Detailed Real-Time Urban 3D Reconstruction From Video

M. Pollefeys[+], D. Nistér[*], J.-M. Frahm[+], A. Akbarzadeh[*], P. Mordohai[+], B. Clipp[+], C. Engels[*], D. Gallup[+], S.-J. Kim[+], P. Merrell[+], C. Salmi[+], S. Sinha[+], B. Talton[+], L. Wang[*], Q. Yang[*], H. Stewénius[*], R. Yang[*], G. Welch[+], H. Towles[+]

[+]Department of Computer Science
University of North Carolina
Chapel Hill, USA

[*] Center for Visualization and Virtual Environments
University of Kentucky
Lexington, USA

Abstract

The paper presents a system for automatic, geo-registered, real-time 3D reconstruction from video of urban scenes. The system collects video streams, as well as GPS and inertia measurements in order to place the reconstructed models in geo-registered coordinates. It is designed using current state of the art real-time modules for all processing steps. It employs commodity graphics hardware and standard CPU's to achieve real-time performance. We present the main considerations in designing the system and the steps of the processing pipeline. Our system extends existing algorithms to meet the robustness and variability necessary to operate out of the lab. To account for the large dynamic range of outdoor videos the processing pipeline estimates global camera gain changes in the feature tracking stage and efficiently compensates for these in stereo estimation without impacting the real-time performance. The required accuracy for many applications is achieved with a two-step stereo reconstruction process exploiting the redundancy across frames. We show results on real video sequences comprising hundreds of thousands of frames.

1 Introduction

Reconstruction of buildings and landscapes in 3D from images and videos has long been a topic of research in computer vision and photogrammetry. Recently, applications such as Google Earth and Microsoft Virtual Earth have been very successful in delivering effective visualizations of large scale models based on aerial and satellite imagery to a broad audience. This has created a demand for ground-based models as the next logical step to offer 3D visualizations of cities. Visualizations using such data are possible in the form of

Figure 1: View from above and details of reconstructed models from a 170,000 frame video

panoramic mosaics [63, 48] or simple geometric models [13] which require less data to be constructed but also limit the user's ability to freely navigate the environment. For totally unconstrained navigation in the virtual environment, accurate and detailed 3D models are required. Google and Microsoft have begun acquiring ground-based videos of primarily cities, but have not yet been able to deliver to the public high-quality ground-based models. The massive amounts of data pose significant challenges for the collection, processing and visualization systems.

In this paper, we introduce a large-scale, 3D reconstruction system that operates at approximately 30Hz while still delivering detailed 3D models in the form of textured polygonal meshes. The system incorporates data from a Global Positioning System (GPS) and an Inertial Navigation System (INS), if available, or relies on traditional structure from motion algorithms otherwise. Our approach uses computationally efficient components and computation strategies to achieve real-time processing for the enormous amounts of video data required to reconstruct entire cities. The selected methods are particularly efficient on typical urban scenes, while preserving the ability to perform reconstructions of general 3D shapes. Additionally, our methods meet the high demand for robustness in order to achieve automatic large scale reconstructions. For image based camera pose estimation we developed a novel 2D feature tracking technique that efficiently estimates the gain of the camera while tracking the 2D feature points [33]. Automatic gain adaptation is essential because the dynamic range in natural scenes is far larger then the range of the camera. Accordingly, camera gain and exposure time must vary during recording which can significantly degrade the performance of standard tracking techniques. The camera pose estimation and fusion with GPS and inertial measurements, if available, is performed by a Kalman filter. Alternatively, a vision based approach similar to [43] is used to estimate the camera motion. A preliminary version of our system in which many of the components were in early stages of development appeared in [1]. The gain-estimating feature tracker was not part of the pipeline, while the stereo, depth map fusion and model generation modules were at very preliminary stages. Moreover, the models shown in that paper are not geo-registered since GPS/INS were not fused with the results of vision-based pose estimation.

For dense 3D reconstruction, we developed a novel two-stage strategy that allows us to achieve high processing rates. By decoupling the problem into the reconstruction of depth maps from sets of images followed by the fusion of these depth maps, we are able to use simple fast algorithms that can be implemented on the Graphics Processing Unit (GPU). The depth maps are reconstructed using an extended plane sweeping stereo technique operating on frames of a single video-camera [22]. Plane sweeping stereo was selected because it is ideal for efficient implementation due to its simplicity and parallelizability [12, 67]. The primary operation of the algorithm is to render images onto planes and this operation can be computed quickly on the GPU. Additionally, we incorporate priors obtained from the reconstructed sparse points into our depth estimation. This aids in areas with little texture and produces a smoother result. We can also significantly reduce computation time by sweeping planes only in those regions with high prior probability according to the obtained sparse data. Details and extensions of our work on stereo not included here can be found in [22].

The fast multi-view stereo algorithm we utilize does not perform a costly global optimization and the resulting error can therefore be comparatively high. The fusion step that follows combines multiple depth maps into a consistent depth map increasing accuracy and decreasing redundancy. It delivers a compact and geometrically consistent representation of the 3D scene, which is especially important for long video sequences. Figure 1 shows an aerial view of a model reconstructed from approximately 170,000 frames, as well as some close-up views of parts of the model. Details an extensions of our stereo fusion algorithm are given in [37].

The paper is organized as follows: Section 2 is an overview of our system; Section 3 briefly presents related work; Section 4 describes our KLT tracker that also estimates the gain of the camera; Section 5 discusses pose estimation with and without GPS/INS data; Section 6 presents our approach to multiple-view stereo; Section 7 describes two algorithms for depth map fusion; Section 8 highlights design choices that enable our system to run in real time; Section 9 contains results and Section 10 concludes the paper.

2 System Overview

This section is a brief overview of the system. The core algorithms operate on the frames of a single video-camera as it moves in space. The reconstructions are based on frames captured at different time instances by the same camera under the assumption that the scenes remain static. Models from multiple cameras can be combined at the end, but the only multi-camera module is pose estimation using a Kalman filter described in Sec. 5.3.

The majority of the data shown in the paper have been collected by a video acquisition system, which consists of eight cameras mounted on a vehicle, with a quadruple of cameras looking to each side. Each camera has a field of view of approximately $40^\circ \times 30^\circ$, and within a quadruple the cameras are arranged with minimal overlap in field of view. Three cameras are mounted in a plane to create a horizontal field of view of approximately 120 degrees. The fourth camera is tilted upward to create an overall vertical field of view of

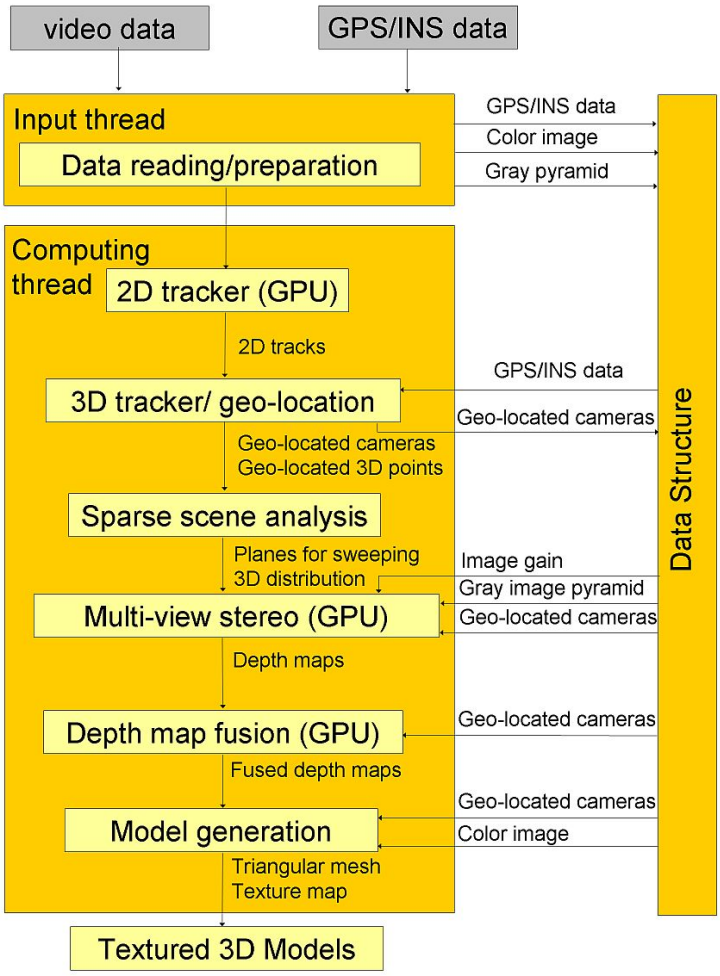


Figure 2: Processing modules and data flow of the 3D reconstruction system.

approximately 60 degrees with the side-looking camera. The cameras have a resolution of 1024×768 pixels and a frame rate of $30Hz$. Additionally, the acquisition system employs an INS and a GPS, which are synchronized with the cameras, to enable geo-registration of the cameras and the reconstructions. As the vehicle is driven through urban environments, the captured video is stored on disk drives in the vehicle. Timestamps are recorded with each frame to facilitate the retrieval of the corresponding GPS/INS data. Approximately 1TB of raw video data is recorded per hour. After a capture session, the drives are moved from the vehicle to a computer cluster for processing. The system is set up so that each video stream is processed by one PC node to achieve real-time performance.

The processing pipeline for 3D reconstruction from video consists of several computation stages (see Figure 2):

1. **Reading and preparation** of the video data as well as the GPS/INS data. This input step runs asynchronously in a thread to improve performance. It efficiently

converts the incoming Bayer-pattern video into grayscale pyramid data for later use in tracking and stereo.

2. **2D tracking** of salient image features. This step provides 2D tracks that potentially belong to a 3D point. This module can run on the GPU with more than $30Hz$ for images of size 1024×768 tracking 1000 salient features [57]. We also propose an extension of the KLT tracker that estimates the gain of the camera [33].

3. **3D tracking/geo-location** estimates the camera pose. A Kalman filter uses the 2D correspondences and GPS/INS data to estimate geo-located camera projection matrices. Alternatively, in the absence of INS/GPS data, we perform pose estimation using structure from motion techniques [42].

4. **Sparse scene analysis** examines the 3D feature points to determine three orthogonal sweeping directions (one for the ground and two for the facades). In addition, it examines the distribution of feature points in 3D to determine the most likely planes of the scene. This information enhances the effectiveness and efficiency of our multi-way plane sweeping stereo module.

5. **Stereo depth estimation** computes depth maps from the (geo-located) camera poses and the images using a fast GPU implementation of an advanced multi-view plane sweeping stereo algorithm. Our algorithm has provisions to deal with non-fronto-parallel surfaces, occlusions and gain changes.

6. **Depth map fusion** combines multiple depth maps to reject erroneous depth estimates and remove redundancy from the data, resulting in a more accurate and smaller in size set of depth maps.

7. **Model generation** creates a triangular mesh for each fused depth map and determines the texture mapping. It also removes duplicate representations of the same surface and fills some of the holes.

3 Related Work

In the last few years there have been significant efforts in large scale reconstruction, typically targeting urban environments or archeological sites. Here, we discuss research on urban reconstruction from ground-based imagery as it is closely related to our work. There has also been a considerable amount of work involving 3D reconstruction from aerial images [18, 26, 69].

A natural choice to satisfy the requirement of modeling the geometry and appearance is the combined use of active range scanners and digital cameras. Früh and Zakhor [20] developed a system that is similar to ours since it is also mounted on a vehicle and captures large amounts of data in continuous mode, in contrast to the previous approaches of Stamos and Allen [60] and El-Hakim et al. [15], that scan the scene from a set of pre-specified

viewpoints. The system of [20] consists of two laser scanners, one for map construction and registration and one for geometry reconstruction, and a digital camera, for texture acquisition. A system with similar configuration, but smaller size, that also operates in continuous mode was presented by Biber et al. [6].

Researchers in photogrammetry and computer vision address the problem of 3D reconstruction from images only. Systems using passive sensors are economic in size, weight and cost. The challenges for these systems are primarily the well-documented inaccuracies in 3D reconstruction from 2D measurements. We also use passive sensors only, employing the work on structure from motion and shape reconstruction within the computer vision community in the last two decades [17, 27].

There has been a large amount of work on estimating camera motion from image data and it is outside of the scope of this paper to review this in detail. Although it has been shown that it is possible to automatically calibrate cameras from videos [16, 47], in this paper we assume the cameras have been pre-calibrated. There are mainly two approaches for estimating the camera motion. The first approach leverages the work in multiple view geometry and typically alternates between robustly estimating camera poses and 3D point locations directly [5, 19, 42, 43]. Often bundle adjustment [2] is used in the process to refine the estimate. The other approach uses an extended Kalman filter to estimate both camera motion and 3D point locations jointly as the state of the filter [3, 58].

Complete systems for large scale urban modeling from images only include the 4D Atlanta project carried out by Schindler et al. [54, 53], which also examines the evolution of the model through time. Recently, Cornelis et al. [13] presented a system for near real-time city modeling that employs a simple model for the geometry of the world. Specifically, they assume that the facades are ruled surfaces consisting of lines all parallel to the gravity vector. The ground plane is defined by the ruled surface on which the car drives from known camera height.

In the remainder of this section we present research related to the main components of our system. The first task of the processing pipeline is automatic tracking of salient features in video, for which the KLT tracker [36] is commonly used. Sinha et al. [57] presented a real-time implementation of the KLT tracker on the GPU. The KLT tracker assumes that features do not change appearance throughout the video. The limited dynamic range of cameras, however, is often too small to accommodate the dynamic range of natural scenes. Accordingly, the gain of the camera needs to change which in turn causes the appearance of the features to change. Baker et al. [4] and Jin et al. [31] proposed different approaches to account for the change in appearance by the estimation of patchwise correction parameters. The gain ratio is computed for each feature independently even though it is a global parameter for each image. With a linear or known camera response function, we can solve for the gain as a global parameter instead of solving separately for each tracked patch. This is both more efficient computationally and more stable with respect to noise.

