. For only three exposures, $w_e \equiv \left\{w_{low}, w_{med}, w_{high}\right\}$ for $e \equiv \{1, 2, 3\}$ respectively. For more exposures, w_{med} is used for all interior exposures.

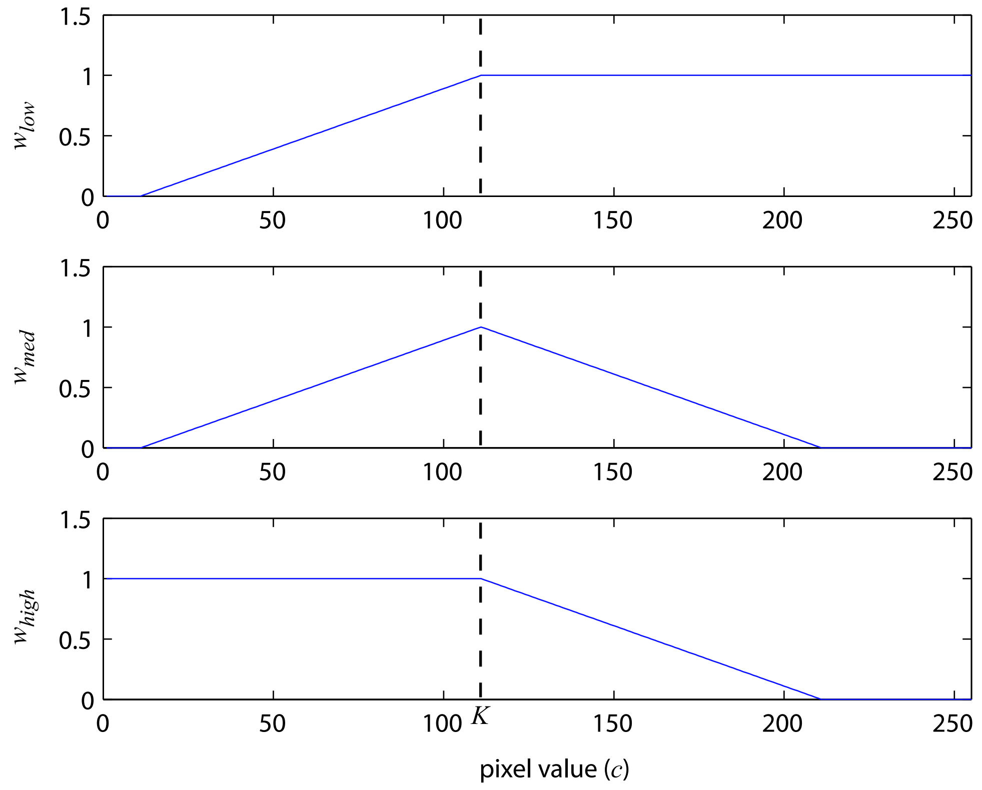


Figure 4.4: Weighting function used for determining pixel contribution when assembling high dynamic range images.

Once the high-dynamic range image has been assembled, it must be mapped back into a low-dynamic range image for display. This process is called tone-mapping. There are several tone-mapping operators available, including methods such as [FLW02]. However, we found the operation that produces the most realistic images for our datasets was a simple gamma-curve tone mapping. This simply maps radiance values to pixel values using

$$p_i = r_i^{\gamma^{-1}} \tag{4.40}$$

with appropriate scaling and offset to fit the desired output range.

Examples of the input and output of the high-dynamic range imagery can be seen in Figures 4.5 and 4.6.

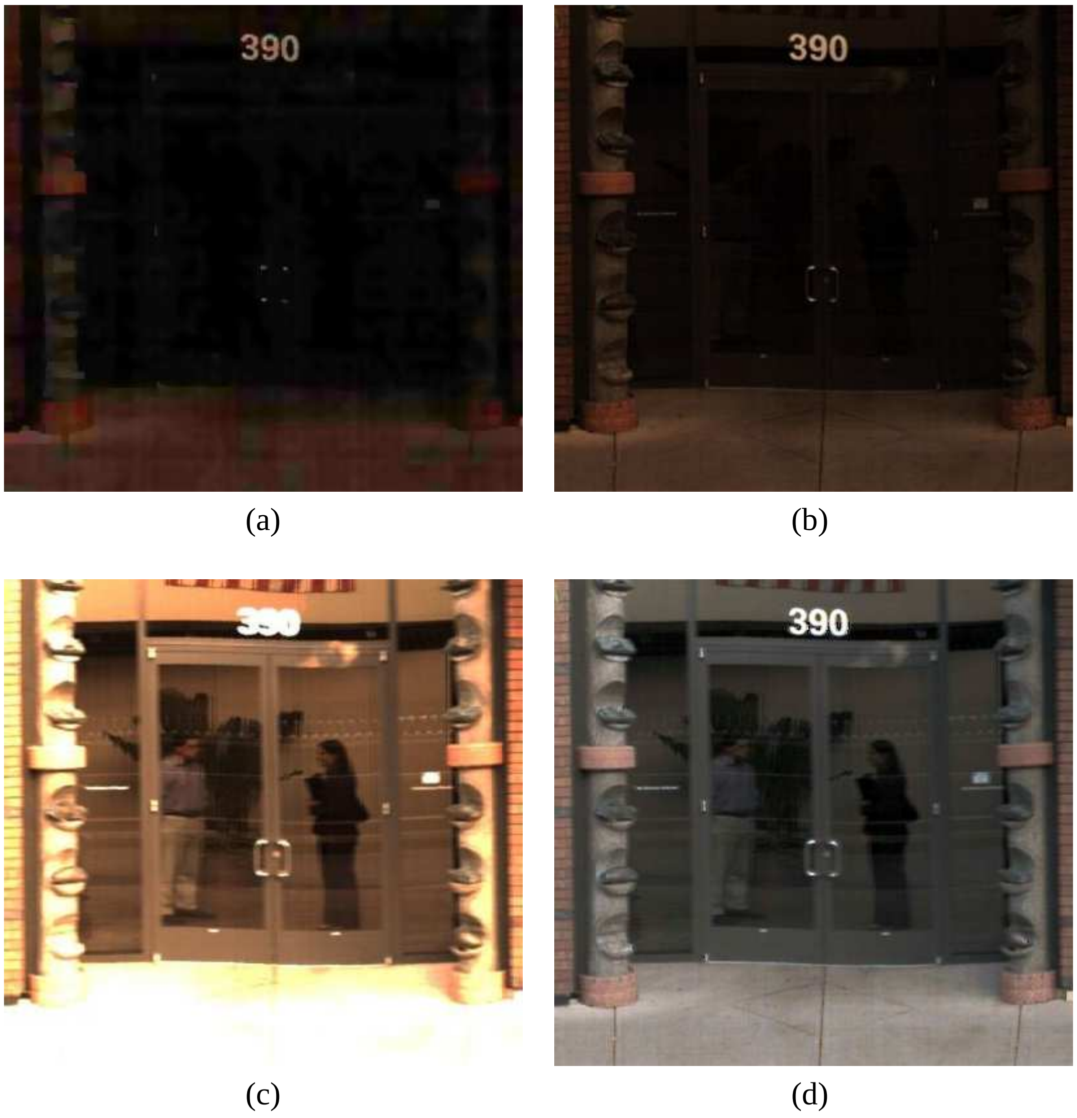


Figure 4.5: An example of the high-dynamic range assembly. This example shows a crop of a particularly dark area of the street, in contrast to the section shown in Figure 4.6. These two crops are taken from the same street dataset and appear in different sections of the same multiperspective image. (a)-(c) A segment of the multiperspective images rendered at each of three exposures. Note that the image in (a) has been brightened by a factor of 32 for display purposes. (d) The same segment after combining the three low-dynamic range images and tone-mapping using a gamma curve. Notice that detail is clearly visible both inside the store and on the sidewalk in front, but that using any individual low dynamic range input image would not be able to capture detail both here and in the example in Figure 4.6.

