ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

MITSUBISHI ELECTRIC
Changes for the Better

Geolocalization using Skylines from Omni-Images

Master Thesis

Sofien Bouaziz[1,2]

[1]EPFL — Ecole Polytechnique Fédérale de Lausanne
VRLAB — Virtual Reality Laboratory

[2]MERL — Mitsubishi Electric Research Laboratories

sofien.bouaziz@gmail.com

2008/2009

Supervisors:
Daniel Thalmann[1]
daniel.thalmann@epfl.ch
Srikumar Ramalingam[2]
ramalingam@merl.com

Abstract

In this thesis a novel technique to accurately estimate the global position of a moving car using an omnidirectional camera and untextured 3D city model is proposed. The key feature in our geolocalization is the skyline, which is essentially the boundary between sky and buildings. The centerpiece of this algorithm is the extraction and matching of these skylines from real images of the city with synthetic ones generated from coarse 3D models. The localization is expected to be very precise if the largest possible skyline is captured and matched. For this reason, an omni-directional camera is employed and oriented towards the sky. The real skyline is extracted from the omni-directional images by first segmenting the sky from the rest. In order to do this a novel graph cuts based segmentation algorithm is proposed where the parameters of the cost function are learned efficiently using a cutting plane algorithm. On the other hand, the synthetic skyline is generated from the 3D model using the calibration parameters from the real omni-directional camera obtained using a generic calibration algorithm. The experimental results are very promising and we believe that our algorithm can be used to develop a vision-based global positioning system (GPS) that can outperform commercially available GPS units in dense urban areas.

Acknowledgements

I want to thank my two supervisors, Prof. Daniel Thalmann and Dr. Srikumar Ramalingam for all their supports and feedbacks. I am especially grateful for the time spent by Srikumar for the proofreading of this thesis. I would like to thank Dr. Matthew Brand, Dr. Jay Thornton and Dr. Peter Sturm for their useful ideas. I also want to thank Mitsubishi Electric Research Laboratories and the Ecole Polytechnique Fédérale de Lausanne for their support.

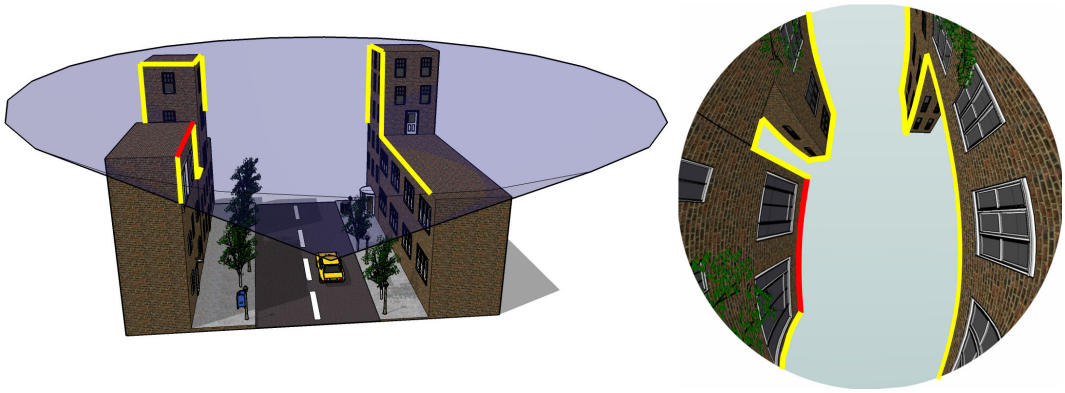
Contents

Chapter 1

Introduction

The skyline has long been a source of fascination for photographers, thought to identify the city as uniquely as a fingerprint. In this work, a step further have been taken to investigate its uniqueness for any given geospatial location. A simple, robust and fast method to compute the geolocation by matching skylines from real fisheye images and synthetic ones from a coarse 3D model of a city is proposed. In other words, the pose estimation problem is solved for an omnidirectional camera in the actual world coordinate system of the 3D model. Pose estimation, although extremely challenging due to several degeneracy problems and inaccurate feature matching, is a well researched topic in computer vision. However most existing solutions are only proven on the smaller scale of the laboratory setup. In this work, the possibility of using the skylines for solving the pose estimation problem is investigated. Such an approach has several interesting properties compared to existing techniques. First, the skyline is compact, unique and extremely informative about a location. Second, it is less prone to occlusions (trees, lampposts, cars and pedestrians) and lighting variations than other features.