We do not review the prolific literature on binocular or multiple-view stereo here, since these papers, with very few exceptions, address the cases of limited still images taken from positions that are further apart than the typical distance between successive frames in a video stream. Excellent surveys can be found in [51, 55]. For dense geometry estimation,

our system uses an extension of the efficient plane sweeping technique, which was introduced by Collins [12] as a way to perform matching across multiple images simultaneously without the need for rectification. The approach was originally targeted at the reconstruction of sparse features. Yang and Pollefeys [67] modified the plane-sweeping stereo algorithm to achieve real-time dense depth estimation on the GPU.

For dense estimation in urban scenes, Werner and Zisserman [65] proposed an approach to reconstruct buildings. It uses sparse point and line correspondences to discover the ground and facade planes. When there is insufficient information to locate a plane, they sweep a hypothesized plane through space to determine the position which best matches the images. It should be noted here that this method does not allow for any details that deviate from the selected plane. Several stereo algorithms explicitly handle slanted surfaces. Burt et al. [11] advocate pre-warping the images to a reference plane, such as the ground, before performing binocular stereo. In this way, they achieve greater accuracy as well faster computation due to the reduced disparity range. Birchfield and Tomasi [7] cast the problem of stereo as image segmentation, which also accounts for affine transformations. Processing iterates between segmentation and affine parameter estimation for each segment using graph cuts [9]. Ogale and Aloimonos [44] proposed an algorithm based on dynamic programming to obtain correspondences between segments of scanlines rather than pixels to account for occlusion. Zabulis and Daniilidis [68] explicitly addressed non-fronto-parallel surfaces by performing correlation on 3D planes instead of the image plane. Thus, correlation kernels can be aligned with the scene surfaces, but the dimensionality of the search space is increased from 1D (disparity) to 3D (disparity and two rotation angles). In this paper, we present methods for aligning the correlation windows with the surfaces without exhaustive search for urban scenes.

Large scale systems typically generate partial reconstructions which are then merged. Conflicts and errors in the partial reconstructions are identified and resolved during the merging process. Surface fusion has received considerable attention in the literature mostly for data produced by range finders, where noise levels and the fraction of outliers are typically lower than those of passive stereo. It should be noted that many surface fusion algorithms are limited to single objects and are not applicable to large scale scenes due to computation and memory requirements.

Turk and Levoy [64] proposed an algorithm for registering and merging two triangular meshes. They remove the overlapping parts of the meshes, connect their boundaries and then update the positions of the vertices. Soucy and Laurendeau [59] introduced a similar algorithm which first updates the positions of the vertices and then connects them to form the triangular mesh.A different approach was presented by Curless and Levoy [14] who employ a volumetric representation of the space and compute a cumulative weighted distance function from the depth estimates. This signed distance function is an implicit representation of the surface. A volumetric approach that explicitly takes into account boundaries and holes was published by Hilton et al. [29]. Wheeler et al. [66] modified the method of [14] to only consider potential surfaces in voxels that are supported by some consensus, instead of just one range image, thus increasing robustness to outliers. An online algorithm using a sensor based on structured light was later introduced by Rusinkiewicz

et al. [49]. It can merge partial reconstructions in the form of point clouds in real-time by quantizing the space in voxels and averaging the points and their normals that fall in the same voxel. A slower, more accurate algorithm was also described.

One of the first approaches for passive data was that of Fua [21] who adopts a particle based representation. The positions and orientations of the particles are initialized from the depth estimates and modulated according to an image-based cost function while forces between particles enforce smoothness. Koch et al. [35] begin by establishing binocular pixel correspondences and proceed by linking more cameras with these correspondences. When a correspondence is consistent with a new camera, the camera is added to the chain. The position of the point is updated using the wider baseline, reducing the sensitivity to noise. Narayanan et al. [40] compute depth maps using multi-baseline stereo and merge them to produce viewpoint-based visible surface models. Holes due to occlusion are filled in from nearby depth maps.

Koch et al. [34] presented a volumetric approach for fusion as part of an uncalibrated 3D modeling system. Given depth maps for all images, the depth estimates for all pixels are projected in the voxelized 3D space. Each depth estimate votes for a voxel in a probabilistic way and the final surfaces are extracted by thresholding the 3D probability volume. Sato et al. [50] also proposed a volumetric method based on voting. Each depth estimate votes not only for the most likely surface but also for the presence of free space between the camera and the surface. Morency *et al.* [38] operate on a linked voxel space to produce triangulations at high rates. Information is fused in each voxel while connectivity information is maintained and updated in order to produce the final meshes. Goesele et al. [24] presented a two-stage algorithm which merges depth maps produced by a simple algorithm. Normalized cross-correlation is computed for each depth estimate between the reference view and several target views. The depth estimates are rejected if the normalized cross-correlation is not large enough for at least two target views. The remaining depth estimates are used for surface reconstruction using [14].

The final step is the conversion of the viewpoint-based representation in the form of depth maps to a 3D model. The typical goals of algorithms that produce mesh from depth maps are fidelity to the input data with as small a number of triangles as possible and suppression of artifacts that result from connecting vertices of different surfaces. One approach is to construct a triangulated irregular network in which vertices are placed more densely in parts of the depth map with details and variations as in the work of Garland and Heckbert [23]. Morris and Kanade [39] also consider properties of the images to refine an initially coarse triangulation to increase its consistency with the imaged surfaces. Our approach is similar to the work of Pajarola [45] which uses a quadtree defined on the image grid to achieve high vertex density where needed very efficiently.

The following sections introduce the components of our real-time 3D urban reconstruction system in more detail.

4 Gain Estimating KLT

First, our reconstruction system detects and tracks the salient features in the video data. Typically the KLT tracker [36, 56] is used to compute the displacements of features (dx, dy) between consecutive video frames. Sinha et al. [57] presented a real-time implementation of the KLT tracker that exploits the parallelism provided by modern programmable graphics hardware. The implementation presented here extends that work by assuming that camera gain is not constant and attempting to estimate it from frame to frame. The original KLT tracker relies on constant appearance of the features in consecutive images (image brightness constancy constraint), which is usually achieved by keeping the exposure time and gain of the camera constant. For large scale urban reconstruction this cannot be enforced due to the large illumination changes that are typically encountered. Therefore, as the exposure time and gain are allowed to adapt, the brightness constancy constraint is no longer satisfied and the performance of the KLT tracker degrades significantly.

We propose a variant of the KLT tracker that estimates the gain change of the camera as a global parameter for a linear or known camera response function. We model the gain ratio $\beta_t^{t+dt} = 1 + d\beta_t^{t+dt}$ of the images at time t and at time $t + dt$ together with the appearance:

$$I(x + dx, y + dy, t + dt) = (1 + d\beta_t^{t+dt})I(x, y, t). \tag{1}$$

For the standard KLT tracker $d\beta_t^{t+dt}$ is considered to be zero. Assuming equal displacement for all pixels of a patch F_i around each feature i, the displacement for each feature (dx_i, dy_i) and the gain ratio β_t^{t+dt} are estimated by minimizing the following error function:

$$E(dx_i, dy_i, d\beta_t^{t+dt}) = \sum_{x,y \in F_i} (I(x + dx_i, y + dy_i, t + dt) - (1 + d\beta_t^{t+dt})I(x, y, t))^2, \tag{2}$$

where n is the number of features. The Taylor series expansion of Equation (2) is:

$$E(dx_i, dy_i, d\beta_t^{t+dt}) = \sum_{x,y \in F_i} (I_x dx_i + I_y dy_i + I_t - d\beta_t^{t+dt} I)^2, \tag{3}$$

with $I = I(x, y, t)$, $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$, $I_t = \frac{\partial I}{\partial t}$. Since Equation (3) is minimized when all partial derivatives vanish for each feature i:

$$\underbrace{\begin{bmatrix} \mathbf{U_i} & \mathbf{w_i} \\ \mathbf{w_i}^T & \lambda_i \end{bmatrix}}_{\mathbf{A_i}} \mathbf{x} = \begin{bmatrix} \mathbf{b_i} \\ c_i \end{bmatrix} \text{ with } \mathbf{U_i} = \begin{bmatrix} \sum_{F_i} {I_x}^2 & \sum_{F_i} I_x I_y \\ \sum_{F_i} I_x I_y & \sum_{F_i} {I_y}^2 \end{bmatrix}, \mathbf{w_i} = \begin{bmatrix} -\sum_{F_i} I_x I \\ -\sum_{F_i} I_y I \end{bmatrix},$$

$$\lambda_i = \sum_{F_i} I^2, \mathbf{b_i} = \begin{bmatrix} -\sum_{F_i} I_x I_t & -\sum_{F_i} I_y I_t \end{bmatrix}^T, c_i = \sum_{F_i} I I_t, \mathbf{x} = \left[dx_i, dy_i, d\beta_t^{t+dt}\right]^T \tag{4}$$

needs to be solved. The unknown displacements (dx_i, dy_i) and the global gain ratio β_t^{t+dt} can be estimated by minimizing the error $E(dx_1, dy_1, ..., dx_n, dy_n, \beta_t^{t+dt})$ for all features

simultaneously. Combined with Equation (3) this leads to:

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} \mathbf{U} & \mathbf{w} \\ \mathbf{w}^T & \lambda \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{b} \\ c \end{bmatrix} \tag{5}$$

with

$$\mathbf{U} = \begin{bmatrix} \mathbf{U_1} & 0 & \dots & 0 \\ 0 & \mathbf{U_2} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \dots & \mathbf{U_n} \end{bmatrix}, \quad \mathbf{w} = \left[\mathbf{w_1}, \dots \mathbf{w_n}\right]^T, \quad \lambda = \sum_{i=1}^{n} \lambda_i \text{ and } c = \sum_{i=1}^{n} c_i$$

$$\mathbf{b} = \begin{bmatrix} b_1 & \dots & b_n \end{bmatrix}^T, \quad \mathbf{x} = \begin{bmatrix} dx_1 & dy_1 & dx_2 & dy_2 & \dots & dx_n & dy_n & d\beta_t^{t+dt} \end{bmatrix}^T.$$

Since the matrix $\mathbf{A}$ is sparse, we can take advantage of the structure to find a computationally efficient solution. Then the global gain ratio $\beta_t^{t+dt} = 1 + d\beta_t^{t+dt}$ is:

$$(-\mathbf{w}^T\mathbf{U}^{-1}\mathbf{w} + \lambda)d\beta_t^{t+dt} = -\mathbf{w}^T\mathbf{U}^{-1}\mathbf{b} + c \tag{6}$$

where the inverse of $\mathbf{U}$ can be computed by inverting each 2×2 diagonal block in $\mathbf{U}$ separately (which is equal to the amount of work needed for the traditional KLT tracker). Once $d\beta_t^{t+dt}$ is found, solving for displacements becomes trivial. For each patch i, dx_i and dy_i are calculated by back-substituting $d\beta_t^{t+dt}$. Hence the proposed estimation adds one additional equation (6) to solve to the original KLT tracking equations.

4.1 Gain Estimation Results

In this section, we discuss the evaluation of our proposed tracker on real image sequences, captured by a 1024×768 resolution Point Grey Flea camera with a 40 degree field of view and a linear response function. The camera operated at 30 fps and all camera settings were held fixed except for automatic gain control to account for the large variation in illumination of the scene. The camera was mounted on a vehicle moving at roughly 10 km/h during video capture. This particular camera also featured the ability to embed the automatically selected gain factor in the image data. We used this information to verify the gain estimation of our KLT tracker. The change of the gain for the sequence of 400 images was computed with the proposed tracker and is compared to the gain reported by the camera in Figure 3. The average error is 0.3%, the standard deviation of the error is also about 0.3% and the maximal observed error is 1.88% with respect to the reported camera gain. Our performance is better than that of the implementation of Birchfield which is available at http://www.ces.clemson.edu/~stb/klt/. In this case the global gain was obtained by averaging the gain for all the tracked features. It achieves an average error of 0.4%, a standard deviation of 0.7% and maximum error of 4.7%.

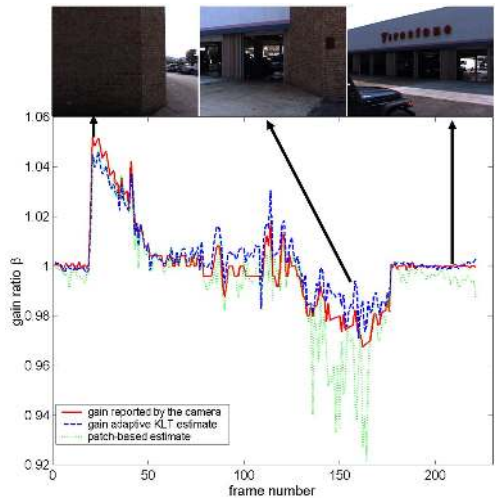


Figure 3: Estimated gain for the video sequence and example images.

5 Camera pose estimation

After tracking the salient 2D features and estimating the camera gain, the next step in our reconstruction system is to estimate the camera poses from the 2D feature tracks. Our system provides two estimation methods for the camera: one based purely on visual inputs [42, 43] and an extended Kalman filter based approach that fuses visual data with GPS/INS data, if they are available. The former method can only provide the camera poses up to scale. Moreover, as Nistér et al. [43] showed, this estimation is reliable only if the system includes a stereo rig, that is at least two cameras with common field of view. The baseline of the stereo rig provides an immediate estimate for the scale. Since this is not true for our system, vision-only pose estimation is especially challenging and susceptible to large drift. We have been able, however, to estimate the pose of the camera for relatively long sequences. The inability to estimate scale does not allow us to combine models made using different cameras since registering them is not trivial. The GPS/INS measurements provide a means of maintaining globally accurate estimates of position, orientation and scale. On the other hand, vision-based tracking allows us to make small refinements and reduce the reprojection errors. The Kalman filter fuses these two types of estimates to maximize the benefits.