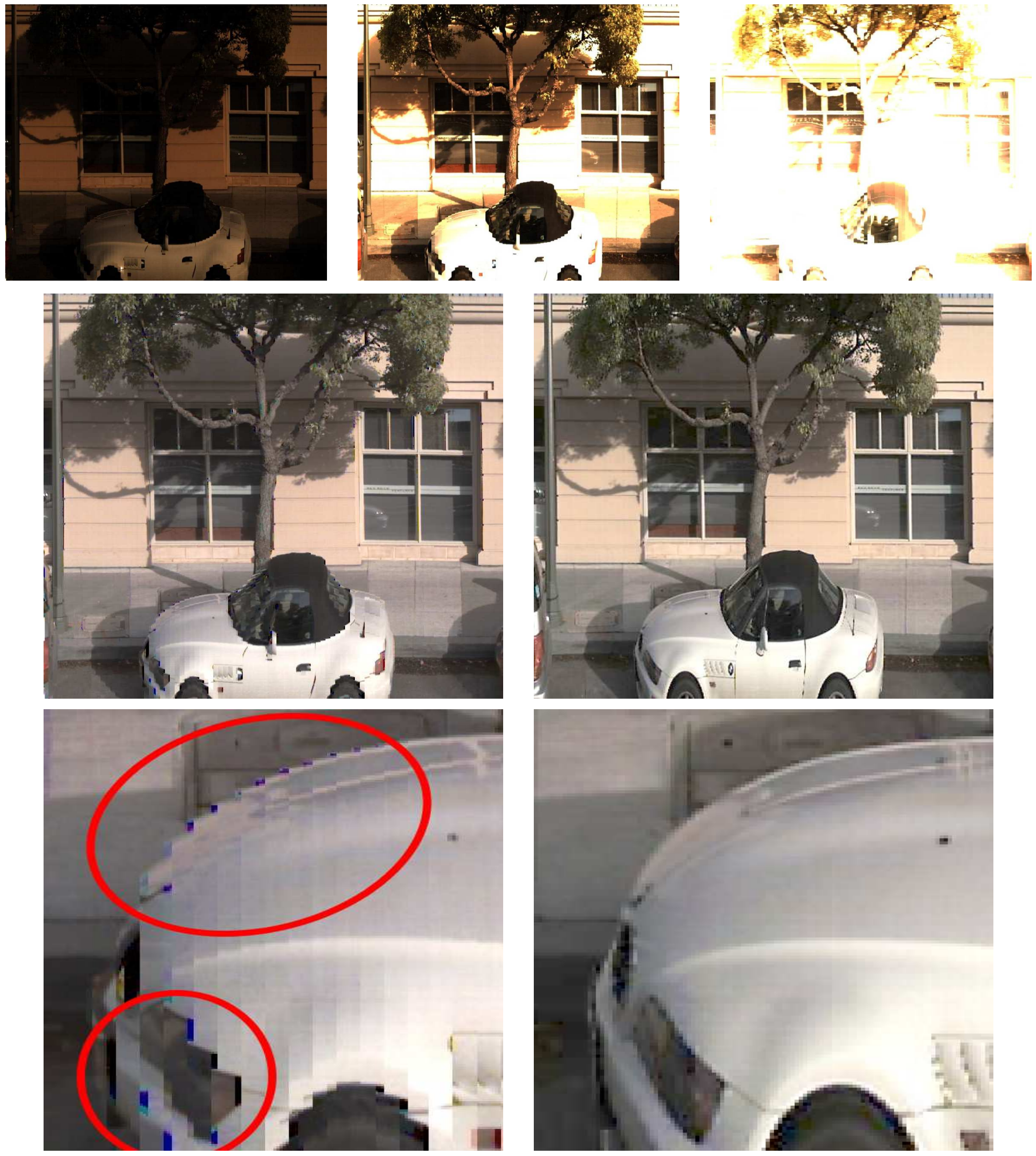


Figure 4.6: An example of the high-dynamic range assembly, including the effects of optical flow. This example shows a crop of a brighter area of the street, in contrast to the darker section shown in Figure 4.5. (top row) A segment of the multiperspective images rendered at each of three exposures. (middle left) The same segment after combining the three low-dynamic range images and tone-mapping using a gamma curve, without optical flow. (middle right) The same region after rendering the three segments using optical flow and then combining them. (bottom row) Zoom of middle row. Notice the lack of color artifacts on the front edge of the car.

4.5 Discussion

The rendering engine is essential to making our user interface effective, and consists of three basic components:

1. The basic crossed-slits image rendering engine that resamples pixels from a set of input images to the specified output image.

2. A view interpolation engine that can be used to trade-off dense input imagery against computational complexity.

3. A high-dynamic range assembly engine to handle the large dynamic range of typical outdoor scenes.

For our user interface, we typically use only the first component to render low-resolution preview images. This provides interactive updates making the user-interface more effective for constructing the desired image. For the final, high-quality output image, we additionally enable the view interpolation and high-dynamic range engines.

Chapter 5

Perspective Optimization

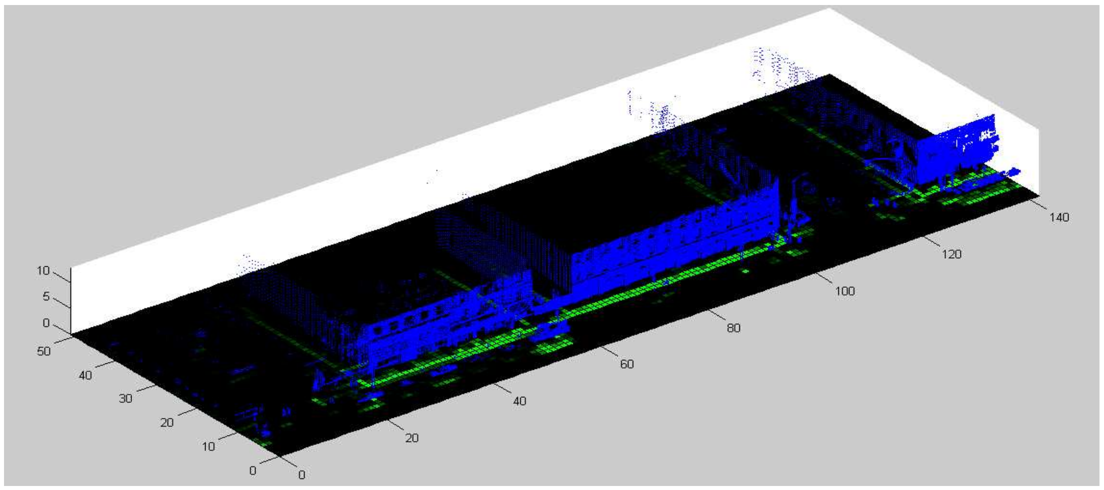
The interactive user interface described in Chapter 3 is useful for building an intuition about multiperspective images. However, it is not practical for constructing multiperspective images on a large scale such as for an entire city. At this scale, it is absolutely essential to be able to automatically generate images. One possible automatic method is to simply generate pushbroom images that face in a direction perpendicular to the vehicle motion, similar to the work by Zheng [Zhe03]. The downside of this approach is that it negates the potential benefits that adaptive multiperspective images can provide as shown in Chapter 3.

We have developed an optimization that can automatically improve pushbroom images and define multiperspective images similar to those shown in Chapter 3. This optimization can be fully automatic or it can accept a user-specified importance map to guide the optimization. The importance map is simply a 2D weighting function that modulates the cost for each scene point. Because this importance map is specified in world space, it is plausible to automatically generate it based on external metadata without requiring any image processing.

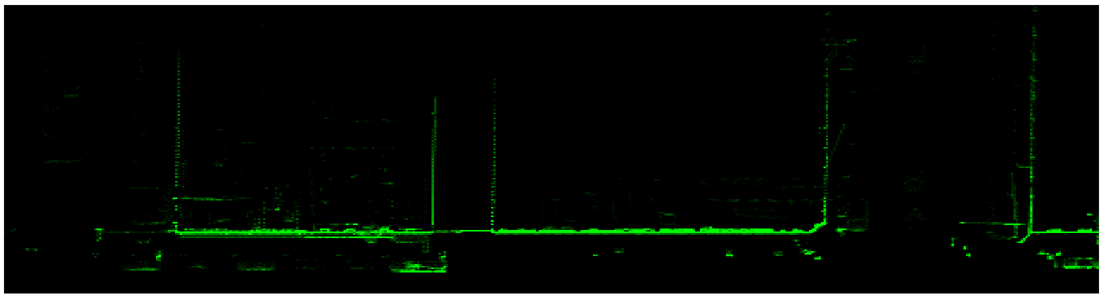
5.1 Input data

The perspective optimization system has similar input requirements to the rendering requirements specified in Section 4.1, however the video images are not used in the optimization. Therefore, the input is assumed to be a set of video frame positions ($\mathbf{C_i}$) and

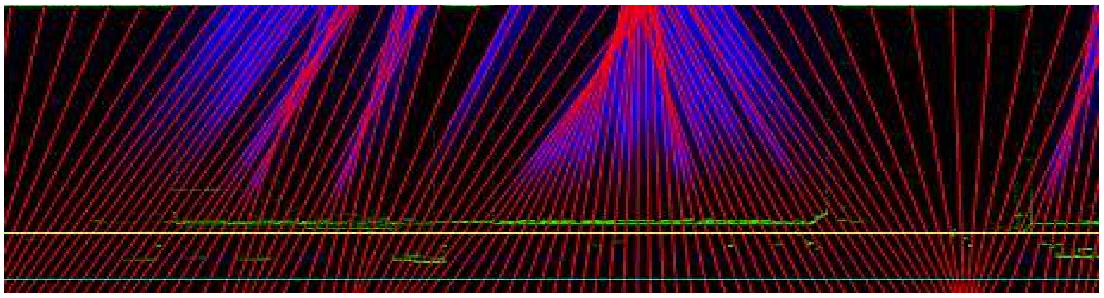
(a)



(b)



(c)



(d)

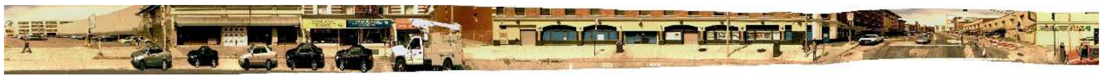


Figure 5.1: Overview of the optimization process. Given the input of a 3D point cloud (a), the 3D points (blue) are projected downward in z to form a 2D histogram (green) shown below the point cloud (a) and as an image in (b). The optimization is then performed on this histogram and the results are shown in (c). The camera path is shown in cyan along the bottom of (c), the optimized picture surface is shown in yellow, and the optimized ray directions are shown in red. The ray directions are used to create the final multiperspective image shown in (d).

orientations ($\mathbf{R_i}$) that describe the camera trajectory through the world scene. In addition, our optimization implementation assumes that the camera trajectory approximately follows a straight line.

In addition, the automated perspective optimization requires some notion of the 3D scene structure. In out implementation, this takes the form of a set of points ($\mathbf{X_j}$) that form a point cloud of the scene. For example, the sparse point cloud generated by most structure-from-motion algorithms is sufficient (Figure 5.10). External sensors such as laser range finders can also provide this information quickly and reliably (Figures 5.5 & 5.6). Alternatively, the user may manually specify important regions in the scene into a 2D importance map if the 3D structure is not appropriate (Figure 5.10). An example of the 3D point cloud can be seen in Figure 5.1.

5.2 Distortion

The most undesirable effect of perspective distortion is a change in the aspect ratio of an object, so we call it the *aspect ratio distortion*. This is caused by the crossed-slits projection and is also described in Zomet [ZFPW03b]. We first quickly review how an object projects onto the picture surface for a normal perspective projection. Then we examine how this changes for a crossed-slits projection, remembering that pushbroom and perspective are special cases of crossed-slits images. Finally, we interpret the result and show how it is consistent with intuition and real-world results.

5.2.1 Perspective Projection

Consider a linear, translating camera path as shown in plan view in Figure 5.2. The picture surface is a plane facing the camera at a fixed distance Z_0 from the camera path. A single planar object exists in the world with dimensions $W \times H$, having a canonical aspect ratio of $A = \frac{W}{H}$. This object is parallel to the picture surface at a distance Δz away. These are signed distances, and all have positive values in the example in Figure 5.2.