The basic idea of matching skylines from 3D models and omni-directional images is shown in Figure 1.1. The main reason to use omni-directional cameras instead of perspective ones come from the fact that the stability of the underlying pose estimation problem increases with the field of view of the camera. It has also been proven formally that omni-directional



(a) (b)

Figure 1.1: *An illustration of the motivating idea. (a) A car auto-mounted with an omni-directional camera facing towards the sky. (b) The image captured by the camera with skylines marked. This image is matched to the synthesized one from the 3D city model to compute the geospatial location of the car.*

cameras can give much better accuracy in motion estimation than perspective ones [19, 5]. In the case of small rigid motions, two different motions can yield nearly identical motion fields for classical perspective cameras, which is not the case for omnidirectional images. This is also obvious in practice as the omni-directional camera allows to acquire the skyline of a complete street in contrast to a perspective camera that can only acquire a small part of it as shown in Figure 1.2.

1.1 Related Work

In the last few years, there has been an increasing interest in inferring geolocation from images [29, 40, 18, 15]. Several recent image based localization approaches have been proposed using feature matching algorithms. The general idea behind most of these approaches is to find the closest images to a given query picture in a database of GPS-tagged images.

In 2004, Robertson and Cipolla [29] showed that it is possible to obtain geospatial localization by wide-baseline matching between a new image from a mobile camera and a database of rectified building facade images. As the feature matching is more efficient and accurate using rectified images, they also rectify the new input image from the mobile camera by computing

(a) (b)

Figure 1.2: *Comparison of a perspective image (a) and an omni-directional image (b) taken from the same location. The perspective image contains only a very small skyline compared to a much larger one from the omni-directional image. As a result, the localization using perspective images can be extremely ambiguous. The marked rectangle shows the overlap between the perspective and the fisheye image.*

the horizontal and vertical vanishing points. The best match is given by the facade image in the database which has the maximum number of robust correspondences with the query image.

In 2006, Zhang and Kosecka [40] showed accurate results by using SIFT (scale invariant feature transform) features to search the closest images in a database of GPS-tagged images. The basic idea is to generate a large quantity of SIFT matches and robustly select the ones lying on the dominant plane. The two nearest images (also individually close to each other) to the query image are selected for motion estimation. In cases where only one reference view can be reliably computed, the location of this image is given to the query image.

In 2007, Jacobs et al. [18] used an interesting approach to geolocate a static outdoor webcam by correlating its images with satellite weather imagery taken at the same time of the day or with images coming from webcams for which the locations are known. The underlying idea is that there is a consistent pattern in the manner in which these images vary over time. Using a simple algorithm that exploits the principal components of a matrix computed from a large database of images, geolocalization has been shown with an ambiguity of 50 miles.

In 2008, Hays and Efros [15] used millions of GPS-tagged images from the web for georeferencing a new image. In order to match the query image with a large collection of images

(6 million images), several features based on color histograms, texton histograms, line features, gist descriptor, geometric context (labeling of the image pixels into sky, vertical and ground), etc. are used. Based on these features, the k-nearest neighbors are computed and a mean-shift clustering is performed on these neighbors based on their GPS coordinates. This enables them to remove spurious outliers and the quantitative evaluation demonstrate a performance up to 30 times better than the prediction by chance.

In contrast to most of these approaches that leverage on the availability of these georeferenced images, the approach presented uses coarse 3D models downloaded from the web. A large repository of coarse 3D models already exists for major cities in the world, thanks to several promising algorithms to reconstruct large urban scenes [37, 7, 12, 2, 1].

There exist few more related approaches that use omni-directional cameras and 3D models for geolocalization [20, 32, 23]. Koch and Teller proposed a localization method using a known 3D model and a wide angle camera for indoor scenes by matching lines from the 3D model with the lines in images [20]. In contrast to their technique, this work relies on the skylines for geolocalization of outdoor scenes. Although, the idea of using the horizon or the skyline has been explored earlier in [32, 23], our approach is different from them in several ways. In [32] a human user input is required to extract the skyline, whereas this work uses an automatic graph cuts algorithm. Moreover, a hash table is precomputed to match the unwrapped horizon with the 3D model, whereas the algorithm presented in this work synthesizes fisheye images on the fly for matching. This relaxes the requirement of pre-processing and storing millions of synthetic skylines and increases the accuracy. Our work is closest to [23], where an omni-directional infrared camera is used instead of omnidirectional visible camera. The main differences between our approach and their work are summarized bellow:

- Their system requires an expensive infrared camera, which makes it impractical for large-scale deployment in consumer oriented applications, such as vehicle or hand-held devices. In addition, they require accurate intra camera parameters and an accurate method of projection. With the IR camera, it is necessary to have infrared rays emitted in certain

patterns. That makes it a challenge to determine a technique for capturing images of cyclical patterns used in camera calibration. Hence, a highly specialized calibration jig with thermal point sources arranged inside is required, which further stands in the way of mass deployment in a consumer market.