5.1 Visual Odometry

Our system uses an efficient implementation of [42] to estimate the pose of a moving calibrated camera using images only. It initializes the camera tracker with the relative pose of three views, given 2D feature correspondences in them. This is followed by a preemptive RANSAC and a local iterative refinement step for robustness [41]. The correct correspondences are triangulated using the computed camera poses and the optimal method of [28]. Additional poses are computed with RANSAC and hypothesis-generation using

constraints from 2D feature to 3D world point correspondences. New world points are re-triangulated using new views as they become available. To provide robustness and isolate the system from instabilities that can occur in the structure and motion computations, the system is periodically re-initialized with a new set of three views. We stitch the new poses into the old coordinate system exploiting the constraints of one overlapping camera. The remaining degree of freedom is the scale of the old and the new coordinate system. It is estimated using corresponding triangulated points in both coordinate frames.

While this approach works well for relatively short image sequences, it is difficult to reliably estimate the camera motion over long distances using video only, especially when using monocular structure from motion [43]. The scale of the reconstruction is particularly hard to estimate consistently over longer distances. Even though our system has multiple cameras, the absence of significant overlap prohibits the use of stereo motion estimation algorithms. It is should theoretically be possible to estimate the absolute scale using a generalized camera model for the multi-camera setup [61], our configuration of two clusters of cameras, however, is degenerate in this respect.

5.2 Geo-Located Pose Estimation

The purely image-based visual odometry approach delivers the camera poses only up to scale in an arbitrary coordinate system defined by the first camera. Our system, however, aims at a geo-registered 3D model. Therefore, whenever GPS/INS data are available we use an extended Kalman filter to fuse the vision based measurements and the GPS/INS data to obtain geo-located camera poses. Our collection system uses eight minimally-overlapping video cameras and a GPS/INS to acquire these data. The knowledge of the lever-arm between the GPS/INS and the cameras, provides the necessary constraints to estimate geo-registered camera poses. Using this information, reconstructed models from different cameras are automatically generated in the same coordinate system and can be viewed together.

The use of a Kalman filter in structure and motion estimation is a well known technique in vision [3, 58]. For a detailed introduction to Kalman filtering [25] is suggested. We use an extended Kalman filter to estimate the pose of the data collection platform and subsequently the cameras. The state of the extended Kalman filter consists of the data collection platform's translation, rotation, velocity and rotation rate as well as estimates of the 3D locations of the tracked salient features in all cameras, in an orthogonal, earth-centered, earth-fixed coordinate system Universal Transverse Mercator (UTM).

The process model of the extended Kalman filter is a smooth motion model, assuming constant velocity in translation and rotation, to model the change in the data collection platform's pose over time. We assume that we observe a static scene and so the 3D features are stationary with respect to the earth. For the GPS/INS measurement model, the state's position and orientation map directly to the position and orientation of the GPS/INS. The measurement model of a 3D feature estimate is the projection of the 3D feature X into

the camera that is tracking it.

$$x = P_i X = T_{pi} \begin{bmatrix} R_p^T & -R_p^T C_p \\ 0 & 1 \end{bmatrix} X, \tag{7}$$

The camera's projection matrix P_i is generated by left multiplying the matrix P_p of the camera platform by the lever-arm transformation matrix for camera i.

A typical Kalman filter implementation of camera tracking would have a covariance matrix with the camera motion parameters and all of the 3D features and their relationships to each other. In a multi-camera system, with each camera tracking possibly several hundred features, the covariance matrix is too large to work with efficiently. Assuming the tracked salient features do not move with respect to the earth, the features are statistically independent. The feature-to-feature covariances are therefore zero and the influence of their measurements can be incorporated into the filter's state estimate sequentially rather than as a block [10]. In our current filter implementation, 3D features only exist in memory so long as their corresponding 2D features are being tracked. This makes loop completion impossible and would result in significant drift over time without the constraints imposed by the GPS/INS system.

5.3 Pose Estimation Results

The GPS/INS system we use, the Applanix POS LV 420, provides highly accurate estimates of the poses, but they can still be improved upon. In the Kalman filter, the difference between the reprojected 3D features and their measurements are used to correct the pose of the data collection system. This correction provides a 10% improvement in reprojection error compared to when the 3D features are estimated but do not correct the data collection system pose. The median improvement is 0.05 pixels which appears small but is approximately one half of the standard deviation of the GPS/INS measurement uncertainty. Under normal operating conditions, the feature estimates' contribution to reconstruction accuracy is relatively minor. However, incorporating 3D feature estimates into the pose estimation makes the reconstruction system robust to failures in the GPS/INS system. Robustness is critical when reconstructing large urban environments as is demonstrated in the following example.

Figure 4 shows the delivered GPS/INS altitude measurements and filter estimates while the collection system vehicle approaches a stoplight, stops and then proceeds. The estimated altitude varies within ± 1 standard deviation of the GPS/INS measurements before the four second mark in the plot. This shows that the GPS/INS measurements tend to drive the large scale system motion while the feature tracks and smooth motion model compensate for errors in the GPS/INS measurements. From four seconds to approximately eleven seconds the GPS/INS incorrectly measures a $10cm$ upward motion while the vehicle is stationary. By detecting that the salient features do not move in the images, the filter is able to compensate for the errant GPS/INS measurements. Robust reconstruction is made possible by fusing multiple complementary measurement sources to overcome a failure in a single sensor. While its main functionality in our system is to protect against intermittent

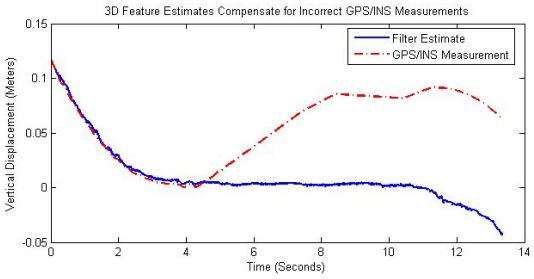


Figure 4: Above: Errant vertical motion measured by the INS/GPS and the Kalman filter estimated vertical motion.

GPS/INS failures, the Kalman filter fusing vision/INS/GPS becomes more valuable with a less accurate INS or with GPS only.

In addition to accuracy, speed is paramount to reconstructing large urban scenes. The Kalman filter based 3D tracker was tested over a sequence of 1000 frames captured with four synchronized cameras along with GPS/INS measurements. The filter ran at $31.1Hz$ with reprojection errors of the tracked salient features of 0.4 pixels. To achieve this speed the total number of tracked 3D salient features was kept around 120. Even with only 120 features, the filter is able to compensate for failures in the GPS/INS system as demonstrated above.

6 Plane Sweeping Stereo with Multiple Sweeping Directions

In the previous sections, we discussed 2D feature tracking and camera pose estimation. The next step of our system is to compute the depth of the scene using stereo [22]. As mentioned in Sec. 2, the cameras in our system have minimal overlap and their purpose is to provide a wider field of view. Thus, stereo is performed on consecutive frames of a single camera as it moves. To achieve real-time performance, we select plane sweeping stereo [12] which naturally maps to the graphics processor of commodity graphics cards [67]. Traditional plane sweeping stereo is biased towards fronto-parallel surfaces. Since in urban scenes the cameras generally observe some of the facades and the ground at oblique viewing angles, we extend plane sweeping stereo to use multiple plane orientations aligned with the facades and the ground. The expected orientations are automatically extracted from the positions of the salient 3D features, which are computed during the camera pose estimation. Additionally, we incorporate priors from these sparse point correspondences into our depth estimation. We can significantly reduce computation time by sweeping planes only in regions with high prior probability.

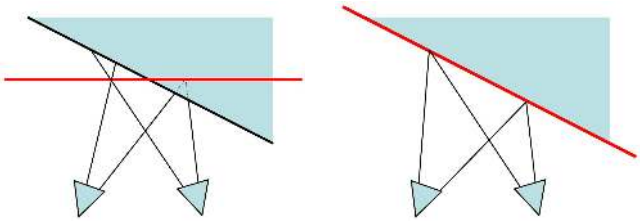


Figure 5: Implications of cost aggregation over a window. Left: Slanted surfaces with fronto-parallel plane-sweeping. Not all points over the window are in correspondence. Right: Surface-aligned sweeping plane to handle slanted surfaces correctly.

6.1 Sparse Scene Analysis: Identifying Sweeping Directions

In accordance to many other researchers, we use the absolute intensity difference as the dissimilarity or cost measure. Since the measurement at a single pixel is in general very sensitive to noise, several measurements from neighboring pixels are combined by summing the absolute intensity differences in a window centered at the pixel under consideration. This reduces sensitivity to noise but introduces an additional requirement for the stereo system: in order to detect the best correspondence for the current pixel, all pixels in the window need to be in correspondence. This is not always the case in many stereo algorithms that use windows of constant disparity since these are optimal only for fronto-parallel surfaces, which are parallel to the image plane. In plane sweeping stereo, the alignment between the actual and the hypothesized surface can be accomplished by modifying the direction of the sweep and thus the normal of the planes. Figure 5 shows the problems caused by sweeping fronto-parallel planes when the scene contains non-fronto-parallel surfaces and our proposed solution. In the remainder of this section, we discuss methods of detecting the appropriate sweeping directions.

Instead of exhaustively sampling the set of all surface orientations, we attempt to identify a much smaller set of likely surface normals by analyzing the sparse data that have been computed before stereo. For example, the motion of vehicle-mounted and even hand-held cameras is generally constrained to be parallel to the ground plane. Also, a scene's planar structure can be determined by sparse features such as lines and points. This is especially true in urban environments where, by examining lines in a single image, vanishing points can be recovered which in turn can be combined to give estimates of plane normals. Additionally, these 3D point or line features are computed as a by-product of the camera pose estimation and are often not utilized in other stereo approaches. Many techniques have been explored to recover planar surfaces from point and line correspondences and vanishing points [65, 8, 52, 30].

We present an effective technique for recovering planar structure in urban environments using 3D point features obtained from structure from motion. We first find the vertical direction or gravity vector, which is either given by an INS system or can be computed from vanishing points. Since most facades are vertical, the vanishing point corresponding to the gravity vector is quite prominent in urban scenes. By assuming the ground plane has

zero slope in the direction perpendicular to the computed camera motion, we can obtain a good estimate for the ground plane normal as:

$$\vec{G} = \frac{(\vec{V} \times \vec{M}) \times \vec{M}}{\left\| (\vec{V} \times \vec{M}) \times \vec{M} \right\|} \tag{8}$$

where $\vec{V}$ is the gravity vector, and $\vec{M}$ is the camera motion direction. Obtaining the ground normal is particularly important, since the ground is imaged at a large angle.

To compute the facade normals, we assume that they are perpendicular to the gravity vector, and are therefore determined by a rotation about the gravity vector. By assuming the facades are orthogonal to one another, only one rotation determines their normals. We recover this remaining rotation of the facades as follows. We first compute the orthogonal projection of each 3D point in the direction of gravity to obtain a set of 2D points. Note that 3D points on a common vertical facade project to a line. We then evenly sample the space of in-plane rotations between 0 and 90 degrees and test each rotation. For each rotation $R = \begin{bmatrix} u & v \end{bmatrix}^T$, we rotate the set of 2D points, and construct two histograms H_u and H_v. Each bin in H_u (resp. H_v) counts the number of points with a similar u (resp. v) component. The histogram pair with the lowest entropy represents the optimal plane orientation (see Figure 6(a) and (b)).

6.2 Plane Selection

Plane-sweeping stereo (Algorithm 1) tests a family of plane hypotheses and records for each pixel in a reference view the best plane using some dissimilarity measure. The algorithm works with any number of cameras, and images need not be rectified. The inputs to the algorithm are M 3D planes for the depth tests, $N+1$ images at different camera positions (we assume images have been corrected for known radial distortion), and their respective camera projection matrices P_k:

$$P_k = K_k[R_k^T \ -R_k^T C_k] \text{ with } k = 1, \ldots, N, \tag{9}$$

(a) Arbitrary rotation (b) Minimum entropy (c) Input frame (d) Best-cost labels

Figure 6: Minimum entropy optimization. The 3D points are projected in the direction of gravity. Then they are rotated into the basis formed by u and v and histograms are generated. The histograms of (a) have more entropy than those of (b). (b) corresponds to the correct surface normals. (c) image from original viewpoint. (d) Best cost selection (ground points are labeled green, and facade points are labeled red or blue).

where K_k is the camera calibration matrix, and R_k and C_k are the rotation and translation of camera P_k with respect to a selected reference camera P_{ref}. The reference camera is assumed to be the origin of the coordinate system and so its projection matrix is $P_{ref} = K_{ref} \left[\begin{array}{cc} I_{3\times 3} & 0 \end{array} \right]$.

$$\Pi_m = \left[\begin{array}{cc} n_m^T & -d_m \end{array} \right] \text{ for } m = 1, \ldots, M, \tag{10}$$

where n_m is the unit length normal of the plane and d_m is the distance of the plane to the origin which is set at the center of the reference camera. The normal $n_m^T = \left[\begin{array}{ccc} 0 & 0 & 1 \end{array} \right]$ is typically used for the sweeping planes. This assumes that the surfaces are parallel to the image plane (fronto-parallel). To account for non-fronto-parallel surfaces such as the ground and facades in urban scenes, our approach performs several plane-sweeps each with a different plane normals selected by the technique of Sec. 6.1.