Under a perspective projection, the object will be imaged on the picture surface with dimensions $w \times h$ (the diagram indicates the projected width w). Using similar triangles, we

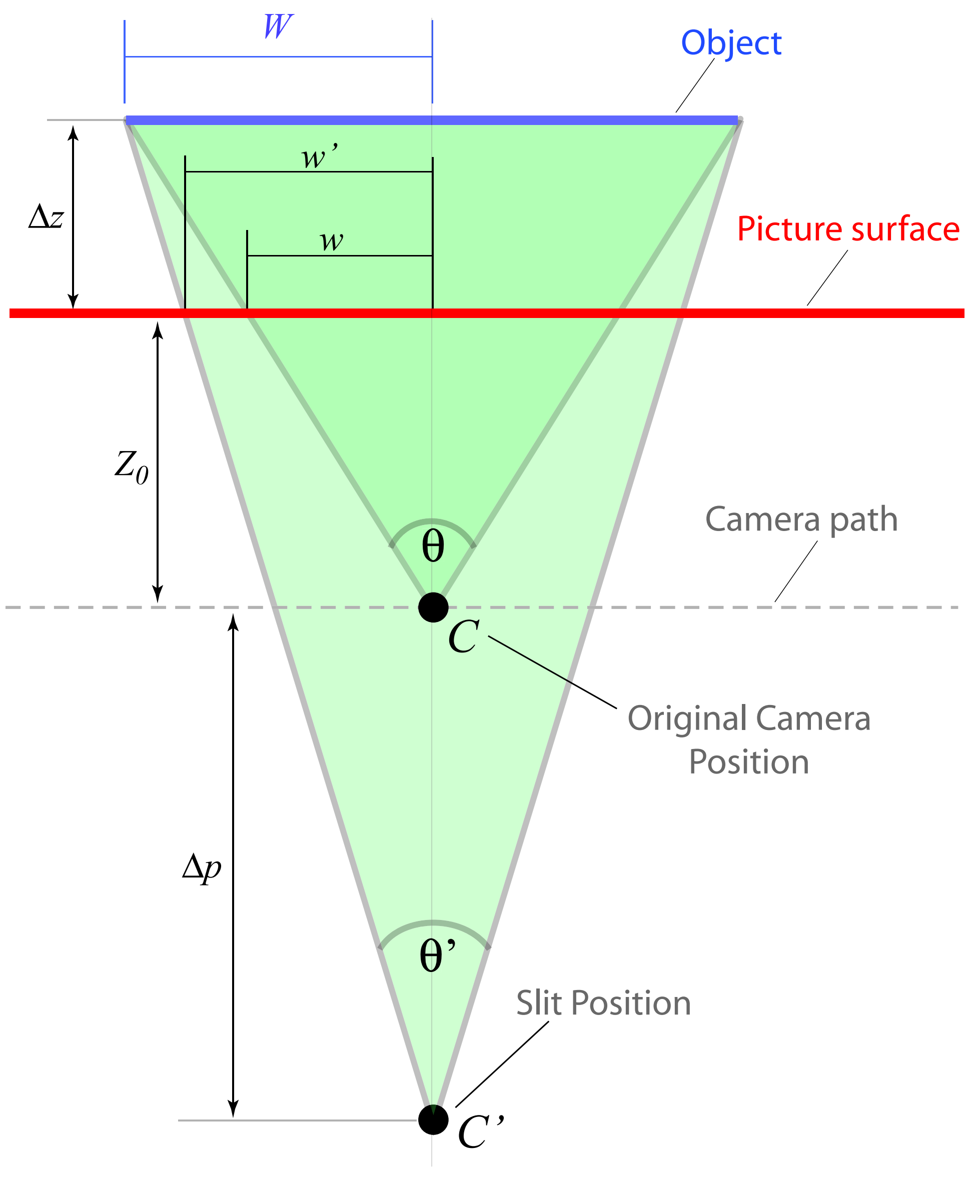


Figure 5.2: Distortion due to non-uniform perspective. This is a plan view of a simple scene consisting of only a single, planar object.

quickly see that

$$w = W\frac{Z_0}{Z_0 + \Delta z} \tag{5.1}$$

That is, the width is reduced proportional to distance behind the picture surface. For example, an object twice as far from the camera as the picture surface ($\Delta z = Z_0$) appears half as large and an object halfway between the picture surface and the camera ($\Delta z = -\frac{Z_0}{2}$) will appear twice as large. Similarly, the projected height will be

$$h = H\frac{Z_0}{Z_0 + \Delta z} \tag{5.2}$$

and so the aspect ratio of the object projected onto the picture surface will be

$$a = \frac{w}{h} = \frac{W}{H} = A \tag{5.3}$$

There is no change in the object's aspect ratio.

5.2.2 Aspect Ratio Distortion

Now consider the projection of the object in a crossed-slits image. Figure 5.2 shows the scene with a second slit placed a distance Δp away from the original camera path, changing the perspective structure horizontally. Notice that this will change the projected width of the object to w'. Again using similar triangles, we find that

$$w' = W\frac{Z_0 + \Delta p}{Z_0 + \Delta z + \Delta p} \tag{5.4}$$

Remember that vertically we still have a perspective image and so the projected height of the object will not change:

$$h' = h = H\frac{Z_0}{Z_0 + \Delta z} \tag{5.5}$$

The aspect ratio of the object under this projection is then

$$a' = \frac{w'}{h} = A\frac{(Z_0 + \Delta z)(Z_0 + \Delta p)}{Z_0(Z_0 + \Delta z + \Delta p)} \tag{5.6}$$

Case	Δz	Δp	D_a
Object on picture surface, any perspective	0		1
Object not on picture surface, normal perspective		0	1
Object not on picture surface, pushbroom		∞	$\frac{(Z_0+\Delta z)}{Z_0}$
Object at infinity, any projection	∞		$\frac{(Z_0+\Delta p)}{Z_0}$

Table 5.1: Common aspect ratio distortions. This table lists several common examples of perspective distortion in multiperspective images. Notice that if an object is aligned with the picture surface, then there is no distortion regardless of the perspective. Similarly, in a normal perspective there is no distortion regardless of the object placement. For pushbroom images, distortion is proportional to the distance from the picture surface.

We define the *aspect ratio distortion*, D_a as the change in the aspect ratio:

$$D_a = \frac{a'}{a} = \frac{(Z_0 + \Delta z)(Z_0 + \Delta p)}{Z_0(Z_0 + \Delta z + \Delta p)} \tag{5.7}$$

This is the basis of our cost function used to evaluate the perspective distortion in a multi-perspective image.

5.2.3 Discussion

We now verify that the distortion metric is consistent with several common cases (see summary in Table 5.1). Consider a perspective image where $\Delta p = 0$. The numerator and denominator are then equal and $D_a = 1$, regardless of the values of Z_0 or Δz, confirming that perspective images have no perspective distortion. Objects on the picture surface are described by $\Delta z = 0$, and again $D_a = 1$ regardless of the type or projection defined by Δp. This explains why the book cover (Figure 2.3) and the building front (Figure 5.7) suffer no distortion; in both cases they are aligned with the picture surface. For a pushbroom image, Δp approaches ∞. In this case, Equation 5.7 simplifies to $D_a = \frac{(Z_0+\Delta z)}{Z_0}$ and thus the distortion is linear with the object's distance from the picture surface.

The relation described in Equation 5.7 also applies to scenarios where the object, picture surface, and camera path are not aligned and can be evaluated by integrating the local distortion for all points across the object. Thus, it is an appropriate metric for quantifying the overall distortion in any multiperspective image.

5.3 Cost function

We want to define a cost function that converts the aspect ratio distortion into an error value. To do this, we make the following observations:

1. Under no distortion, an aspect ratio of 1 should correspond to an error of zero.

2. An object that is stretched to be twice as wide as usual should have the same error as an object that is half as wide as usual. More generally, an object with an aspect ratio of D_a should have the same error as objects with an aspect ratio of $\frac{1}{D_a}$.

We therefore define the following cost function to convert the aspect ratio distortion into an error:

$$E = \begin{cases} D_a - 1 & 1 \leq D_a \\ \dfrac{1}{D_a} - 1 & 0 \leq D_a < 1 \\ \lambda - \dfrac{1}{D_a} - 1 & -1 < D_a < 0 \\ \lambda - D_a & D_a \leq -1 \end{cases} \tag{5.8}$$

This relationship gives equal error to an object with half its normal aspect ratio and an object with twice its normal aspect ratio. Values of $\lambda > 1$ penalize negative aspect ratios where objects are horizontally inverted. We have experimentally determined $\lambda = 10$ to be appropriate to suppress any significant inversion in the optimization.

5.4 Optimization

The optimization is initialized with a pushbroom image placing the picture surface at an initial distance Z_0 from the camera path, as shown in Figure 5.3(a).

The picture surface is then discretized into N equal-length segments. Each of these segments represents a portion of the picture surface with a single type of perspective projection. The type of perspective projection is defined by the angle of the boundaries between the two segments. To enforce that the perspective across segments varies smoothly, it is

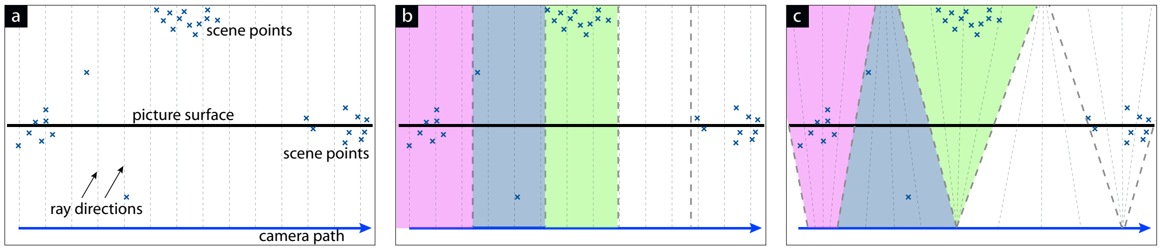


Figure 5.3: The optimization starts with an initial pushbroom image defined by a picture surface and a set of parallel rays as shown in (a). (b) The picture surface is then divided into several segments. For each segment, we compute the error of all scene points that are projected into that segment, as indicated by the shaded regions. (C) The type of perspective projection in each segment is defined by the angle of that segment's boundaries. These angles are altered to minimize the total aspect ratio distortion in the final image. The position of the picture surface can be adjusted in combination with the ray directions to further minimize distortion as described in Section 5.4.

necessary for neighboring segments to share the boundary. Therefore the perspective of the N segments can be parameterized by the $N+1$ boundaries.

These boundaries are defined in terms of the angle of the boundary with respect to the picture surface, θ_i. For example, the initial pushbroom image is described by $\theta_i = \frac{\pi}{2}$. The intersection of the two boundaries corresponds to the point C' in Figure 5.2, and thus we can directly compute the local perspective Δp for each of the segments. We can therefore compute the error of any scene point in a particular segment. The range of values for θ_i is limited by the field of view of the input imagery.