Once the sweeping directions have been computed, we generate a family of planes for each. Each family is parameterized by the distance of the plane to the origin d_m. The range $[d_{near}, d_{far}]$ can be determined either by examining the points obtained from structure from motion or by applying useful heuristics. For example, in outdoor environments, it is usually not useful for the ground plane family to extend above the camera center. The spacing of the planes in the range can be uniform, as in [68]. However, it is best to place the planes to account for image sampling (pixels). Ideally, when comparing the respective image warpings induced by consecutive planes, the amount of pixel motion should be less than or equal to one [62]. This is particularly important when matching surfaces that exhibit high-frequency texture.

We ensure that the planes are properly spaced by computing the maximum disparity change between consecutive planes in a family. The planar mapping from the image plane of the reference camera P_{ref} to the image plane of the camera P_k can be described by the homography H_{Π_m, P_k} induced by the plane Π_m:

$$H_{\Pi_m, P_k} = K_k \left(R_k^T + \frac{R_k^T C_k n_m^T}{d_m} \right) K_{ref}^{-1}. \tag{11}$$

The location (x_k, y_k) in image I_k of the mapped reference pixel (x, y) is computed by:

$$\left[\begin{array}{ccc} \tilde{x} & \tilde{y} & \tilde{w} \end{array} \right]^T = H_{\Pi_m, P_k} \left[\begin{array}{ccc} x & y & 1 \end{array} \right]^T \text{ with } x_k = \tilde{x}/\tilde{w}, \quad y_k = \tilde{y}/\tilde{w}. \tag{12}$$

We define the disparity change between two planes Π_m and Π_{m+1} to be the maximum displacement in all images.

$$\Delta D(\Pi_m, \Pi_{m+1}) = \max_{k=0,\ldots,N-1} \max_{(x,y)\in I_k} \sqrt{(x_k^m - x_k^{m+1})^2 + (y_k^m - y_k^{m+1})^2}. \tag{13}$$

where (x_k^m, y_k^m) (resp. (x_k^{m+1}, y_k^{m+1})) are obtained by (12). To avoid orientation inversions we do not use planes that intersect the convex hull of the camera centers. Typically the greatest displacement occurs at the boundaries of the most distant images. Thus, to compute the disparity, we warp the boundaries of image I_k with the planar homography

H_{Π_m,P_k} into the reference view I_{ref}. We then compute the disparity change of the vertices of the common region. Since only those planes that do not intersect the convex hull of the camera center are used, the common region is guaranteed to be convex, and thus the maximum disparity change is bounded by the maximum disparity change of its vertices. The family of planes is then constructed so that the disparity change of consecutive planes differs in magnitude by no more than one.

6.3 Plane Sweeping Stereo

In order to test the plane hypothesis Π_m for a given pixel (x, y) in the reference view I_{ref}, the pixel is projected into the other images $k = 1, \ldots, N$ according to (11) and (12). If the plane is close to the surface that projects to pixel (x, y) in the reference view and assuming the surface is Lambertian, the colors of $I_k(x_k, y_k)$ and $I_{ref}(x, y)$ are similar. We use the absolute intensity difference as the dissimilarity measure and aggregate several measurements in a window $W(x, y)$ centered at the pixel (x, y). To increase robustness against occlusion, we adopt an algorithm proposed by Kang et al. [32]. For each pixel we compute the cost for each plane using the left and right subset of the cameras and select the minimum as the cost of the pixel. This scheme is very effective against occlusions, since typically the visibility of a pixel changes at most once in a sequence of images. Therefore, the pixel should be visible in either the entire left of right set of cameras.

Once the cost function (16) for all pixels has been computed, the depth map may be extracted. Algorithm 1 illustrates the steps for a single sweeping direction. The same process is repeated for each direction. Although our algorithm can be generalized to any number of sweeping directions, for simplicity we describe it in terms of three sweeping directions labeled $L = \{l_g, l_{f_1}, l_{f_2}\}$. Each direction also has an associated surface normal n^l and a cost volume $\mathcal{C}^l(x, y)$. The first step is to select the best plane $\tilde{\Pi}^l$ of every sweeping direction according to (17) and (18) at each pixel in the reference view. The plane of minimum cost over all sweeping directions is selected as the depth estimate for the pixel.

$$\tilde{\Pi}(x, y) = \underset{\tilde{\Pi}^l}{\operatorname{argmin}} \mathcal{C}^l(x, y, \tilde{\Pi}^l). \tag{19}$$

$\tilde{\Pi}(x, y)$ is also called the best-cost or winner-takes-all solution. For a given plane $\Pi_m = [{n_m}^T \; -d_m]$ the coordinates of the 3D point that corresponds to a pixel (x, y) can be computed by the intersection of Π_m and the ray through the pixel's center. For instance, the Z-coordinate of the 3D point is given by:

$$Z(x, y) = \frac{-d_m}{\begin{bmatrix} x & y & 1 \end{bmatrix} K_{ref}^{-T} n_m}. \tag{20}$$

6.4 Incorporating Plane Priors

The minimum-entropy histograms computed in Section 6.1 are used to find the location of the facades. They can also be used as a prior in a Bayesian formulation when selecting $\tilde{\Pi}^l$.

Algorithm 1 Plane Sweeping Stereo

1: A plane is swept through space in steps along a predefined direction.

2: For each plane Π_m, all images (I_k) are projected on the plane and rendered in the reference view (I_{ref}), and the matching cost is calculated for the left and right set of cameras as:

$$\mathcal{C}_L(x, y, \Pi_m) = \sum_{k<ref} |I_{ref}(x, y) - \beta^k_{ref} I_k(x_k, y_k)| \quad (14)$$

$$\mathcal{C}_R(x, y, \Pi_m) = \sum_{k>ref} |I_{ref}(x, y) - \beta^k_{ref} I_k(x_k, y_k)| \quad (15)$$

where I_k are the projected views, (x_k, y_k) are obtained by applying the homography H_{Π_m, P_k} as shown in (12) and β^k_{ref} is the gain ratio between image k and the reference image. The minimum of these two costs:

$$\mathcal{C}(x, y, \Pi_m) = \min\{\mathcal{C}_L(x, y, \Pi_m), \mathcal{C}_R(x, y, \Pi_m)\}$$

is assigned in the cost volume as the cost for the pixel (x, y) to be on Π_m as in [32]. The dimensions of the cost volume are the image width times height times the number of planes.

3: Under the local smoothness assumption, a boxcar filter is applied on the slices of the cost volume. (Each slice of the cost volume corresponds to a plane.)

$$\mathcal{C}^a(x, y, \Pi_m) = \sum_{(x_i, y_i) \in W(x,y)} \mathcal{C}(x_i, y_i, \Pi_m), \quad (16)$$

where $W(x, y)$ is a square window centered around (x, y) in the reference image and $\mathcal{C}^a(x, y, \Pi_m)$ is the aggregated cost for the pixel.

4: Depth values for each pixel are computed by selecting the plane that corresponds to the minimum aggregated cost:

$$\tilde{\mathcal{C}}(x, y) = \min_{\Pi_m}\{\mathcal{C}^a(x, y, \Pi_m)\}, \quad (17)$$

$$\tilde{\Pi}(x, y) = \operatorname*{argmin}_{\Pi_m}\{\mathcal{C}^a(x, y, \Pi_m)\} \quad (18)$$

where $\tilde{\mathcal{C}}(x, y)$ is the best cost for the pixel and $\tilde{\Pi}$ is the corresponding plane.

The posterior probability of a plane Π_m^l at pixel (x, y) is:

$$P(\Pi_m^l|\mathcal{C}^l(x,y)) = \frac{P(\mathcal{C}^l(x,y)|\Pi_m^l)P(\Pi_m^l)}{P(\mathcal{C}^l(x,y))} \tag{21}$$

where $P(\Pi_m^l)$ is the prior probability of the surface being at plane Π_m^l, and $P(\mathcal{C}^l(x,y)|\Pi_m^l)$ indicates the likelihood of the correct plane having matching cost $\mathcal{C}^l(x,y)$. $P(\mathcal{C}^l(x,y))$ is the marginal likelihood of the cost. The prior is obtained by sampling the normalized histogram of the 3D feature points at the location of the plane. For a plane Π_m^l chosen from sweeping direction l, the location in the histogram H_l is given by the plane depth d_m^l. The prior is:

$$P(\Pi_m^l) = \frac{H_l(d_m^l)}{\sum_i H_l(i)}. \tag{22}$$

The cost likelihood depends on image noise, camera pose error, the surface texture, and the alignment of the plane normal n_m^l with the surface normal. This is extremely difficult to model correctly. Instead, we choose an exponential distribution:

$$P(\mathcal{C}^l(x,y)|\Pi_m^l) = e^{\frac{-\mathcal{C}^l(x,y)}{\sigma}} \tag{23}$$

where σ is determined empirically. The exponential is self-similar, and so it does not rely on assumptions about the minimum cost, which is often difficult to determine beforehand.

Since we are only interested in the maximum likelihood solution we ignore $P(\mathcal{C}(x,y))$ and modify the plane selection equation (19) as follows:

$$\tilde{\Pi}^l(x,y) = \underset{\Pi_m^l}{\operatorname{argmax}}\, e^{\frac{-\mathcal{C}^l(x,y)}{\sigma}} P(\Pi_m^l). \tag{24}$$

Maximizing this likelihood is equivalent to minimizing its negative logarithm:

$$\tilde{\Pi}^l(x,y) = \underset{\Pi_m^l}{\operatorname{argmin}} \left\{ -\log e^{\frac{-\mathcal{C}^l(x,y)}{\sigma}} P(\Pi_m^l) \right\} = \underset{\Pi_m^l}{\operatorname{argmin}} \{\mathcal{C}^l(x,y) - \sigma \log P(\Pi_m^l)\}. \tag{25}$$

Surfaces with little or no texture exhibit low matching costs over a range of planes. The minimum of the cost may be determined more by noise than by true correspondence. The depth prior $P(\Pi_m^l)$ helps to eliminate such ambiguities and to produce a smoother surface. To incorporate it in the depth selection we modify the cost function $\tilde{\mathcal{C}}_l(x,y)$ to:

$$\tilde{\mathcal{C}}_l(x,y) = \mathcal{C}\left(x, y, \tilde{\Pi}_l(x,y)\right) - \sigma \log P(\tilde{\Pi}_l(x,y)). \tag{26}$$

This additional computation comes at little cost, but contributes significantly to the results.

We can also use the prior to significantly reduce our computation time by not testing plane hypotheses with a low prior probability. A scene typically requires hundreds of planes for each direction to adequately sample the disparity range. While our algorithm is able to compute that many plane hypotheses at several frames per second, we have found

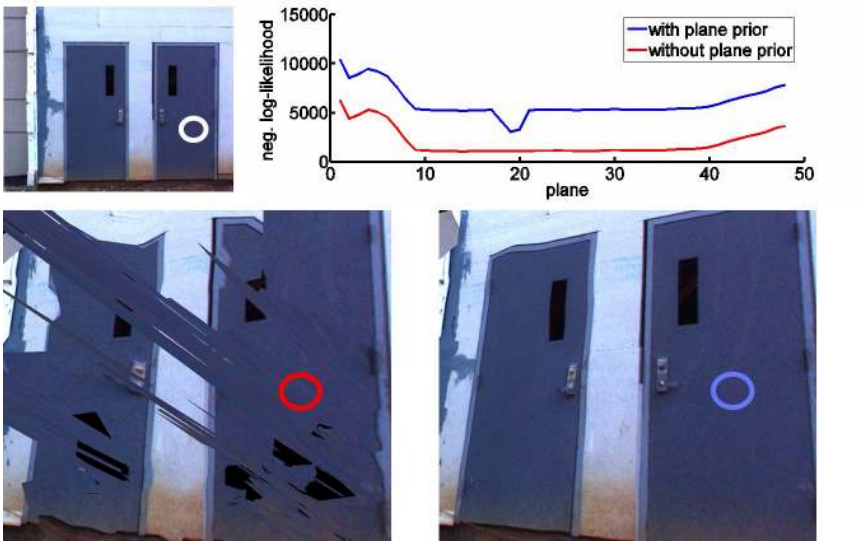


Figure 7: Illustration of the effectiveness of the prior term in stereo. Top: reference image and matching costs with and without the prior (in red and blue respectively) for the pixel in the white circle. Bottom: without the prior, matching costs are ambiguous and the reconstruction is noisy; the prior correctly disambiguates the cost function on the right.

that it is possible to obtain quality reconstructions almost an order of magnitude faster by testing only a few dozen planes. The selected planes are those with the highest prior probability. This is only effective when sweeping in multiple directions. For example, if the sweeping direction were not aligned to the ground and were instead fronto-parallel, it would require many plane hypotheses to reconstruct the ground. However, having determined the ground's surface normal and predicted its location from the prior, we can reconstruct it with only a few plane hypotheses.

6.5 Confidence of Stereo Matches

The depth with the lowest cost may not be the true depth because of noise, occlusion, lack of texture, surfaces not aligned with the chosen plane families, and many other factors. Parts of the image that have little texture are difficult to accurately reconstruct using stereo. To evaluate confidence, a measure of the accuracy of the depth of each pixel is important. We use a heuristic and assume that the cost is perturbed by a Gaussian distribution. Let $\mathcal{C}(x, y, \Pi_m)$ be the matching cost for plane Π_m at pixel (x, y). We wish to estimate the likelihood that the depth with the lowest cost, which corresponds to plane $\tilde{\Pi}$, no longer has the lowest cost after the cost is perturbed. It is proportional to:

$$e^{-(\mathcal{C}(x,y,\Pi_m)-\mathcal{C}(x,y,\tilde{\Pi}))^2/\sigma^2} \tag{27}$$

for σ from (23) that depends on the strength of the noise. Then, the confidence $c(x, y)$ is defined as the inverse of the sum of these probabilities for all possible depths:

$$c(x,y) = \left(\sum_{\Pi_m \neq \tilde{\Pi}} e^{-(\mathcal{C}(x,y,\Pi_m) - \mathcal{C}(x,y,\tilde{\Pi}))^2/\sigma^2} \right)^{-1}, \tag{28}$$

which produces a high confidence when the cost has a single sharp minimum and a low confidence when the cost has a shallow minimum or several low minima. The confidence is used in the depth map fusion process.

6.6 Plane Sweeping Stereo on the GPU

To achieve real-time performance, we use the graphics processing unit (GPU) for the plane sweeping technique. In Algorithm 1, we gain performance by executing steps 2, 3 and 4 on the GPU. We discuss the implementation of these steps in more detail below.

Plane sweeping stereo assumes that the images are corrected for radial distortion. Images typically contain radial distortion that needs to be corrected. In our processing system the only step that requires the radially undistorted images is stereo. Accordingly, we decided to compensate for it only in stereo processing. The input to the stereo module is a stream of images with known poses. Each new image is loaded on the GPU for stereo processing in a ring buffer of size equal to the number of images used in stereo. The first step that is performed on the GPU after loading is the radial undistortion of the incoming image through a vertex shader that performs a non-linear mapping with bilinear interpolation that essentially moves the pixels to the positions they would have in a pinhole camera. We now have images without radial distortion and all the following operations can be performed linearly.

In step 2 of Algorithm 1, for each view, for each plane Π_m all views I_k are projectively mapped with hardware support to the reference view I_{ref} by the homography H_{Π_m,p_k} that corresponds to the plane Π_m. In addition, since the graphics hardware is most efficient at processing 4-channel (RGB + alpha) color images, it allows us to compute four depth hypotheses at once. Once the projective mapping is completed, we use the pixel shader to calculate the gain corrected absolute differences of the projected views and the reference view according to (14) and (15), the minimum of which is written to an output texture. Since the speed is bound by the projective texture mapping operations (mostly memory access), gain correction does not add a measurable overhead. One may observe that by delaying gain correction until after image warping we must perform the per-pixel multiplication for every plane hypothesis. The image could be normalized before warping, thus saving computation time. However, this would require higher precision for image storage to account for the possible dynamic range of natural scenes and higher memory bandwidth. To avoid this overhead intensity values are stored as 8-bit integers and only converted to higher precision floating point numbers during the computation of the dissimilarity. In our implementation, we have verified that there is no observable time penalty for applying gain correction.

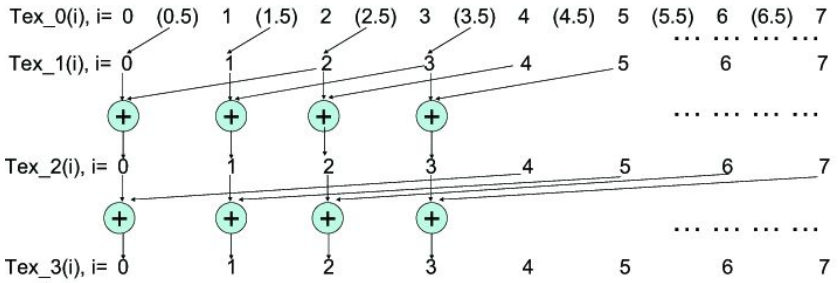


Figure 8: On the right are the indices of the textures, on the left textures that are used to save the aggregation results.

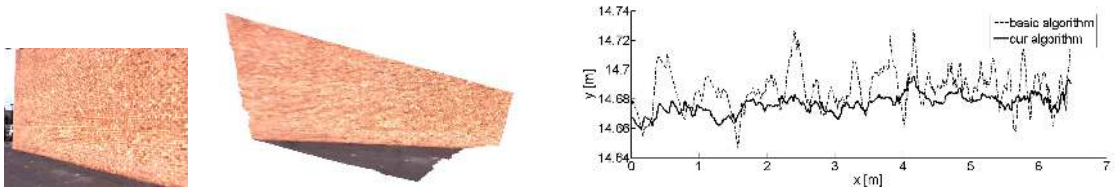


Figure 9: Comparison to fronto-parallel plane sweep. Left: the original viewpoint. Middle: The depth map triangulated into a polygonal mesh viewed from an elevated viewpoint. Right: A cross-section of the surface. Please note the scale. The standard deviation of the surface from the best-fit line was $1.31cm$ for the fronto-parallel sweep and $0.61cm$ for our algorithm.

Cost aggregation is performed in step 3. We implement a GPU boxcar filter with multiple passes. In the first pass, we take advantage of the graphics card's bilinear texture interpolation to compute the output of a 2×2 boxcar filter. For a given pixel (x, y) we access the cost image $\mathcal{C}^m$ (the slice corresponding to Π_m) at address $(x + 0.5, y + 0.5)$. The graphics card's built-in bilinear interpolation returns the average of four pixel costs, $(\mathcal{C}^m(x, y) + \mathcal{C}^m(x + 1, y) + \mathcal{C}^m(x, y + 1) + \mathcal{C}^m(x + 1, y + 1))/4$, which we then multiply by 4 before storing the result in an 8-bit texture. This avoids losing precision to discretization for low-cost regions while the potential saturation in high-cost regions has no impact on the best cost result. The result is written to an output texture $\mathcal{C}_1$ which stores the 2×2 boxcar result. In each subsequent pass, the results from previous boxcars are combined to produce a boxcar result of twice the size in each dimension. Thus in pass i for pixel (x, y), we compute the $2^i \times 2^i$ boxcar result $\mathcal{C}_i$ from the previous result as $\mathcal{C}_{i-1}(x, y) + \mathcal{C}_{i-1}(x + 2^{i-1}, y) + \mathcal{C}_{i-1}(x, y + 2^{i-1}) + \mathcal{C}_{i-1}(x + 2^{i-1}, y + 2^{i-1})$. Figure 8 summarizes the algorithm. The advantage of this approach is that the memory access is continuous. Although more texture memory accesses are required, this approach is faster than alternative approaches.

Figure 10: Left: two video frames, middle: gain corrected stereo, right: standard stereo.

6.7 Stereo Results

In this section, we first demonstrate our novel plane sweeping algorithm on several video sequences captured by a camera mounted on a vehicle moving through streets in an urban environment. Then, we evaluate our stereo algorithm on a hand-held video sequence. We compute the three sweeping directions as described in Section 6.1, and then compute depth maps from 11 grayscale 512×384 images. We processed the data on a PC with an NVIDIA GeForce 8800 GTX graphics card.

First, we compare our algorithm with the basic fronto-parallel sweep. The scene in Figure 9 is a flat brick wall which is viewed obliquely. We reconstruct the scene using 144 plane hypotheses for both algorithms. For our algorithm we selected the depths from the multiple sweeping directions using best-cost, and achieved a processing rate of at $6.33Hz$. Figure 9 compares a scan-line of the depth maps resulting from the fronto-parallel sweep algorithm and our algorithm. The surface from our algorithm is much smoother and better approximates the planar facade. For both algorithms we compute sub-pixel matches by parabolic fitting of the best cost. Even when using this parabolic fitting, the fronto-parallel sweep is unable to compute a smooth surface due to the cost aggregation window. Since only a fraction of the points in the aggregation window actually have the correct depth, the matching costs to which the polynomial is fit are less indicative of the depth of the surface.

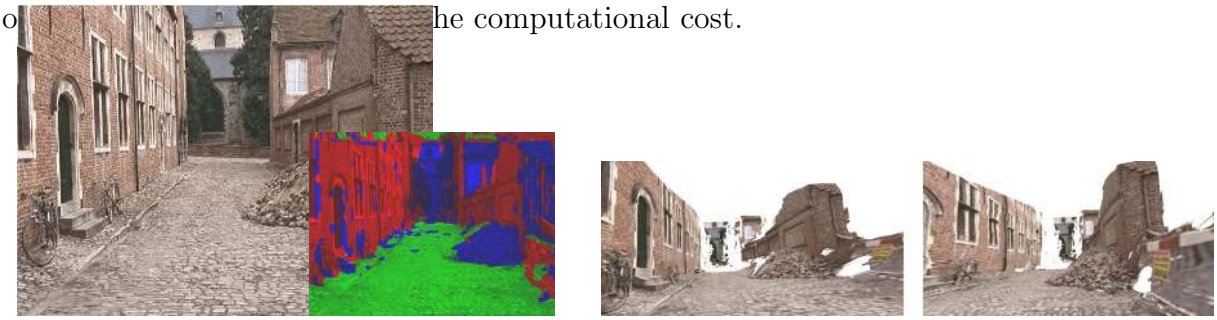
We also evaluated our stereo algorithm using the gain provided by the proposed 2D tracker from Section 4 on the same scene. For this evaluation, we selected a part of the same wall in which the cumulative gain ratio over the set of images used for stereo is $\beta^k_{ref} = 1.44$. This change is caused by capturing first in bright sunlight and afterwards in shadow. In Figure 10, we see that our system is able to correctly handle the large change in gain during stereo and produce an accurate depth map. The gain-corrected depth map has $1.9cm$ average error. Without accounting for gain, the depth map produced has severe errors: $37.4cm$ on average.

In Figure 11, we demonstrate the ability to compute accurate depth maps with only a small number of plane hypotheses. By testing only the planes with highest prior probability, we produced quality reconstructions with just 48 plane hypotheses per frame. This increases the speed of our algorithm to 33.7 Hz. Although we assume the presence of planar surfaces in the scene, our algorithm is a general stereo matcher, and we are still able to reconstruct non-planar objects such as bushes. A quantitative analysis can be found in Figure 11(c), which shows a comparison between a regular execution of the algorithm

(a) Reference image (b) Part of the reconstruction (c) Cumulative histograms

Figure 11: Evaluation of the use of priors. The dotted black curve is the cumulative histogram of the sparse points for the image of (a). The x-axis for the dotted black curve refers to the planes sorted in descending order with respect to the number of points they contain. The solid blue curve is a similar histogram for the dense reconstructed points, with the x-axis in this case corresponding to the planes sorted by the number of dense points they contain. The dashed red curve is a cumulative histogram of the dense points using the ordering of planes according to the *sparse* points they contain. See the text for more details.

using 144 planes (48 in each of the three directions) and a run that uses the 48 planes with the highest prior selected among the three families of planes. The dotted black curve in Figure 11(c) is a cumulative histogram of the sparse points in which the x-axis indexes the planes sorted in descending order with respect to the number of sparse points each of them contains. A similar histogram is shown by the solid blue curve for the dense reconstructed points after stereo, where the planes on the x-axis have been sorted according to the number of dense points they contain. Finally, the dashed red curve is a cumulative histogram of the dense points but with respect to the planes sorted according to the number of sparse points they contain. The sweep using the priors represents 77.8% of the dense points with o he computational cost.



(a) Input Frame (b) Best-cost labels (c) View of the reconstruction

Figure 12: Reconstruction of a video captured by a hand-held camera. (This figure is best viewed in color.)

Finally, we present the reconstruction of a scene captured by a *hand-held camera*. We computed the direction of the gravity vector from vertical vanishing points and computed the sweeping directions as described in Section 6.1. The reconstruction of the scene is

shown in Figure 12.

7 Depth Map Fusion

Due to the speed of the algorithm, the raw stereo depth maps contain errors and do not completely agree with each other. These conflicts and errors are identified and resolved in the fusion stage. In this step, a set of depth maps from neighboring camera positions are combined into a single fused depth map for one of the views. Among the advantages of the fusion step is that it produces a more compact representation of the data because the number of fused depth maps that are outputted is a small fraction of the original number of depth maps. Much of the information in the original depth maps is redundant since many nearby viewpoints observe the same surface. We opted for this type of viewpoint-based approach since its computational complexity has a fixed upper-bound, for a given Q, regardless of the size of the scene to be modeled. Many of the surface fusion algorithms reported in the literature are limited to single objects and are not applicable to our datasets due to computation and memory requirements.

After fusion, a polygonal model is constructed by a simple module that produces a multi-resolution triangular mesh using a simplified version of the quad-tree approach of Pajarola [45]. The same module also detects and merges overlapping surfaces between consecutive fused depth maps and fills holes.

7.1 Visibility-Based Depth Map Fusion

The input to the fusion step is a set of Q depth maps denoted by $D_1(\mathbf{x}), D_2(\mathbf{x}), \ldots, D_Q(\mathbf{x})$ which record the estimated depth of pixel $\mathbf{x}^\dagger$ of the Q images. Each depth map has an associated confidence map labeled $c_1(\mathbf{x}), c_2(\mathbf{x}), \ldots, c_Q(\mathbf{x})$ computed according to (28). One of the viewpoints, typically the central one, is selected as the reference viewpoint. We seek a depth estimate for each pixel of the reference view. The current estimate of the 3D point seen at pixel $\mathbf{x}$ of the reference view is called $\hat{F}(\mathbf{x})$. $R_i(\mathbf{X})$ is the distance between the center of projection of viewpoint i and the 3D point $\mathbf{X}$. To simplify the notation, we define the term $\hat{f}(\mathbf{x}) \equiv R_{ref}(\hat{F}(\mathbf{x}))$ which is the distance of the current depth estimate $\hat{F}(\mathbf{x})$ for the reference camera.

The first step of fusion is to render each depth map into the reference view. When multiple depth values project onto the same pixel, the nearest depth is kept. Let D_i^{ref} be the depth map D_i rendered into the reference view and c_i^{ref} be the confidence map rendered in the reference view. Given a 3D point $\mathbf{X}$, we need a notation to describe the value of the depth map D_i at the location where $\mathbf{X}$ projects into view i. Let $P_i(\mathbf{X})$ be the image coordinates of the 3D point $\mathbf{X}$ projected into view i. To simplify the notation, the following definition is used $D_i(\mathbf{X}) \equiv D_i(P_i(\mathbf{X}))$. $D_i(\mathbf{X})$ is likely to be different from $R_i(\mathbf{X})$ which is the distance between $\mathbf{X}$ and the camera center.