The error of a single segment is computed by summing the error from all scene points that project into that segment as indicated by the shaded regions in Figure 5.3(b). The error of the entire image is simply the sum of the errors of each of the segments. The optimization therefore finds the set of (Z_0, θ_i) that minimize the overall distortion. Optionally, Z_0 can be fixed and the set of boundary orientations that minimizes the error for the picture surface at that location can be found.

This can be described mathematically as follows. A given set of ray directions θ_i $(i = 0 \ldots N+1)$ defines N segments S_i and corresponding local perspectives Δp_i. Within

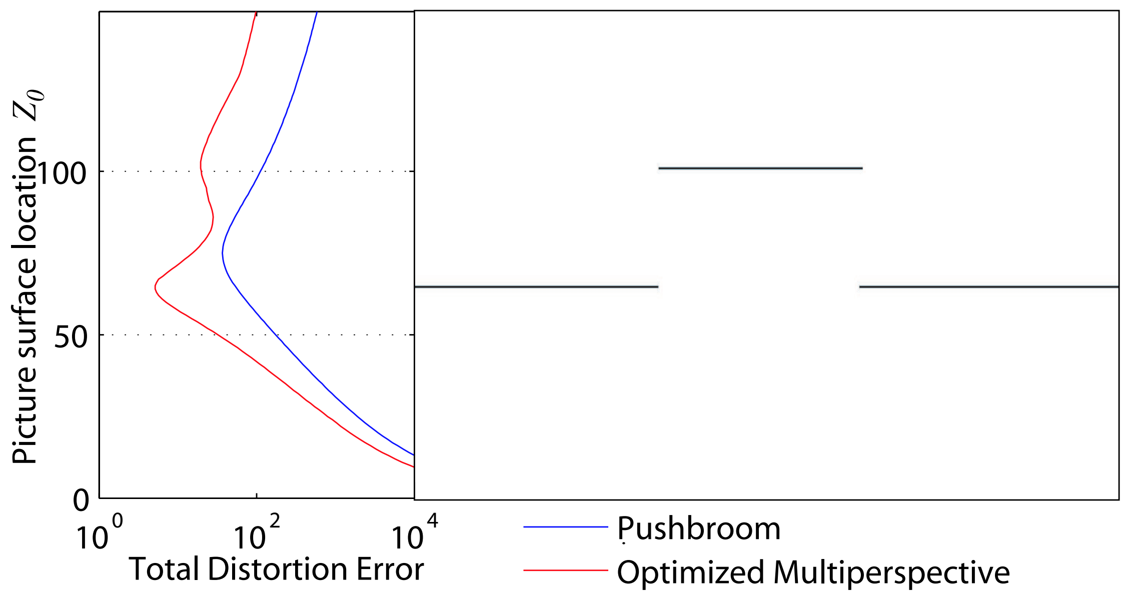


Figure 5.4: This diagram shows how the total distortion error of a multiperspective image depends on the picture surface location Z_0. On the right is the importance map for a simple synthetic scene consisting of only three planes. The graph on the left shows the error of both the initial pushbroom configuration and the optimized multiperspective configuration when the picture surface is fixed at the corresponding row in the importance map.

a particular segment, Δp and Z_0 are constant, and therefore the error of a point q is described by $E(D_a(\Delta z_q, \Delta p_i, Z_0))$ where Δz_q is the orthogonal distance of q to the picture surface. The minimization therefore is:

$$\underset{\theta_i, Z_0}{\operatorname{argmin}} \left(\sum_S \sum_{q \in S_i} E(D_a(\Delta z_q, \Delta p_i, Z_0)) \right) \tag{5.9}$$

5.5 Implementation

The optimization was implemented in C++ and used the Opt++[Mez94] numerical optimization library to perform a bounded Newton optimization. There are several techniques used to make computing the error function faster and easier.

Instead of manually computing the derivative of the cost function, we take advantage of an automatic differentiation technique described by Piponi[Pip04] which is faster and more accurate than numerical differentiation. When evaluating the aspect ratio within a given segment, Δp_i is determined by the intersection of the two ray directions of the segment:

$$Z_0 + \Delta p_i = -\frac{D}{2}\left(\frac{\cos(\theta_i + \theta_{i+1}) - \cos(\theta_{i+1} - \theta_i)}{\sin(\theta_{i+1} - \theta_i)}\right) \tag{5.10}$$

where D is the length of the picture surface segment. Unfortunately, this can cause a divide-by-zero error in the common situation that the rays are parallel. We avoid this intermediate computation by substituting this expression into the equation 5.7 and simplifying to obtain:

$$D_a = \frac{Z_0 + \Delta z}{Z_0}\frac{1}{K+1} \tag{5.11}$$

where

$$K = \frac{2\Delta z}{D}\left(\frac{\sin(\theta_{i+1} - \theta_i)}{\cos(\theta_i + \theta_{i+1}) - \cos(\theta_{i+1} - \theta_i)}\right) \tag{5.12}$$

Notice that the distortion in (5.7) depends only on the depth (Δz) and local perspective (Δp) of that point. It does **not** depend on the height of that point above the ground plane. We can therefore take the input 3D scene geometry and project the points down onto the ground plane, removing the z component. We then quantize the x and y values into bins and count the number of points in each bin creating a 2D histogram as shown in Figure 5.3(a). Instead of searching for scene points that fit within each segment, we simply compute the bounds of the segment within the histogram and compute the error for that region.

This histogram is simply an image analogous to an overhead density map of the scene. An example can be seen in Figure 5.1. By default all scene points contribute equally to the error function. The user may optionally augment the histogram with a 2D importance map that modulates the histogram. This allows the user to manually emphasize or de-emphasize regions of the scene. We have used this in Figure 5.10 to guide the optimization toward important regions of the scene.

Assuming the picture surface is aligned with the x axis of the histogram, the distortion is constant along x within each segment. We can efficiently integrate the contribution of each row using a summed area table [Cro84].

To avoid local minima, we perform a multiresolution optimization. Both the importance

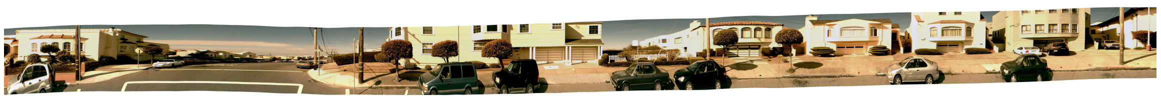


Figure 5.5: A portion of a long multiperspective image spanning 2 km of 18th Ave in San Francisco, south of Golden Gate National Park. The perspective varies continuously along this image, enabling an arbitrarily long seamless panorama. The perspective is automatically computed to minimize aspect ratio distortions in regions that have large depth variation such as road intersections. The image isn't perfectly horizontal because the city itself has hills and they are captured in the panorama.

map and the number of segments along the picture surface are reduced hierarchically.

These implementation techniques make the optimization very fast. On a dual Xeon 3.2GHz PC with 1GB of ram, the entire city street example in Figure 5.5 takes just over 5 minutes (302 seconds) to optimize, specifying the varying perspective for a 600 megapixel image (325k pixels wide).

5.6 Results

We have applied our technique to indoor and outdoor scenes: a room inside a museum (Figure 5.10), Mission Street (presented in Figure 5.6), and 18th Ave (Figure 5.5) in San Francisco. The museum scene was acquired moving a sideways looking video camera along a straight line parallel to the scene and spans approximately 20m. The camera path was extracted from the video using the freely-available Voodoo Camera Tracker [Uni03] structure-from-motion software which also outputs a sparse 3D point cloud. The street scenes were captured using a sideways-looking, high-speed camera (Basler A504kc) in a car driving in normal traffic (0-20mph). The camera pose was estimated using accelerometers and GPS via a Kalman filter. The 3D scene structure was acquired using time-of-flight range finders. The Mission street image spans about 860m while the 18th Ave image spans about 2088m. It is possible to use SFM to generate the required projection matrices and scene estimates for the street scenes, however many SFM algorithms do not handle extremely long, linear scenes robustly.

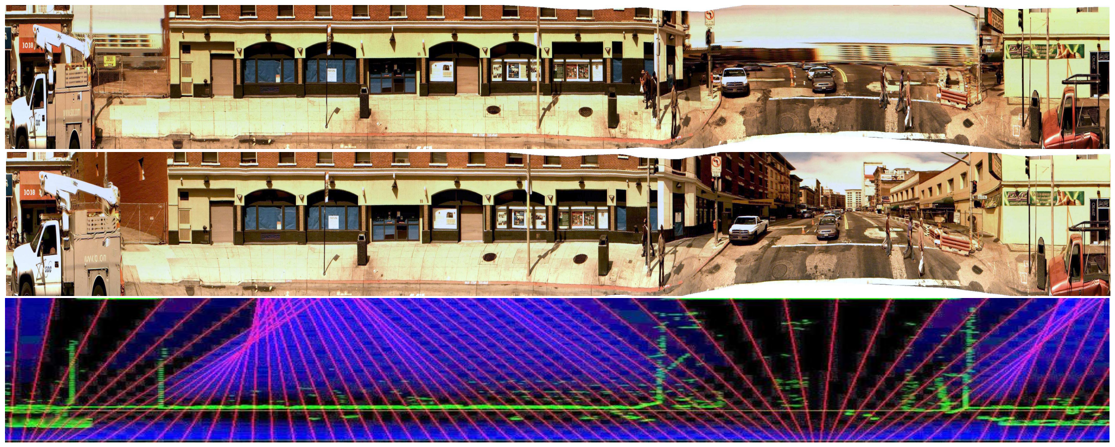
(a)
(b)
(c)


Figure 5.6: Multiperspective images generated automatically from a video stream with lateral movement: (a) A pushbroom image, which uses the perspective provided by the input stream in the *y*-direction and an orthographic projection in *x*, in order to combine the information of all frames. Notice the difference in perspective in *x* and *y* leads to severe distortion at the intersection and the alleyway. (b) A multiperspective image generated automatically using our technique. While the perspective in *y* is still the same, we optimized the perspective in *x* in each image segment in order to minimize distortion. Notice that this is an image with multiple perspectives – there is a vanishing point down the alleyway and a separate vanishing point down the intersection. (c) A plan view of the street showing the optimized ray directions (red). You can see how the rays nearly converge to a perspective at the intersection and again near the alleyway. The yellow line denotes the picture surface. The blue channel is a visualization of the cost function over the entire space. Notice that this set of ray directions minimizes the intersection between the scene points (green) and the error (blue).