[†] $\mathbf{x}$ is used in this section instead of (x, y) to denote pixel coordinates in order to simplify the notation

Our approach considers three types of visibility relationships between hypothesized depths in the reference view and computed depths in the other views. These relations are illustrated in Fig. 13(a). The point A' observed in view i is behind the point A observed in the reference view. There is a conflict between the measurement and the hypothesized depth since view i would not be able to observe A' if there truly was a surface at A. We say that A violates the free space of A'. This occurs when $R_i(A) < D_i(A)$.

In Fig. 13(a), B' is in agreement with B since they are in the same location. In practice, we define points B and B' as being in agreement when $\frac{|R_{ref}(B) - R_{ref}(B')|}{R_{ref}(B)} < \epsilon$.

The point C' observed in view i is in front of the point C observed in the reference view. There is a conflict between these two measurements since it would be impossible to observe C if there truly was a surface at C'. We say that C' occludes C. This occurs when $D_i^{ref}(\mathbf{C}') < \hat{f}(\mathbf{C}) = D_{ref}(\mathbf{C})$.

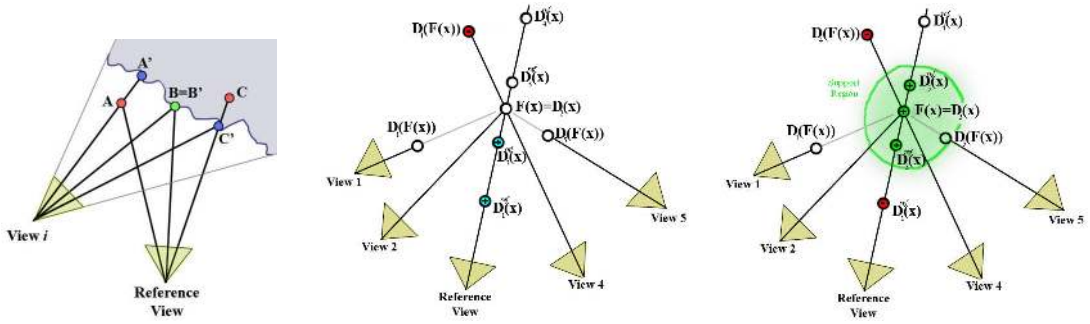
Note that operations for a pixel are not performed on a single ray, but on rays from all cameras. Occlusions are defined on the rays of the reference view, but free space violations are defined on the rays of the other depth maps. The reverse depth relations (such as A behind A' or C in front of C') do not represent visibility conflicts.

The raw stereo depth maps give different estimates of the depth at a given pixel in the reference view. We first present a method that tests each of these estimates and selects the most likely candidate by exhaustively considering all occlusions and free-space constraints. We then present an alternative approach that selects a likely candidate upfront based on the confidence and then verifies that this estimate agrees with most of the remaining data. The type of computations required in both approaches are quite similar. Most of the computation time is spent rendering a depth map seen in one viewpoint into another viewpoint. These computations can be performed efficiently on the GPU.

7.2 Algorithm 1: Stability-Based Fusion

If a depth map occludes a depth hypothesis $\hat{F}(\mathbf{x})$, this indicates that the hypothesis is too far away from the reference view. If the current depth hypothesis violates a free-space constraint, this indicates the hypothesis is too close to the reference view. The stability of a point $S(\mathbf{x})$ is defined as the number of depth maps that occlude $\hat{F}(\mathbf{x})$ minus the number of free-space violations. Stability measures the balance between these two types of visibility violations. If the stability is negative, then most of the depth maps indicate that $\hat{F}(\mathbf{x})$ is too close to the camera to be correct. If the stability is positive then at least half of the depth maps indicate that $\hat{F}(\mathbf{x})$ is far enough away from the reference camera. Stability generally increases as the point moves further away from the camera. The final fused depth is selected to be the closest depth to the camera for which stability is non-negative. This depth is not the median depth along the viewing ray since free-space violations are defined on rays that do not come from the reference view. This depth is balanced in the sense that the amount of evidence that indicates it is too close is equal to the amount of evidence that indicates it is too far away.

With this goal in mind, we construct an algorithm to find the closest stable depth. To



(a) Visibility relations (b) Stability calculation (c) Support estimation

Figure 13: (a) Visibility relations between points. The point A' seen in view i has its free space violated by A seen in the reference view. B' supports B. C seen in the reference view is occluded by C'. (b) Stability Calculation. In this example, there are two occlusions which raise stability and one free-space violations which lowers it. The stability is +1. (c) Support calculation. Three measurements are close to the current estimate and add support to it. Outside the support region, there is one occlusion and one free-space violation which lower the support.

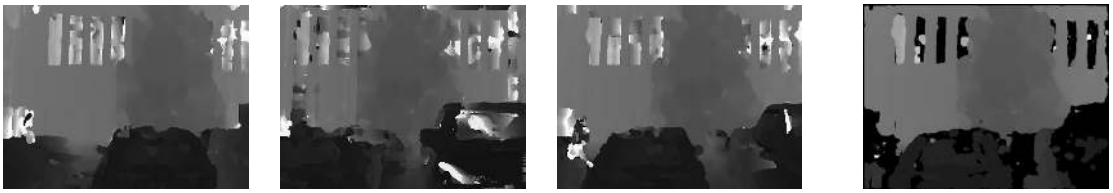


Figure 14: A few raw depth maps and the fused depth map (far right) using confidence-based fusion. 11 total depth maps were used for this computation

begin, all of the depth maps are rendered into the reference view. In the example shown in Fig. 13(b), five depth maps are rendered into the reference view. The closest depth is selected as the initial estimate. In the example, the closest depth is $D_1^{ref}(\mathbf{x})$ and so its stability is evaluated first. The point is tested against each depth map to determine if the depth map occludes it or if it violates the depth map's free space. If the depth estimate is found to be unstable, we move onto the next closest depth. Since there are Q possible choices, the proper depth estimate is guaranteed to be found after $Q-1$ iterations. The total number of depth map renderings is bound by $O(Q^2)$. In the example, the closest two depths $D_1^{ref}(\mathbf{x})$ and $D_3^{ref}(\mathbf{x})$ were tested first. Figure 13(b) shows the test being performed on the third closest depth $D_2^{ref}(\mathbf{x})$. A free-space violation and two occlusions are found and thus the stability is positive. In this example, $D_2^{ref}(\mathbf{x})$ is the closest stable depth.

The final step is to compute a confidence value for the estimated depth. The distance to the selected depth $R_i(\hat{F}(\mathbf{x}))$ is compared with the estimate in depth map i given by $D_i(\hat{F}(\mathbf{x}))$. If these values are within ϵ, the depth map supports the final estimate. The confidences of all the estimates that support the selected estimate are added. The resulting fused confidence map is passed on to the mesh generation module. An example of a few input depth maps and the resulting fused depth map can be seen in Fig. 14.

7.3 Algorithm 2: Confidence-Based Fusion

Stability-based fusion tests up to $Q-1$ different depth hypotheses. In practice, most of these depth hypotheses are close to one another, since the true surface is likely to be visible and correctly reconstructed in several depth maps. Instead of testing so many depth estimates, an alternative approach is to combine multiple close depth estimates into a single estimate and then perform only one test. Because there is only one hypothesis to test, there are only $O(Q)$ renderings to compute. This approach is typically faster than stability-based fusion which tests $Q-1$ hypotheses and computes $O(Q^2)$ renderings, but the early commitment may cause additional errors.

Combining Consistent Estimates Confidence-based fusion also begins by rendering all the depth maps into the reference view. The depth estimate with the highest confidence is selected as the initial estimate for each pixel. At each pixel $\mathbf{x}$, we keep track of two quantities which are updated iteratively: the current depth estimate and its level of support. Let $\hat{f}_0(\mathbf{x})$ and $\hat{c}_0(\mathbf{x})$ be the initial depth estimate and its confidence value. $\hat{f}_k(\mathbf{x})$ and $\hat{c}_k(\mathbf{x})$ are the depth estimate and its support at iteration k, while $\hat{F}(\mathbf{x})$ is the corresponding 3D point.

If another depth map $D_i^{ref}(\mathbf{x})$ produces a depth estimate within ϵ of the initial depth estimate $\hat{f}_0(\mathbf{x})$, it is very likely that the two viewpoints have correctly reconstructed the same surface. In the example of Fig. 13c, the estimates $D_3(\hat{F}(\mathbf{x}))$ and $D_5(\hat{F}(\mathbf{x}))$ are close to the initial estimate. These close observations are averaged into a single estimate. Each observation is weighted by its confidence according to the following equations:

$$\hat{f}_{k+1}(\mathbf{x}) = \frac{\hat{f}_k(\mathbf{x})\hat{c}_k(\mathbf{x}) + D_i^{ref}(\mathbf{x})c_i(\mathbf{x})}{\hat{c}_k(\mathbf{x}) + c_i(\mathbf{x})} \tag{29}$$

$$\hat{c}_{k+1}(\mathbf{x}) = \hat{c}_k(\mathbf{x}) + c_i(\mathbf{x}) \tag{30}$$

The result is a combined depth estimate $\hat{f}_k(\mathbf{x})$ at each pixel of the reference image and a support level $\hat{c}_k(\mathbf{x})$ measuring how well the depth maps agree with the depth estimate. The next step is to find how many of the depth maps contradict $\hat{f}_k(\mathbf{x})$ in order to verify its correctness.

Conflict Detection The total amount of support for each depth estimate must be above the threshold c_{thres} or else it is discarded as an outlier and is not processed any further. The remaining points are checked using visibility constraints. Figure 13 shows that $D_1(\hat{F}(\mathbf{x}))$ and $D_3(\hat{F}(\mathbf{x}))$ occlude $\hat{F}(\mathbf{x})$. However, $D_3(\hat{F}(\mathbf{x}))$ is close enough (within ϵ) to $\hat{F}(\mathbf{x})$ to be within its support region and so this occlusion does not count against the current estimate. $D_1(\hat{F}(\mathbf{x}))$ is occluding $\hat{F}(\mathbf{x})$ outside the support region and thus contradicts the current estimate. When such an occlusion takes place the support of the current estimate is decreased by:

$$\hat{c}_{k+1}(\mathbf{x}) = \hat{c}_k(\mathbf{x}) - c_i^{ref}(\mathbf{x}) \tag{31}$$

When a free-space violation occurs outside the support region, as shown with the depth $D_4(\hat{F}(\mathbf{x}))$ in Fig. 13, the confidence of the conflicting depth estimate is subtracted from the support according to:

$$\hat{c}_{k+1}(\mathbf{x}) = \hat{c}_k(\mathbf{x}) - c_i(P_i(\hat{F}(\mathbf{x}))) \tag{32}$$

We have now added the confidence of all the depth maps that support the current depth estimate and subtracted the confidence of all those that contradict it. If the support is positive, the majority of the evidence supports the depth estimate and it is kept. If the support is negative, the depth estimate is discarded as an outlier. The fused depth map at this stage contains estimates with high confidence and holes where the estimates have been rejected.

Hole filling in fused depth map After discarding the outliers, there are holes in the fused depth map. In practice, the depth maps of most real-world scenes are piecewise smooth and we assume that any small missing parts of the depth map are most likely to have a depth close to their neighbors. To fill in the gaps, we find all inliers within a $w \times w$ window centered at the pixel we wish to estimate. If there are enough inliers to make a good estimate, we assign the median of the inliers as the depth of the pixel. If there are only a few neighboring inliers, the depth map is left blank. Essentially, this is a median filter that ignores the outliers. In the final step, a median filter with a smaller window w_s is used to smooth out the inliers.

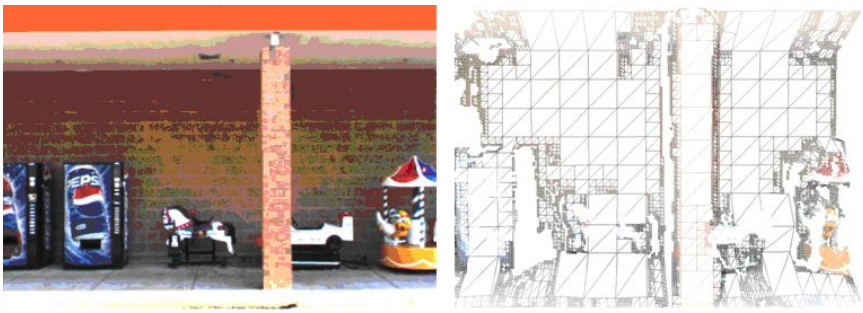


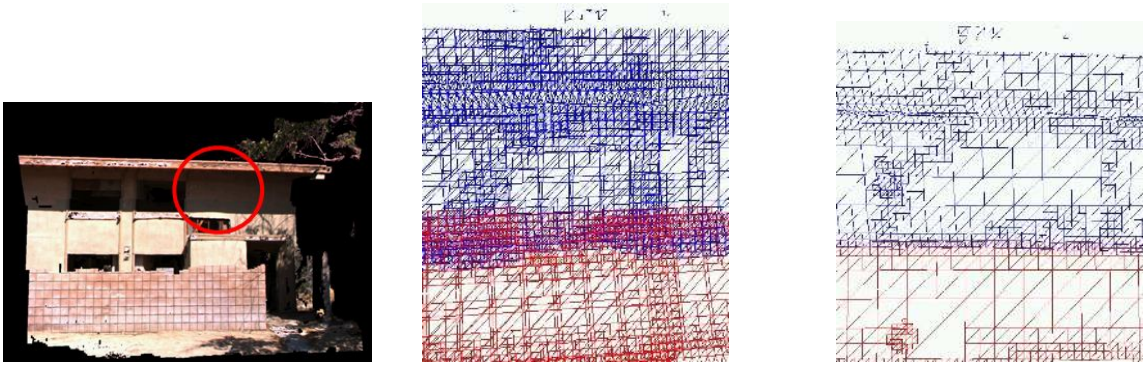
Figure 15: Left: Input image. Right: Illustration of multi-resolution triangular mesh

7.4 Model Generation from the Depth Maps

We have presented two algorithms for generating fused depth maps. In this section we show how to convert this viewpoint-based representation into a 3D model. Consecutive fused depth maps partially overlap one another and the overlapping surfaces are unlikely to be aligned perfectly. The desired output of our system, is a smooth and consistent model of the scene without artifacts in the form of small gaps that are caused by this misalignment. To this end, consistency between consecutive fused depth maps is enforced in the final model. Each fused depth map is compared with the previous fused depth map as it is being generated. If a new estimate violates the free space of the previous fused depth maps, the new estimate is rejected. If a new depth estimate is within ϵ of the previous fused depth map, the two estimates are merged into one vertex which is generated only once in the output. Thus redundancy is removed along with any gaps in the model where two representations of the same surface are not connected. More than one previous fused depth map should be kept in memory to properly handle surfaces that disappear and become visible again. In most cases, two previous fused depth maps are sufficient.