Table 5.2 summarizes the scene size, the number of input frames, number of optimization segments, output sizes, and the timings for the perspective optimization. The only user-selectable parameter is the number of segments to optimize, which should be chosen according to the scene length. Due to the hierarchical optimization and the use of summed area tables the optimization performs well even for the larger street scenes.

5.6.1 Discussion

In all three scenes the artifacts due to aspect ratio distortion after optimization have been reduced to a minimum compared to the pushbroom panoramas. We will now focus on the

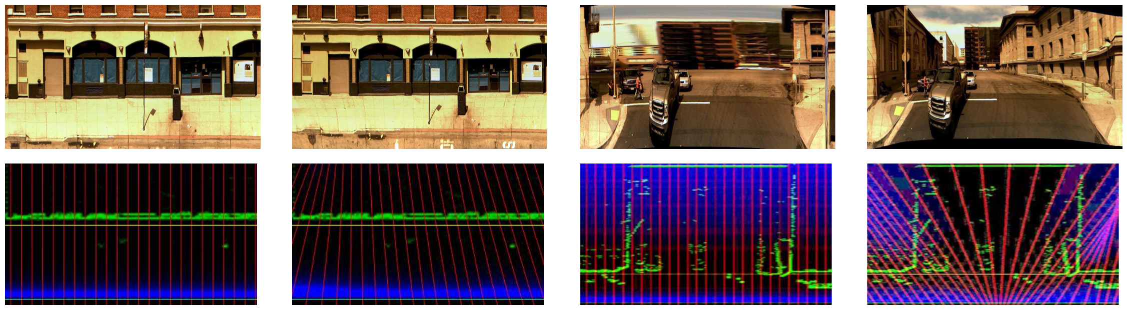

(a) (b) (c) (d)

Figure 5.7: These figures indicate how the image is effected by changes in Δz. The bottom row shows the ray directions (red), the picture surface (yellow), the scene histogram (green) and the distortion error (increasing shades of blue). Portions of the scene that are aligned with the picture surface, such as the store front in (a) and (b), are not affected by the ray directions. In contrast, the error of regions that have significant depth variation, such as in (c) and (d), is sensitive to the ray directions.

scene	size (in m)	# input frames	histogram size	# segments	output resolution	optimization (in min.)
Museum Scene	20	941	896×425	64	2522×438	0:35
Mission Street	860	21520	2482×183	512	61320×1000	2:16
18th Ave	2088	61092	7312×200	512	325240×11538†	5:02

Table 5.2: Facts about the different scenes. The optimization is fast even for large scenes. † Because 18th Ave is not flat, large sections of this image are blank. The actual image area is approximately 600MP instead of 3.4GP.

performance of the optimization by analyzing special cases in the Mission St. panorama. Figure 5.7 visualizes the dependence of the error function with regard to the placement of the picture surface z_0. If the scene is at the picture surface (Fig. 5.7(a) and (b)) there is no aspect ratio distortion, no matter which perspective is selected. For surface points off the picture surface, (c) and (d), the error can only be minimized by approaching the original camera perspective. If the area of the depth deviation is too large to be covered by a single input image, as in case of Figure 5.8, our optimization resorts to the closest crossed-slits perspective that spans the entire gap.

The proposed error metric only accounts for aspect ratio distortion. This has the effect

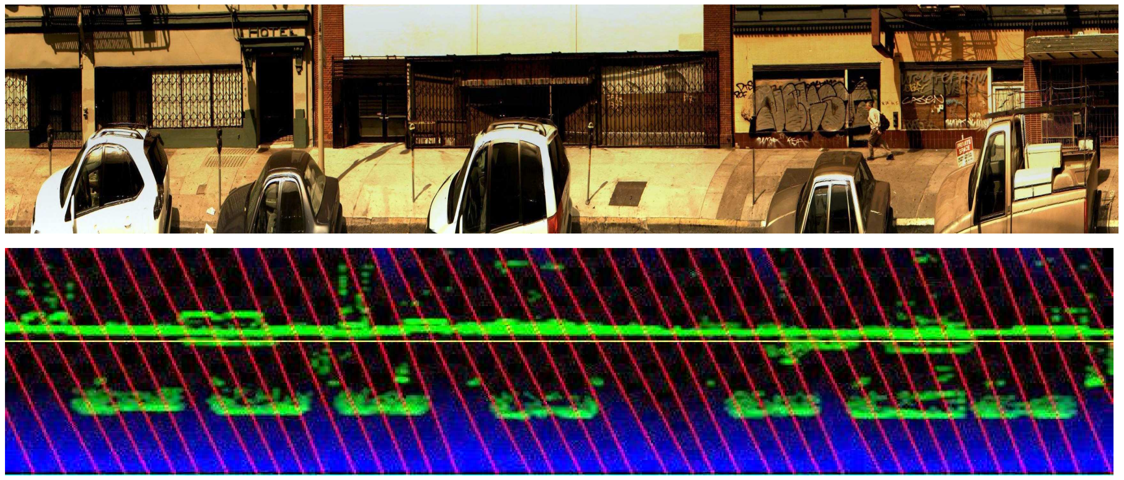


Figure 5.8: Limitations of our algorithm. The optimization is unable to eliminate the distortion of the cars because the cars form a continuous region with large depth variation. The best solution therefore is an extreme crossed-slits – approaching a pushbroom perspective across the entire region. Furthermore, our error metric does not account for shear in the projection. The shear is influenced by neighboring regions not shown.

that a sheared perspective contributes the same error as a more symmetric setup. In Figure 5.8 all cars are shown from an oblique view. Notice that the oblique view in fact does not introduce any further distortions. Our optimization does not prefer one over the other and has the freedom to chose whatever shear fits best in order to optimize for neighboring regions.

Our approach is unaware of occlusions (Figure 5.10(bottom)). In this case our algorithm during optimization may consider the error even for an object that will be occluded in the final output. An optimization considering occlusions would have some impact on the resulting shear. One expects that the shear will be chosen such that foreground objects occlude as many scene points which are off the picture surface as possible.

An artifact due to an incorrect estimate of scene geometry is presented in Figure 5.9. Because of the limited range of the 3D range finders the building in the background does not show up in the depth histogram and the algorithm allows some rays to cross in front of the building resulting in multiple copies of the building in the output image. In the pushbroom image the building is visible only once.

The method uses a rough estimate of the depth variation in the scene, currently in the

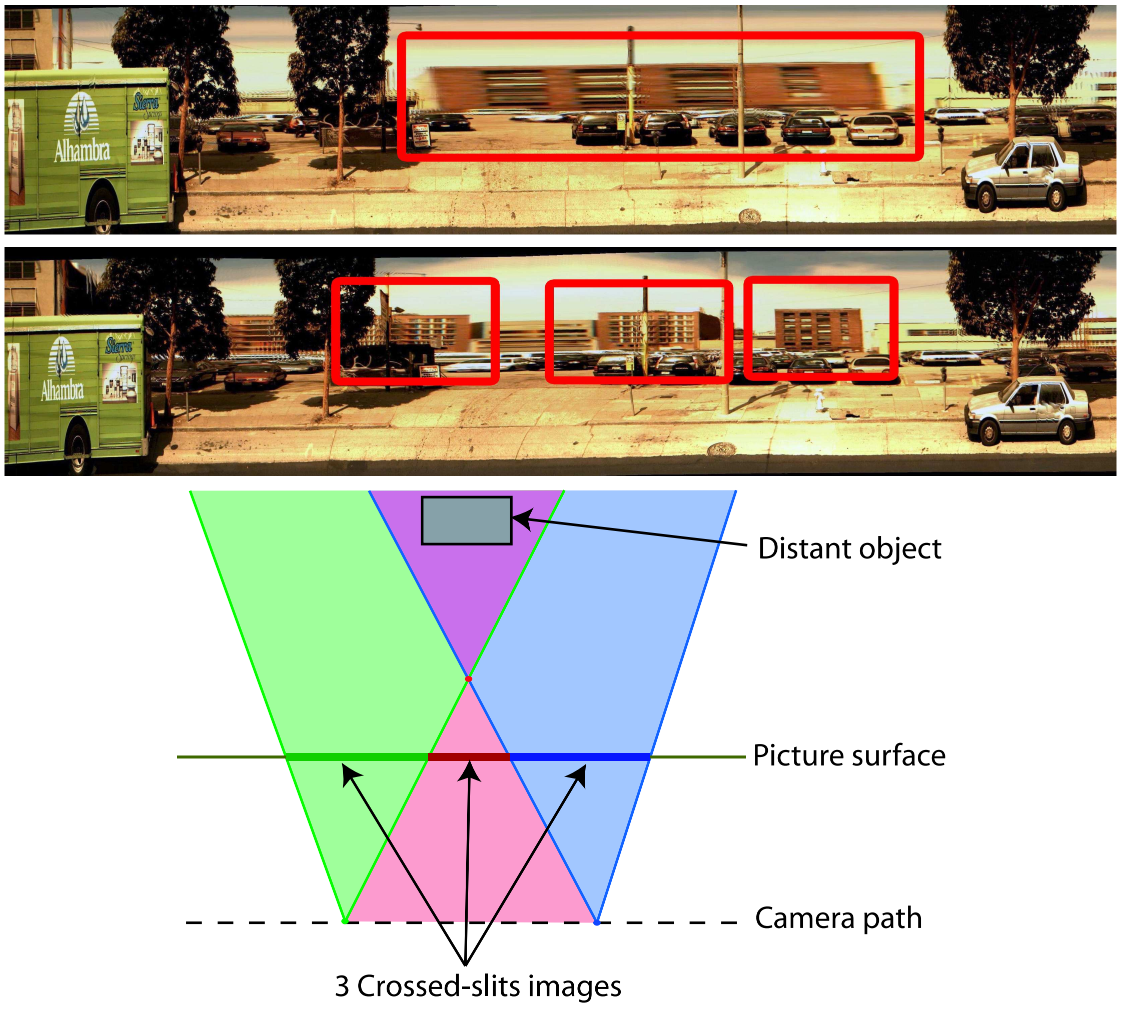
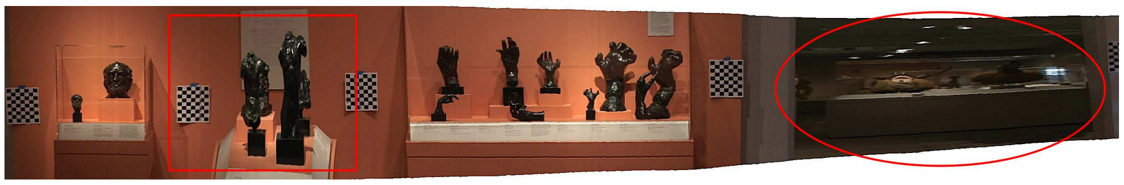


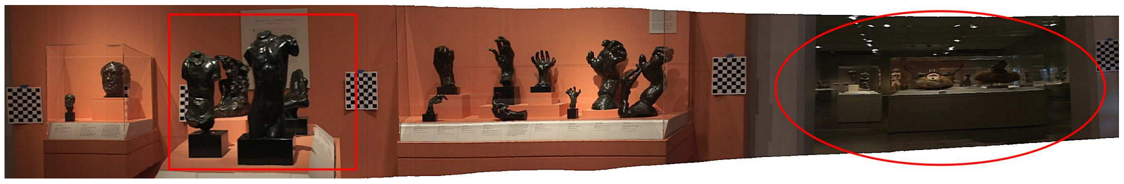
Figure 5.9: Limitations of our algorithm. On top is a pushbroom image showing a single, distant, distorted building. Unfortunately the building was too far for the 3D scanner to detect and therefore was not considered in our optimization. The resulting optimized rays cross in front of the building causing a triple image to occur, highlighted in the second image. This effect is shown in the bottom diagram. Objects that are behind the the intersection of rays appear in three separate images. In the diagram, the object will appear as perspective in both the blue and the green images. The intermediate red crossed-slits image will also show the object, however the object will be horizontally reverse. The final result is a triple image of the object.

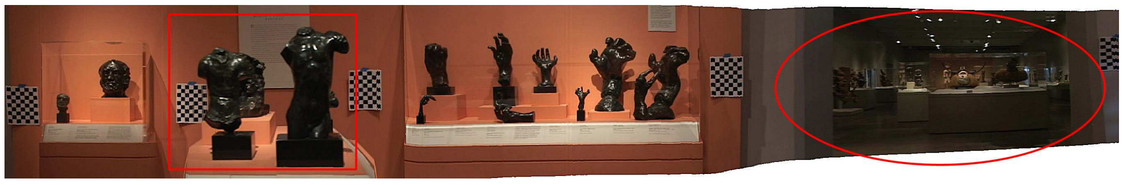
form of a histogram of scene points in the *x*-*z* plane. One can easily modify the influence of particular scene objects on the optimization by manipulating their contribution to the histogram. By analyzing the input sequence one might be able to perform an object segmentation and determine which scene parts are important to have minimum distortion. Similarly, one could think of detecting and emphasizing scene regions with highly regular textures for which aspect ratio distortion produces a higher visual impact than for non-textured regions.

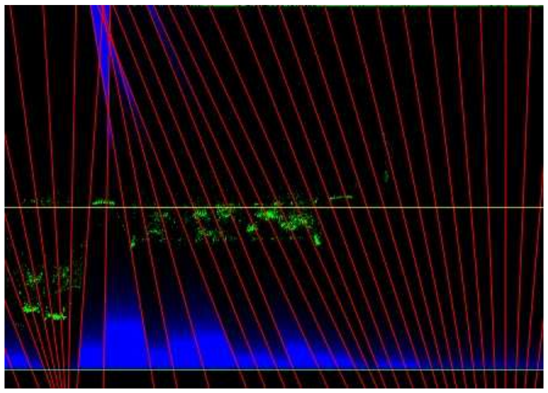
In the future we plan to extend our work to correctly handle occlusions. Instead of evaluating the error metric on a projected depth histogram one could evaluate it for each rendered pixel in the final output image. This way, pixels that are occluded would not contribute to the overall error. While this approach might yield even more precise results it is inherently much costlier to compute than our presented technique.

Another aspect which is interesting to investigate is to allow for non-planar picture surfaces which could be used to emphasize or enlarge specific features in the scene. However, it is not yet clear what kinds of artifacts will be introduced by the change in sampling resolution and the resulting change in relative size of real-world objects in the final output. While our optimization algorithm is flexible enough to handle even curved picture surfaces our current error metric does not account for this kind of distortion introduced by the variation in the output sampling.


(a)


(b)


(c)


(d)

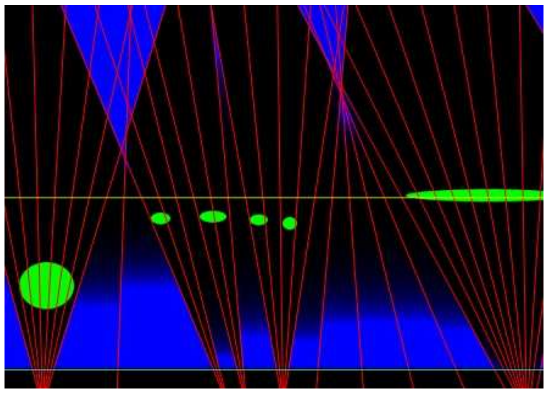

(e)

Figure 5.10: Museum scene. (a) Pushbroom. The sculptures on the left and the hallway on the right are severely distorted. (b) Automatic perspective. The optimization reduces the distortion but is unable to distinguish artwork and other scene content. There is still some distortion in the hallway. (c) Importance map. Adjusting the histogram to emphasize the relevant artwork results in less distortion for these objects. However, the overall geometric distortion has been increased. Notice that our algorithm does not consider effects of occlusions as seen in the sculptures on the left.

Chapter 6

Discussion

The ideas presented in this thesis are only a start toward the task of digitizing an entire city. In this chapter, I will discuss important extensions and considerations in extending this work to a larger scale.

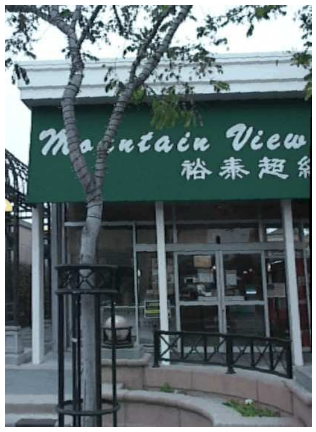
6.1 Input requirements and rendering

6.1.1 Implementation

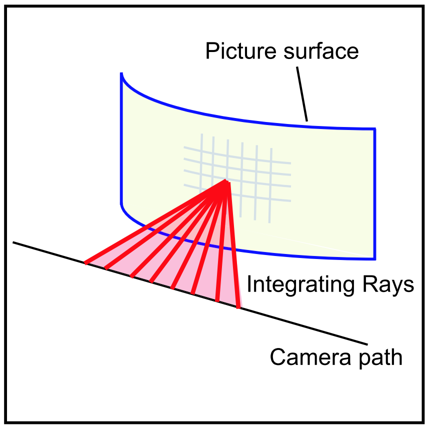
The dense input video requirements are the most stringent requirement for the results described in this thesis. Capturing this input video on a large scale is certainly an engineering challenge. In our case, we used a high-speed camera so that we can acquire images as dense as an image every centimeter even while driving at speeds of up to 10 mph. However, in the common case that speeds above 10 mph were necessary, the view interpolation schemes described in Section 4.3 were used successfully.

6.1.2 Relaxing the requirements

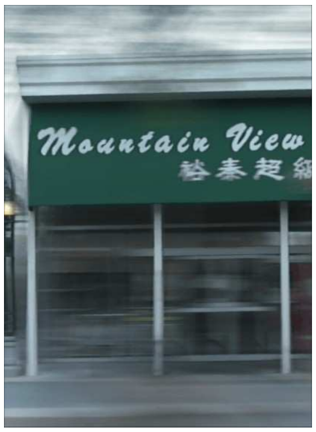
A key concept in this work is the reliance on using the linear crossed-slits camera projection and compositing those projections to form the final multiperspective image. This representation has many benefits as described in Section 2.1.3, however it suffers from the



(a) Perspective image



(b) Integration on picture surface



(c) Synthetic focus image

Figure 6.1: Here we show an extension of our rendering algorithm to generate synthetically focused images. By integrating over an angular arc for each point on the picture surface as shown in (b), we can simulate a large 1D aperture focused at the picture surface. (c) is an example of such a synthetically focused image with the focus fixed at the plane of the facade. Note that the tree present in (a) is blurred out.

requirement of dense input video imagery. In recent literature, there have been a number of related research publications that suggest alternatives or workarounds to this limitation.

For example, Agarwala et al. [AAC+06] have described a graph-cut rendering algorithm that chooses nonlinear seams between images and minimizes the distortion along the seam. By using this approach, they have demonstrated that with properly aligned input images they can reconstruct a high resolution, high quality photomosaic. Unfortunately, their algorithm does not robustly handle significant depth variation. One could imagine integrating our distortion metric into their seam-selection cost function to improve the output.

6.1.3 Extensions

There are a number of interesting extensions to the collected data that could improve both the utility of the dataset and quality of the resulting images.

Synthetic Aperture Photography

All of our examples have an effectively infinite depth of field (assuming pinhole cameras for input). By averaging multiple input video frames projected onto the picture surface, we can simulate a synthetic aperture as discussed in Vaish et al. [VWL04]. This results in an extremely shallow depth of field image. Because our input dataset is only 3D (as opposed to a 4D lightfield), our synthetic aperture is only 1D, whereas the apertures in [VWL04] are 2D. Figure 6.1 shows an example of this effect. Synthetic aperture imaging could be used both to enhance a relevant portion of the scene (e.g. deciphering partially occluded text) or to blur out an undesireable portion of the scene (e.g. blur out people to protect privacy).

Field of view trade-off

The trade-off between the image resolution and the field of view of the input imagery is a difficult one to handle. On the one hand, having a very wide input field of view increases the flexibility of the optimization in choosing viewpoints for regions of the image. For example, if the input imagery has a sufficiently wide field of view, even a very wide street intersection could conceivably come from a single input image, completely removing any distortion. On the other hand, a wider field of view inherently reduces the detail for each pixel throughout the image.

It would be interesting to investigate using a non-uniformly sampled image sensor (or equivalently, a nonuniform lens) that allows both a very wide overall field of view but similarly keeping a high-resolution central region on the assumption that the straight-on view is preferable. So far, this effectively describes a fish-eye lens. For this particular application, however, a lens that is horizontally fish-eye only might be preferable since we are taking only vertical strips from each input image.

Multiple cameras

It's possible to combine images from multiple cameras. For example, instead of using only a single video camera mounted on a vehicle, consider mounting an entire column of video cameras. As the vehicle moves down the street, the column of video cameras will acquire a full 4D lightfield. A similar dataset can also be acquired by driving multiple times with a camera at different heights, although in this case there is the additional problem of combining multiple datasets taken at different times (see Section 6.3 below).