After duplicate surface representations have been merged, a mesh is constructed taking into account the corresponding confidence map to suppress any remaining outliers. By using the image plane as a reference both for geometry and for appearance, we can construct the triangular mesh very quickly. We employ a multi-resolution quad-tree algorithm in order to minimize the number of triangles while maintaining geometric accuracy as in [45]. We use a top-down approach rather than a bottom-up approach to lower the number of triangles that need to be constructed and processed. Starting from a coarse resolution, we form triangles and test if the triangles correspond to nonplanar parts of the depth map, if they bridge depth discontinuities or if points with low confidence (below c_{thres}) are included in them. If any of these events occur, the quad, which is formed out of two adjacent triangles, is subdivided. The process is repeated on the subdivided quads up to the finest resolution. A part of a multi-resolution mesh can be seen in Figure 15.

We use the following simple planarity test proposed by Pajarola [46] for each vertex of each triangle:



(a) Screenshot of model (b) Overlapping triangles (c) After removal of duplicates

Figure 16: Duplicate surface removal for the circled area of the model in (a), which consists of six sub-models, three each from the side and upward cameras. (b) shows the overlapping meshes as wireframes. Triangles in red are generated by the side camera and the ones is blue by the upward camera. The results of duplicate surface removal are shown in (c).

$$\left| \frac{z_{-1} - z_0}{z_{-1}} - \frac{z_0 - z_1}{z_1} \right| < t. \tag{33}$$

Where z_0 is the z-coordinate, in the camera coordinate system, of the vertex being tested and t is a threshold. z_{-1} and z_1 are the z-coordinates of the two neighboring vertices of the current vertex on an image row. (The distance between the corresponding pixels of two neighboring vertices is equal to the size of the quad's edges.) The same test is repeated along an image column. If either the vertical or the horizontal tests fails for any of the vertices of the triangle, the triangle is rejected since it is not part of a planar surface and the quad is subdivided. For these tests, we have found that 3D coordinates are more effective than disparity values. Since we do not require a manifold mesh and are interested in fast processing speeds, we do not maintain a restricted quad-tree [45].

An illustration of the duplicate surface removal process can be seen in Figure 16. The model shown in Figure 16(a) was reconstructed using the side and upward camera. Since the cameras are synchronized an the relative transformation from one coordinate system to the other are known, we can register the partial reconstructions in the same coordinate system. Figure 16(b) shows the overlapping meshes that are generated for different fused depth maps from the same video stream, as well as from different video streams. Our scheme is able to remove most of the duplicate representations and produce a simpler mesh shown in Figure 16(c).

Despite its simplicity, this scheme is effective when the camera maintains a dominant direction of motion. It does not handle, however, the case of a part of the scene being revisited in a later pass, since the entire reconstructed model cannot be kept in memory. A more efficient representation for parts of the model potentially in the form of a set of bounding boxes is among our future research directions.

(a) Model with sky artifacts (b) Same model after sky detection (c) Detail

Figure 17: Sky detection and removal to improve the visual quality of the models. Notice that no parts of the building, ground and trees have been removed.

Figure 18: Example of hole-filling. Left: screenshot or original reconstruction. Right: the windows that are not photo-consistent have been filled-in by our algorithm.

7.5 Model Clean-up and Hole-filling

In this section we present techniques for improving the visual quality of the models. The first technique aims at removing visually annoying artifacts. A typical artifact of stereo reconstruction is the "foreground fattening" effect. The extent of textured foreground surfaces is often over-estimated when uniform parts of the image, that are partially occluded in some of the views, appear next to them. This occurs in our reconstruction where the sky often appears to be attached to the edges of buildings. We designed a simple module that learns the color distribution of the sky offline via k-means clustering and then suppresses pixels that are most likely sky from the final model (Figure 17). To improve processing speed, sky detection is performed during image input and pixels marked as sky are not processed further by any other module.

After surface merging and sky removal, most of the remaining artifacts are holes in the facades. These typically occur in parts of the surfaces that were not photo-consistent such as windows and other reflective materials. We have developed a technique to detect surfaces with holes and fill them with planes after verifying geometric constraints. These ensure that the surface is added in a planar hole, which is completely enclosed by surfaces. The normal of the added plane has to be consistent with the normals of the surfaces it abuts and within 45° of the viewing ray from the camera. An example in which two windows are filled in can be seen in Figure 18.

8 Real-time Implementation

In this section we describe a real-time implementation of our system that operates on 512×384 video streams collected at 30 frames per second. Please refer to Figure 2 for a flowchart of the processing pipeline. In order to maximize throughput, we take the following steps on a high-end PC featuring a dual-core AMD Opteron processor at $2.4GHz$ and an NVidia GeForce 8800 series GPU:

- We use the GPS/INS data exclusively for pose estimation, thus saving the computational cost for 2D and 3D tracking and image flow based frame decimation. Unless rare failures of the GPS/INS system occur, this provides pose estimates of adequate quality for stereo. As mentioned in Sec. 5.3, the improvement in reprojection error caused by the Kalman filter is in general a small fraction of a pixel.

- Reading the images from the disk is performed by a separate thread, while frames with no or very little motion according to the GPS/INS data are not read at all. The thread also constructs a single level of the grayscale pyramid on which stereo is performed. This thread causes no delays to the main thread.

- Stereo is performed using 7 images and 48 fronto-parallel planes. No scene analysis is performed since sparse data are not computed. A depth map is computed for every frame in $24ms$.

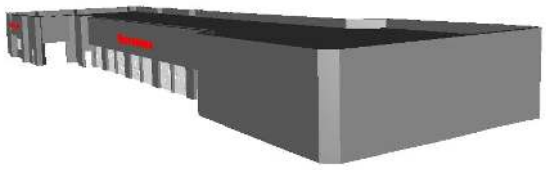
- Depth map fusion is performed on 17 half-resolution depth maps every 16 frames. Each computation takes $78ms$, or $4.88ms$ per frame.

- Model generation is performed for each fused depth map and takes $30ms$, or $1.88ms$ per frame. The JPEG image used for texture mapping is written by a separate thread.

The other parameters were set to the following values: the stereo window size to 16×16, the threshold for the planarity test of Equation (33) to $t = 0.05$, the maximum quad size for mesh construction to 16×16 pixels and the minimum to 2×2 pixels. Using these settings, our pipeline achieves $32.5Hz$. In practice, several frames are decimated due to small motion of the vehicle and the throughput can be even higher, especially when the vehicle velocity does not exceed $30km/h$. The size of the model is approximately $5.5MB$, including both geometry and images for texture-mapping, per 100 frames of video of an urban area using the above settings.

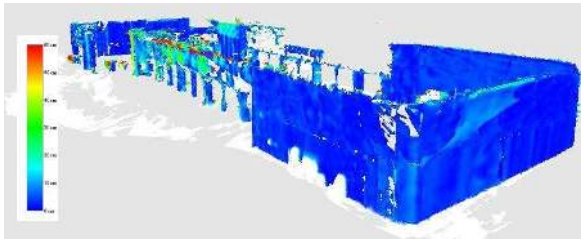
9 Results

To evaluate the ability of our method to reconstruct urban environments, a 3,000 frame video of the exterior of a Firestone store was captured with two cameras to obtain a more complete reconstruction. One of the cameras was horizontal and the other was pointed

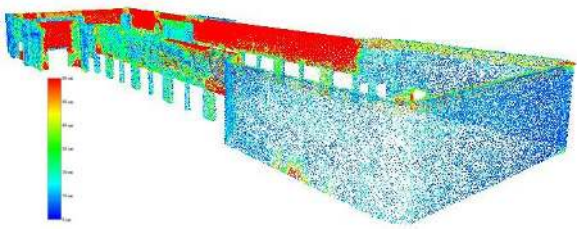


(a) Surveyed model of the Firestone building

(b) Reconstructed model using fusion



(c) Visualization of the accuracy evaluation, where white indicates parts of the model that have not been surveyed and blue, green and red indicate errors of $0cm$, $30cm$ and $60cm$ or above, respectively.



(d) Completeness of the Firestone building. The color coding is the same as in (c). Red areas mostly correspond to unobserved or untextured areas.

Figure 19: Firestone building accuracy and completeness evaluation

up 30°. The videos from each camera were processed separately. The Firestone building was surveyed to an accuracy of 6 mm and the reconstructed model (Figure 19(b)) was directly compared with the surveyed model (Figure 19(a)). There are several objects such as parked cars that are visible in the video, but were not surveyed. Several of the doors of the building were left open causing some of the interior of the building to be reconstructed. The ground which slopes away from the building also was not surveyed. Since accurate measurements of all of these reconstructed objects were unavailable they are manually removed from the evaluation. To measure the accuracy of each reconstructed vertex, the distance from the vertex to the nearest triangle of the ground truth model is calculated. The error measurements for each part of a reconstruction after fusion are displayed in Fig. 19(c). Completeness measures how much of the building was reconstructed and is defined in a way similar to [55]. Sample points are chosen at random on the surface of the ground truth model in a way that ensures that there are 50 sample points per square meter of surface area. The distance from each sample point to the nearest reconstructed point is measured. A visualization of these distances are shown for one of the reconstructions in Figure 19(d).

We performed a quantitative evaluation between raw stereo depth maps and the results of the fusion algorithm. For the results labeled as *stereo-reference*, we evaluate the raw depth maps from each of the reference views. For the results labeled *stereo-exhaustive*, we evaluated the depth maps from *all* images as the representation of the scene. Table 1 contains the median and mean error values for each method as well as the completeness achieved on the Firestone building using the horizontal and upward-facing cameras.

(a) Screenshots of reconstructed buildings

(b) View from above and details of large scale reconstruction

Figure 20: Screenshots of 3D models. All models are produced by more than one camera using GPS/INS data, except the one at the top left which is a vision-only reconstruction from a single camera.

Fusion Method	Stereo-exhaustive)	Stereo-reference)	Confidence	Stability
Median Error(cm)	4.87	4.19	2.60	2.19
Mean Error(cm)	40.61	39.20	6.60	4.79
Completeness	94%	83%	73%	66%

Table 1: Accuracy and Completeness for different fusion methods using the default parameters (both cameras).

We also conducted an experiment using different levels of compression for the input videos. The option to compress the data on-board the capturing platform is intriguing since it can significantly reduce the storage space and bandwidth requirements of the collection and processing systems. We repeated the evaluation procedure on fused depth maps generated using inputs that were compressed either according to the JPEG standard to file sizes up to 12 times smaller than the uncompressed data or using the MPEG-4 standard to file sizes up to 100 times smaller than the original. The evaluation results were very similar compared to those on uncompressed data while the appearance of the models was virtually indistinguishable.

Our methods were applied on several videos of urban environments. Screenshots of reconstructed models are shown in Figure 20. The model at the lower left was created from a single camera using the vision-only version of the processing pipeline. The other models are geo-registered reconstructions from multiple video inputs. A very long 170,000 frame video was used to reconstruct a large model shown in Figure 1. These videos can be processed at between $32.5Hz$ using the settings of Sec. 8. Even if we tune parameters such as the number of planes for stereo in order to maximize quality, the processing rate of our system remains in the range of several frames per second.

10 Discussion

We have presented a real-time system for 3D reconstruction from video. It produces dense, geo-registered, 3D models from video captured by a multi-camera system in conjunction with INS/GPS measurements. Our approach is aimed at reconstructing ground-based 3D models of complete cities which yields a number of significant challenges, including the shear size of the video data to be processed, the large variability of illumination, the varying distance and orientation of the observed scene and the presence of objects that are hard to model, such as trees and windows. Despite these challenges, we think that we have achieved reconstructions that provide effective visualization of the environment.

The real-time processing rate was achieved by strict selection of the algorithms used and by leveraging the processing power of modern commodity graphics cards. Instead of trying to achieve the best possible results from a minimum set of data, we prefer to focus our effort on leveraging the redundancy of data (each surface element is typically seen in tens of views) to obtain consensus results supported by a lot of data fast. We have shown

in this paper that our two-stage approach to recover dense surface reconstructions from multi-view stereo followed by depth map fusion was very effective both in terms of speed and accuracy. Experiments with ground-truth data show that our real-time approach is able to recover the surfaces of large scale buildings with an accuracy of a few centimeters.

Another important aspect of our work was to develop algorithms that could effectively deal with the large brightness variation of outdoor scenes. Our approach consists of using auto-exposure to guarantee good image quality. Our 2D tracker was adapted to track gain changes throughout the video sequences and the multi-view stereo was adapted to compensate for gain changes. Care was taken to allow a maximum of flexibility without impacting performance.

While urban scenes typically contain a lot of structure, they also contain a lot of unstructured elements. In the past some approaches have sought to enforce strong priors on the reconstructed scene, sacrificing generality. In contrast, while the approach proposed in this paper was optimized for both performance and quality on urban structures, we also made sure it would provide results similar to more generic approaches on other parts of the scene such as trees. In fact, trees are a major challenge in urban scenes, not only because they are hard to model, but because they occlude the facades of buildings. In our approach this problem is attenuated by the redundancy of video data. We can still reconstruct a surface behind a tree if it is visible in at least a few views even if it is occluded in the majority of views.

A challenge related to the visualization of the models has been their size. A fixed, limited number of frames is processed by the different modules at each step allowing us to generate models using up to hundreds of thousands of frames as input without any special considerations in terms of processing time per frame or memory requirements. Visualizing a model of this size, however, is not as straightforward since rendering millions of polygons stretches the limits even of high-end computers and graphics adapters. A side-effect of processing in sliding windows is that the output consists of several sub-models which can be viewed as a tiling of the scene and can be loaded and unloaded from the 3D model viewer's memory as the user navigates. This does not eliminate the problem of displaying zoomed out views of very large models, since we currently do not provide levels of detail for the representation. Both the issue of loading and unloading partial models and the generation of levels of detail should be addressed in future work.

Future challenges include a scheme for better utilization of the multiple video streams to improve the quality of pose estimation and to reconstruct each surface under the optimal viewing conditions, while reducing the redundancy of the overall model. We also intend to improve the visual quality of our models by better handling surfaces with complex view-dependent appearance, like windows, and moving objects in the scene. Note that by incorporating plane priors based on the sparse structure we were able to provide a partial solution to the problem of untextured surfaces.

Along a different axis, we intend to investigate ways of decreasing the cost and size of our system by further exploring implementations that use low-end INS and GPS systems or even only GPS instead of the current system which is very accurate but also expensive and cumbersome. Beyond this a challenging but exciting area of further research is in the

area of large-scale structure from motion. Wide-baseline feature matching techniques can potentially be used to efficiently identify when the same part of the scene is revisited and would thus allow to close the loop and remove drift. This problem is also related to the Simultaneous Localization and Mapping (SLAM) problem in robotics. The same matching techniques could potentially also be used to provide geo-location in the absence of GPS by registering the current reconstruction with a database of geo-located images. Beyond the problem of matching across imagery, large scale datasets also pose other challenges such as the efficient solution of the non-linear least-squares structure and motion refinement aka bundle adjustment.

Potential longer term directions are change detection and the capability to perform incremental model updates using video acquired at different times.

Acknowledgment

We gratefully acknowledge the support of the DARPA UrbanScape project as well as the support of the DTO VACE project "3D Content Extraction from Video Streams".

References

[1] A. Akbarzadeh, J.-M. Frahm, P. Mordohai, B. Clipp, C. Engels, D. Gallup, P. Merrell, M. Phelps, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewenius, R. Yang, G. Welch, H. Towles, D. Nister, and M. Pollefeys. Towards Urban 3D Reconstruction From Video. In *Proceedings of International Symposium on 3D Data, Processing, Visualization and Transmission*, 2006.

[2] American Society of Photogrammetry. *Manual of Photogrammetry (5th edition)*. Asprs Pubns, 2004.

[3] A. Azarbayejani and A.P. Pentland. Recursive estimation of motion, structure, and focal length. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(6):562–575, 1995.

[4] S. Baker, R. Gross, I. Matthews, and T. Ishikawa. Lucas-kanade 20 years on: A unifying framework: Part 2. Technical Report CMU-RI-TR-03-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 2003.

[5] P. Beardsley, A. Zisserman, and D. Murray. Sequential updating of projective and affine structure from motion. *Int. J. Computer Vision*, 23(3):235–259, Jun-Jul 1997.

[6] P. Biber, S. Fleck, D. Staneker, M. Wand, and W. Strasser. First experiences with a mobile platform for flexible 3d model acquisition in indoor and outdoor environments – the waegele. In *ISPRS Working Group V/4: 3D-ARCH*, 2005.

[7] S. Birchfield and C. Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *Int. Conf. on Computer Vision*, pages 489–495, 1999.

[8] M. Bosse, R. Rikoski, J. Leonard, and S. Teller. Vanishing points and 3d lines from omnidirectional video. *The Visual Computer*, 19(6):417–430, 2003.

[9] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001.

[10] R. G. Brown and P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering, 3rd Edition*. John Wiley and Sons, New York, 1997.

[11] P. Burt, L. Wixson, and G. Salgian. Electronically directed "focal" stereo. In *Int. Conf. on Computer Vision*, pages 94–101, 1995.

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

[13] N. Cornelis, K. Cornelis, and L. Van Gool. Fast compact city modeling for navigation pre-visualization. In *Int. Conf. on Computer Vision and Pattern Recognition*, 2006.

[14] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *SIGGRAPH*, 30:303–312, 1996.

[15] S.F. El-Hakim, J.-A. Beraldin, M. Picard, and A. Vettore. Effective 3d modeling of heritage sites. In *4th International Conference of 3D Imaging and Modeling*, pages 302–309, 2003.

[16] O. Faugeras, Q.-T. Luong, and S. Maybank. Camera self-calibration: Theory and experiments. In *European Conf. on Computer Vision*, pages 321–334. Springer-Verlag, 1992.

[17] O.D. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.

[18] A. Fischer, T.H. Kolbe, F. Lang, A.B. Cremers, W. Förstner, L. Plümer, and V. Steinhage. Extracting buildings from aerial images using hierarchical aggregation in 2d and 3d. *Computer Vision and Image Understanding*, 72(2):185–203, 1998.

[19] A. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. In *European Conf. on Computer Vision*, pages 311–326, 1998.

[20] C. Früh and A. Zakhor. An automated method for large-scale, ground-based city model acquisition. *Int. J. of Computer Vision*, 60(1):5–24, 2004.

[21] P.V. Fua. From multiple stereo views to multiple 3-d surfaces. *Int. Journ. of Computer Vision*, 24(1):19–35, 1997.

[22] D. Gallup, J.-M. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. In *Int. Conf. on Computer Vision and Pattern Recognition*, 2007.

[23] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97*, pages 209–216, 1997.

[24] M. Goesele, B. Curless, and S. M. Seitz. Multi-view stereo revisited. *CVPR*, 2:2402–2409, 2006.

[25] M. S. Grewal and A. P. Andrews. *Kalman Filtering Theory and Practice Using MATLAB, 2nd Edition*. John Wiley and Sons, New York, 2001.

[26] A. Gruen and X. Wang. Cc-modeler: A topology generator for 3-d city models. *ISPRS Journal of Photogrammetry & Remote Sensing*, 53(5):286–295, 1998.

[27] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[28] Richard I. Hartley and Peter Sturm. Triangulation. *Computer Vision and Image Understanding: CVIU*, 68(2):146–157, 1997.

[29] A. Hilton, A.J. Stoddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *European Conf. on Computer Vision*, pages 117–126, 1996.

[30] D. Hoiem, A.A. Efros, and M. Hebert. Putting objects in perspective. In *Int. Conf. on Computer Vision and Pattern Recognition*, pages 2137–2144, 2006.

[31] H. Jin, P. Favaro, and S. Soatto. Real-time feature tracking and outlier rejection with changes in illumination. In *Int. Conference on Computer Vision*, pages 684–689, 2001.

[32] S.B. Kang, R. Szeliski, and J. Chai. Handling occlusions in dense multi-view stereo. In *Int. Conf. on Computer Vision and Pattern Recognition*, pages 103–110, 2001.

[33] S.J. Kim, D. Gallup, J.-M. Frahm, A. Akbarzadeh, Q. Yang, R. Yang, D. Nistér, and M. Pollefeys. Gain adaptive real-time stereo streaming. In *Int. Conf. on Vision Systems*, 2007.

[34] R. Koch, M. Pollefeys, and L. Van Gool. Robust calibration and 3d geometric modeling from large collections of uncalibrated images. In *DAGM*, pages 413–420, 1999.

[35] R. Koch, M. Pollefeys, and L.J. Van Gool. Multi viewpoint stereo from uncalibrated video sequences. In *European Conf. on Computer Vision*, volume I, pages 55–71, 1998.

[36] B.D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Int. Joint Conf. on Artificial Intelligence*, pages 674–679, 1981.

[37] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J.-M. Frahm, R. Yang, D. Nister, and M. Pollefeys. Real-Time Visibility-Based Fusion of Depth Maps. In *Proceedings of International Conf. on Computer Vision*, 2007.

[38] L.P. Morency, A. Rahimi, and T.J. Darrell. Fast 3d model acquisition from stereo images. In *3D Data Processing, Visualization and Transmission*, pages 172–176, 2002.

[39] D.D. Morris and T. Kanade. Image-consistent surface triangulation. In *Int. Conf. on Computer Vision and Pattern Recognition*, pages I: 332–338, 2000.

[40] P.J. Narayanan, P.W. Rander, and T. Kanade. Constructing virtual worlds using dense stereo. In *Int. Conf. on Computer Vision*, pages 3–10, 1998.

[41] D. Nistér. Preemptive RANSAC for live structure and motion estimation. In *Int. Conf. on Computer Vision*, volume 1, pages 199–206, 2003.

[42] D. Nistér. An efficient solution to the five-point relative pose problem. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(6):756–777, 2004.

[43] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23(1), 2006.

[44] A.S. Ogale and Y. Aloimonos. Stereo correspondence with slanted surfaces: Critical implications of horizontal slant. In *Int. Conf. on Computer Vision and Pattern Recognition*, pages 568–573, 2004.

[45] R. Pajarola. Overview of quadtree-based terrain triangulation and visualization. Technical Report UCI-ICS-02-01, Information & Computer Science, University of California Irvine, 2002.

[46] R. Pajarola, Y. Meng, and M. Sainz. Fast depth-image meshing and warping. Technical Report UCI-ECE-02-02, Information & Computer Science, University of California Irvine, 2002.

[47] M. Pollefeys, R. Koch, and L. Van Gool. Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters. *Int. J. of Computer Vision*, 32(1):7–25, 1999.

[48] A. Román, G. Garg, and M. Levoy. Interactive design of multi-perspective images for visualizing urban landscapes. In *IEEE Visualization*, pages 537–544, 2004.

[49] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3d model acquisition. *ACM Trans. on Graphics*, 21(3):438–446, 2002.

[50] T. Sato, M. Kanbara, N. Yokoya, and H. Takemura. Dense 3-d reconstruction of an outdoor scene by hundreds-baseline stereo using a hand-held video camera. *Int. Journ. of Computer Vision*, 47(1-3):119–129, April 2002.

[51] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. of Computer Vision*, 47(1-3):7–42, 2002.

[52] G. Schindler and F. Dellaert. Atlanta world: an expectation maximization framework for simultaneous low-level edge grouping and camera calibration in complex man-made environments. In *Int. Conf. on Computer Vision and Pattern Recognition*, pages 203–209, 2004.

[53] G. Schindler, F. Dellaert, and S.B. Kang. Inferring temporal order of images from 3d structure. In *Int. Conf. on Computer Vision and Pattern Recognition*, 2007.

[54] G. Schindler, P. Krishnamurthy, and F. Dellaert. Line-based structure from motion for urban environments. In *3DPVT*, 2006.

[55] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Int. Conf. on Computer Vision and Pattern Recognition*, pages 519–528, 2006.

[56] J. Shi and C. Tomasi. Good Features to Track. In *Int. Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[57] S. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. Feature tracking and matching in video using programmable graphics hardware. *Machine Vision and Applications*, 2007.

[58] S. Soatto, P. Perona, R. Frezza, and G. Picci. Recursive motion and structure estimation with complete error characterization. In *Int. Conf. on Computer Vision and Pattern Recognition*, pages 428–433, 1993.

[59] M. Soucy and D. Laurendeau. A general surface approach to the integration of a set of range views. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(4):344–358, 1995.

[60] I. Stamos and P.K. Allen. Geometry and texture recovery of scenes of large scale. *Computer Vision and Image Understanding*, 88(2):94–118, 2002.

[61] H. Stewénius, D. Nistér, M. Oskarsson, and K. Åström. Solutions to minimal generalized relative pose problems. In *Workshop on Omnidirectional Vision*, Beijing China, October 2005.

[62] R. Szeliski and D. Scharstein. Sampling the disparity space image. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(3):419–425, 2004.

[63] S. Teller, M. Antone, Z. Bodnar, M. Bosse, S. Coorg, M. Jethwa, and N. Master. Calibrated, registered images of an extended urban area. *Int. Journ. of Computer Vision*, 53(1):93–107, June 2003.

[64] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *SIGGRAPH*, pages 311–318, 1994.

[65] T. Werner and A. Zisserman. New techniques for automated architectural reconstruction from photographs. In *European Conf. on Computer Vision*, pages 541–555, 2002.

[66] M.D. Wheeler, Y. Sato, and K. Ikeuchi. Consensus surfaces for modeling 3d objects from multiple range images. In *Int. Conf. on Computer Vision*, pages 917–924, 1998.

[67] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *Int. Conf. on Computer Vision and Pattern Recognition*, pages 211–217, 2003.

[68] X. Zabulis and K. Daniilidis. Multi-camera reconstruction based on surface normal estimation and best viewpoint selection. In *3DPVT*, 2004.

[69] Z. Zhu, A.R. Hanson, and E.M. Riseman. Generalized parallel-perspective stereo mosaics from airborne video. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(2):226–237, 2004.