Once you have a full 4D lightfield, there are many interesting possibilities that open up. For example, you can construct fully perspective images (rather than crossed-slits) that are set much further back, removing distortion caused by the difference between the vertical and horizontal perspectives. In addition, image-based rendering techniques could be used to virtually explore the region.

6.2 Extending the optimization

There are a number of assumptions in the described optimization implementation that restrict the applicability of this research. Here I want to briefly examine what these assumptions are, how they can be overcome, and other interesting extensions to this work.

The primary restriction lies in the constraint that the picture surface must be parallel to the original camera trajectory. This restriction is not imposed by any of the theoretical constructions of our optimization, but rather by our particular implementation and is a result of the desire to have a very fast optimization that can be run quickly even over massive datasets. Without the projection to a ground plane, instead of using a histogram to represent the scene, the full list of 3D points can be used. This will also lose the summed-area-table implementation speedup, however the optimization can still be run simply by applying the error function to all 3D points that lie between the two planes that represent the boundaries of the crossed-slits projection. All of this only affects the speed of evaluating the cost function for a particular arrangement of crossed-slits projections, not the optimization itself.

A second major restriction in the optimization is that the picture surface is assumed to be vertical. While this is appropriate for most urban environments, it is possible that non-vertical picture surfaces may be desired. If the orientation of the picture surface is known, the optimization can be implemented by computing the cost function from the full list of 3D points and taking into account the orthogonal distance between each point and the picture surface. Ideally, the orientation of the picture surface would be included in the optimization.

6.3 Combining multiple datasets

A new and very interesting challenge introduced by this visualization technique is how to remove undesirable occluders by combining multiple datasets. In particular, imagine that the first dataset images the first part of the street satisfactorily but a large truck occludes the second part of the street. On a second pass, the second part is imaged satisfactorily but the first part is occluded. There are a number of challenges involved in combining these two datasets, even in the simplest case where you might simply want to take half from the first and half from the second.

6.3.1 Registration

In order for two datasets to be seamlessly combined, they must be correctly registered to each other. The case of visualizing an urban environment, it is reasonable that the datasets should all be registered to a global reference frame, for sample GPS-based. However, GPS measurements typically only accurate to within a few meters, or a few centimeters if differential GPS is used. Worse, GPS measurements are especially noisy in urban environments because of the reflections off buildings and it is therefore often very difficult to get precise global positioning between two datasets.

Instead, it is likely to be necessary to register the datasets themselves, either by scan-matching the LIDAR data or by using image-based alignment such as structure-from-motion (SFM) algorithms that track features across both sequences. For the feature tracking, special care will have to be taken because the two sequences are likely to violate many of the common assumptions such as static scenes and constant lighting, as described in sections 6.3.2 and 6.3.3 described below. Sand et al [ST04] have demonstrated a robust video registration technique that is resilient to changes in motion, timing, and lighting.

6.3.2 Occluders and moving objects

Video and LIDAR data of the same region captured at different times have some interesting properties. Assuming that this data can be properly registered, this provides the ability to quickly and easily segment out transient objects such as people, delivery trucks, other moving vehicles, and other undesirable occluders. Any significant differences between the data make it difficult to use in the registration process, however.

6.3.3 Lighting compensation

A significant challenge in combining multiple datasets taken at different times into a single datasets is in the rendering. Specifically, how to combine image data from both datasets while avoiding ugly seams between the images. This is especially difficult when the datasets are collected at different times of the day, and will therefore have different lighting conditions. Lighting conditions can drastically affect the color and appearances of objects as shown in Figure 6.2.

6.3.4 Other sensing modalities

It is also possible to combine this data with completely separate collection sensors, such as aerial LIDAR and aerial photography. This has been investigated by Früh and Zakhor [FZ04]. Automatically combining street-level imagery with aerial imagery provides a convenient mechanism for handling tall buildings that exceed the field of view of the ground-based input video camera.

6.4 Privacy

The possibility of having public urban areas automatically imaged and easily publicly accessible opens up new concerns about privacy. It may be necessary to remove people to protect their identities if the resulting data is to be publicly accessible. If this is the case, then it is not desirable for this process to be done manually. Instead, it would be better to automatically detect and remove or blur people out.

6.5 Segmentation

In addition to applying special processing to people in order to protect privacy, it is desirably to detect and segment out other world objects in a semantic sense. This can enable

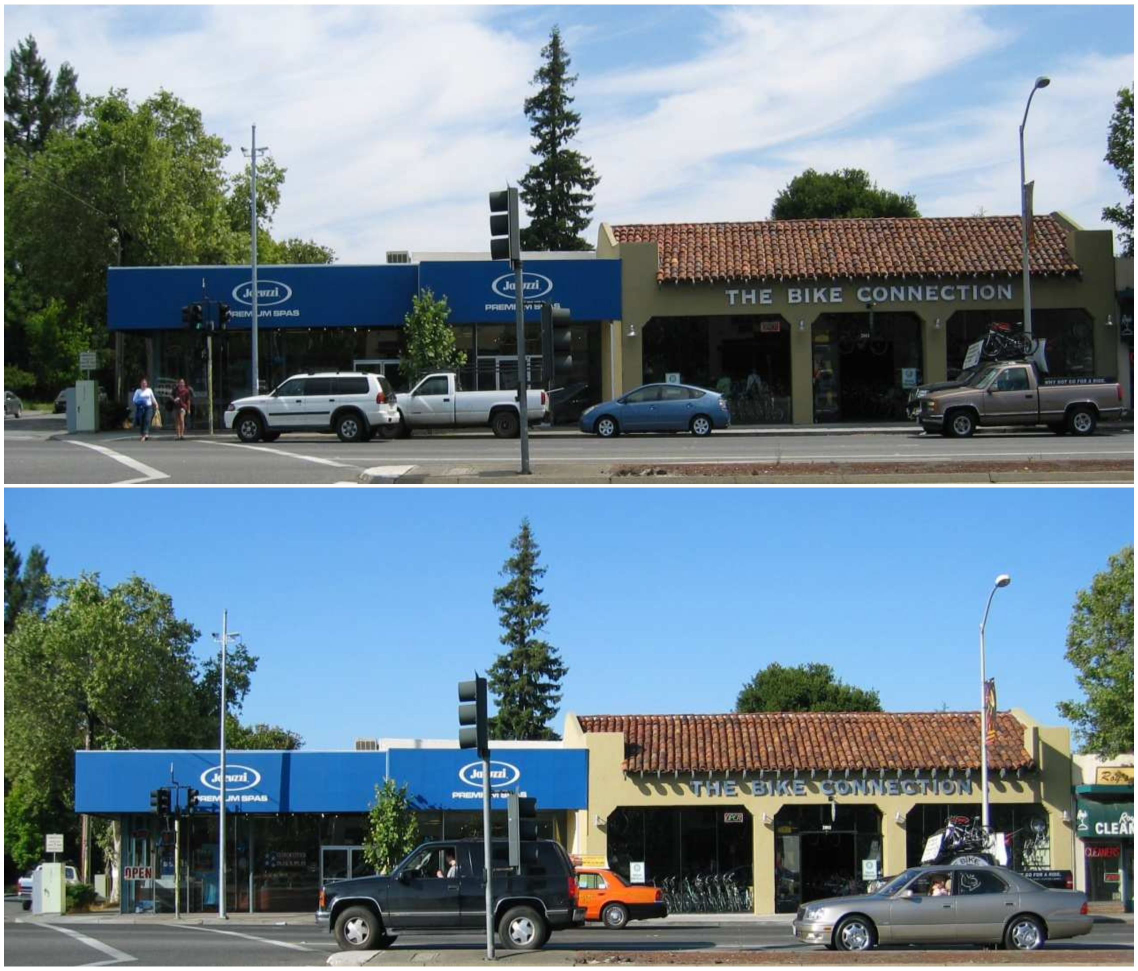


Figure 6.2: This figure shows how variation in lighting conditions can drastically affect the appearance of a region. Both of these images were taken in approximately the same location and of the same scene only a few hours apart. Notice in the top image the cloudy sky and soft shadows, along with the dark blue and beige colors of the storefronts. Contrast that with the bottom image. The sky is now a clear, deep blue. The storefronts are more vibrant with entirely different shades of blue and a light tan color. Also, there are now strong, sharp shadows. Also, notice that the differing angle of the sunlight causes the opposite highlights on the tree on the far left. Finally, the reflections in the stores of the two images are vastly different, causing significant differences in the visibility into the stores between the two images.

separate rendering algorithms for objects at varying depths, for example.

6.6 After the image

There is much interesting work to be done after a satisfactory multiperspective image is created. Optical character recognition (OCR) could be used to automatically extract street addresses or store names from the resulting image. Notice that extracting information from the multiperspective image results in a substantial decrease in the amount of image data that must be processed compared to the original video sequence. In addition, because the picture surface is defined in world coordinates, external knowledge of the location of the picture surface can guide text extraction. For example in Figure 5.6: Because it's known that this is Mission St. in San Francisco, a list of stores that are known to be on that street can be used to constrain text extraction. Many types of metadata could be extracted in a similar fashion: street names and addresses, parking information, store hours, etc.

The presentation of the multiperspective image also provides a unique challenge. As demonstrated by the example in Figure 5.5, these images can be extremely wide—much wider than generally available display technology—and require special consideration for presentation. Multiresolution display approaches similar to Google Maps [Goo06a] that allow a user to quickly and naturally browse enormous images seem like an appropriate interface.

An alternative to viewing these as purely 2D images is to combine with with aerial or satellite imagery. As noted in the introduction, one of the drawbacks of aerial imagery is that it does not adequately represent the view a user expects when standing at a particular location. These multiperspective images do represent that view. Combining the two views effectively poses a challenge. Another approach entirely is to incorporate the images into abstract maps as in the map in Figure 6.3.

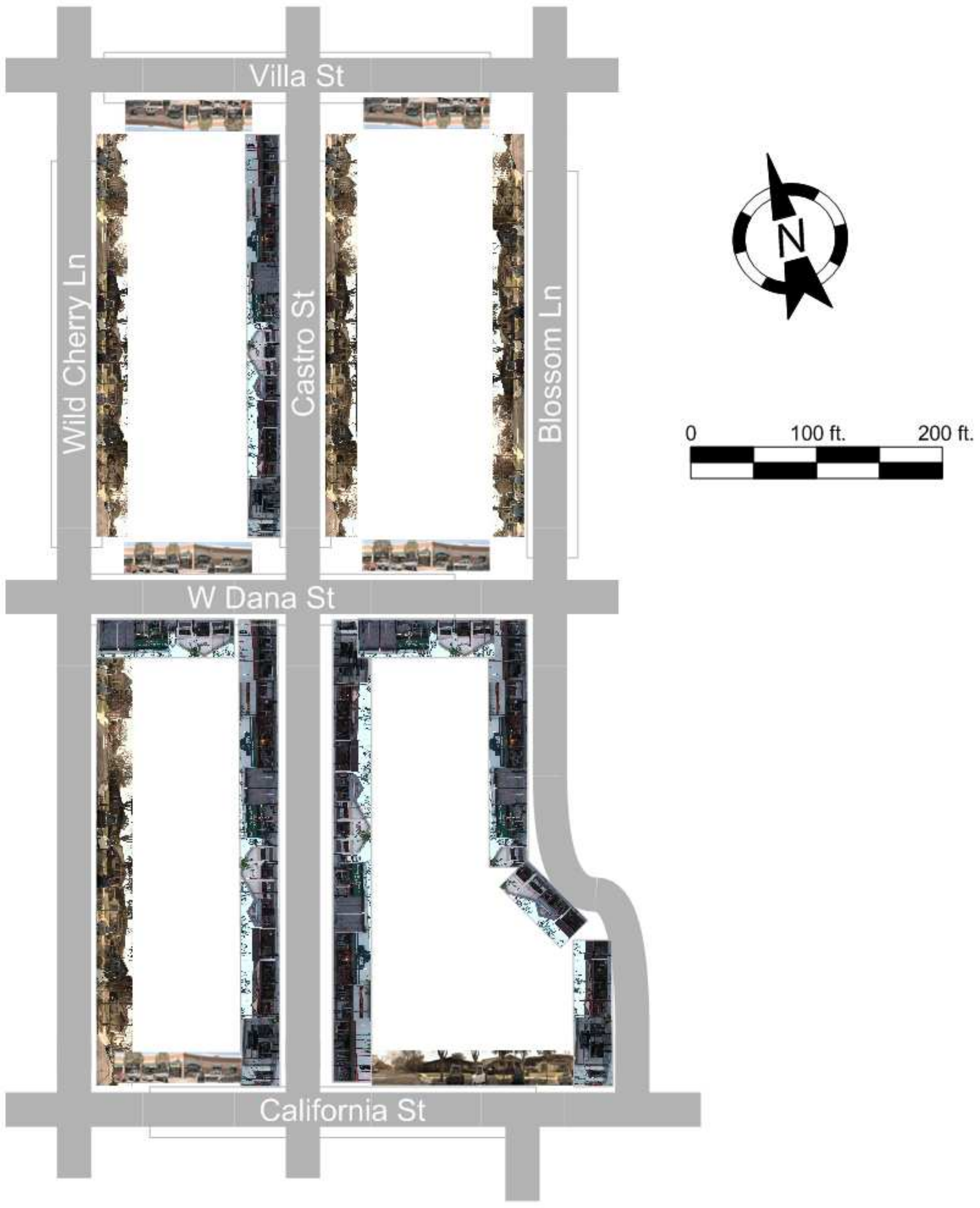


Figure 6.3: A mockup map that embeds multiperspective images of street blocks into a traditional city map. With this map, a person can see the actual storefronts along the street and more easily locate a desired store. This can also help a person locate themselves by comparing their view to the map images.

Bibliography

[2D306] 2D3. Boujou. http://www.2d3.com, 2001-2006. 33

[AAC+06] Aseem Agarwala, Maneesh Agrawala, Michael Cohen, David Salesin, and Richard Szeliski. Photographing long scenes with multi-viewpoint panoramas. *ACM Trans. Graph.*, 25(3):853–861, 2006. 8, 70

[BA93] Michael Black and P. Anandan. A framework for the robust estimation of optical flow. *Fourth International Conference on Computer Vision (ICCV)*, pages 231–236, 1993. 40, 41, 42

[Bla92] Michael Black. *Robust Incremental Optical Flow*. PhD thesis, Yale University, Sep 1992. 40, 41, 42

[Che95] Shenchang Eric Chen. Quicktime VR: An image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer Graphics and Interactive Techniques*, pages 29–38. ACM Press, 1995. 4

[Cro84] Franklin C. Crow. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 207–212, New York, NY, USA, 1984. ACM Press. 61

[CT01] Nelson Siu-Hang Chu and Chiew-Lan Tai. Animating chinese landscape paintings and panorama using multi-perspective modeling. In *CGI '01: Computer Graphics International 2001*, pages 107–112, Washington, DC, USA, 2001. IEEE Computer Society. 7

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. *Computer Graphics*, 27(Annual Conference Series):279–288, 1993. 40

[DM97] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 369–378, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 46, 47

[FLW02] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 249–256, New York, NY, USA, 2002. ACM Press. 48

[FTV00] A. Fusiello, E. Trucco, and A. Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12(1):16–22, 2000. 43

[FZ04] Christian Früh and Avideh Zakhor. An automated method for large-scale, ground-based city model acquisition. *Int. J. Comput. Vision*, 60(1):5–24, 2004. 2, 74

[GH97] Rajiv Gupta and Richard I. Hartley. Linear pushbroom cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):963–975, 1997. 7, 14

[GKG04] Indra Geys, Thomas P. Koninckx, and Luc Van Gool. Fast interpolated cameras by combining a gpu based plane sweep with a max-flow regularisation algorithm. *Second International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT'04)*, pages 534–541, 2004. 40

[Gla00] Andrew S. Glassner. Cubism and cameras: Free-form optics for computer graphics. *Microsoft Research Technical Report MSR-TR-2000-05*, 2000. 7

[Goo06a] Google. Maps. http://maps.google.com, 2002-2006. 76

[Goo06b] Google. Earth. http://earth.google.com, 2004-2006. 1

[HG94] Richard I. Hartley and Rajiv Gupta. Linear pushbroom cameras. In *Proceedings of the third European conference on Computer Vision*, pages 555–566. Springer-Verlag New York, Inc., 1994. 5

[HZ04] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004. 12

[Kol04] Michael Koller. Seamless city. http://www.seamlesscity.com, 2004. 5

[KUWS03] Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High dynamic range video. *ACM Trans. Graph.*, 22(3):319–325, 2003. 46

[Mez94] J. C. Meza. Opt++: An object-oriented class library for nonlinear optimization. Technical report, Sandia National Laboratories, Livermore, CA, March 1994. 60

[Nay97] Shree K. Nayar. Catadioptric omnidirectional camera. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 482–488. IEEE Computer Society, 1997. 2

[Ope92] OpenGL Architecture Review Board. *OpenGL Reference Manual: The official reference document for OpenGL, Release 1*. Addison Wesley, 1992. 11

[Pip04] Dan Piponi. Automatic differentiation, c++ templates, and photogrammetry. In *The Journal of Graphics Tools*, volume 9, 2004. 61

[PRRAZ00] Shmuel Peleg, Benny Rousso, Alex Rav-Acha, and Assaf Zomet. Mosaicing on adaptive manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1144–1154, 2000. 5

[RB98] Paul Rademacher and Gary Bishop. Multiple-center-of-projection images. In *Proceedings of the 25th annual conference on Computer Graphics and Interactive Techniques*, pages 199–206. ACM Press, 1998. 6

[SD96] Steven M. Seitz and Charles R. Dyer. View morphing. *Computer Graphics*, 30(Annual Conference Series):21–30, 1996. 40

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, New York, NY, USA, 1985. ACM Press. 40

[SK03] Steven M. Seitz and Jiwon Kim. Multiperpective imaging. *IEEE Computer Graphics and Applications*, 23(6):16–19, 2003. 8

[SS00] Heung-Yeung Shum and Richard Szeliski. Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision*, 36(2):101–130, 2000. 4

[ST04] Peter Sand and Seth Teller. Video matching. *ACM Trans. Graph.*, 23(3):592–599, 2004. 73

[Uni03] University of Hannover. Voodoo camera tracker. http://www.digilab.uni-hannover.de/docs/manual.html, 2003. 33, 62

[Van06] Virtually Vancouver. Virtual visit of vancouver. http://www.virtuallyvancouver.com/index2.html, 2004-2006. 4

[VC01] Scott Vallance and Paul Calder. Multi-perpective images for visualisation. In *Proceedings of the Pan-Sydney area workshop on Visual Information Processing 2001*, pages 69–76. Australian Computer Society, Inc., 2001. 7

[VWL04] Vaibhav Vaish, Bennett Wilburn, and Marc Levoy. Using plane + parallax for calibrating dense camera arrays. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (to appear)*. IEEE Computer Society, 2004. 71

[WFH$^+$97] Daniel N. Wood, Adam Finkelstein, John F. Hughes, Craig E. Thayer, and David H. Salesin. Multiperspective panoramas for cel animation. In *Proceedings of the 24th annual conference on Computer Graphics and Interactive Techniques*, pages 243–250. ACM Press/Addison-Wesley Publishing Co., 1997. 6

[XDCW02] Feng Xiao, Jeffrey M. DiCarlo, Peter B. Catrysse, and Brian A. Wandell. High dynamic range imaging of natural scenes. In *The Tenth Color Imaging Conference: Color Science and Engineering Systems, Technologies, Applications*, pages 337–342, 2002. 46, 47

[XS04] J. Xiao and M. Shah. Tri-view morphing. *Computer Vision and Image Understanding*, 2004. 40

[YM04] Jingyi Yu and Leonard McMillan. General linear cameras. In *Proceedings of the eighth European conference on Computer Vision (to appear)*. Springer-Verlag New York, Inc., 2004. 7, 18, 19, 27

[ZFPW03a] A. Zomet, D. Feldman, S. Peleg, and D. Weinshall. Mosaicing new views: The crossed-slits projection, 2003. 5, 7, 14

[ZFPW03b] Assaf Zomet, Doron Feldman, Shmuel Peleg, and Daphne Weinshall. Mosaicing new views: The crossed-slits projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(6):741–754, 2003. 5, 54

[Zhe03] Jiang Yu Zheng. Digital route panoramas. *IEEE MultiMedia*, 10(03):57–67, 2003. 8, 52

[Zhe04] Jiang Yu Zheng. Stabilizing route panoramas. In *ICPR (4)*, pages 348–351, 2004. 8