- The method requires a high-resolution 3D digital surface model (DSM).

- The method unwraps the infrared images into a rectangular panorama, from which edges are extracted to generate a linear profile of the surrounding buildings. They use an azimuth projection specialized for their camera design. The profile is then correlated with the profiles in the restoration images. Neither the unwrapped infrared profiles nor the profiles in the restoration images directly reflect the actual skyline. As a result, their approach can lead to potential inaccuracies when the camera is not vertically aligned. Their angle of view is restricted to 20-70 degrees, and is limited to only provide a 2D location and 1 D orientation.

1.2 Overview of the algorithm

In Figure 1.3 the four main steps of our approach are shown: fisheye camera calibration, fisheye synthesis, sky detection and skyline matching. The main idea of the algorithm is to match skylines accurately generated from a coarse 3D model using the camera calibration information with real skylines from a video or a set of images. The skyline are extracted from the real images thanks to a sky detection algorithm based on graph cuts where the parameters of the energy function are learned from a training set. The synthetic and the real skylines are matched using a RANSAC (random sample consensus) based approach robust to outliers as deformations of the 3D model or occlusions in the real images. Many images are generated at different positions and orientations in the 3D model and the position and orientation that give the lower matching score is keep as the final location. The four main parts of the algorithm are

summarized bellow.

1.2.1 Camera Calibration

In this thesis all the experimental results use a fisheye lens based omni-directional camera with a field of view of $360^\circ \times 183^\circ$. However, the calibration algorithm used in this work does not make any assumption about the type of omni-directional lenses. In order to propose a method that could work with all kinds of omni-directional cameras a generic calibration approach which treats every camera as a mapping between a pixel and its corresponding projection rays has been used. As a result of this generality, it is easier to change the camera configuration and still obtain the geolocalization. This general model has been studied in various works [26, 14, 25, 27, 33, 28] and it was recently shown that the generic calibration algorithm outperforms standard parametric approaches in the case of very high distortions [28, 8].

1.2.2 Fisheye Synthesis

Although synthesis of non-linear projection such as fisheye projection have been studied in computer graphics since a long time and are used in different type of applications as immersive display, computer games or scientific visualization, we are not aware of any work where fisheye images are synthesized using the generic calibration information and where fisheye synthesis has been used for geolocalization. Further, the fisheye synthesis is also difficult because of the limitations in the **OpenGL** (and **DirectX**) pipelines: they do not support non-linear projections because the graphics card itself is made and optimized for linear operations on vertices and pixels. Although a software rasterization is still an option, such approaches can never lead to a realtime performances. Hence, most approaches use graphics card capabilities and tradeoff accuracy and generality for speed. There exist few types of method for synthesizing non-linear projections for single center of projection models. It can be done at the vertex level by computing the correct projection of the mesh vertices in a vertex shader [31]. Another approach

include rendering a perspective view of the scene in a texture and deforming a grid covering the screen on which this texture is mapped [31, 39]. Purely vertex based approaches are generally inaccurate in case of large distortions as it involves a linear interpolation between the vertices during the rasterization process. Hence, in order to have a good visual result the mesh must be tessellated into small triangles. Pixel based approaches are generally more accurate but slower. One technique consists of rendering the scene in a texture and instead of computing the deformation at the vertex level, post-process the texture in a pixel shader in order to deform it at the pixel level. An other method consists of rendering a cube map and process it to render the non-linear view [38]. The pixel based approaches are less inaccurate than the vertex base approach. Nevertheless, the texture quantization can also introduce artifacts, and hence the accuracy is determined by the texture size. There exist some hybrid approaches involving both vertex and pixel processing [17, 13], where [17] also introduces a more efficient algorithm to compute a projection for multi-center models. In addition of all these works [6] proposed a framework for flexible viewing volume rendering which allows the user to deform the frustum of the camera in more complex shapes and obtain a non-linear projection.

The main idea of the algorithm presented in this work is partly similar to [38], but instead of an analytical model, a mapping table computed using the generic calibration is used. The synthesis is done with the graphic processor unit (GPU) using a shader program implemented in OpenGL shading language (GLSL) in order to generate a large number of fisheye images in few seconds.

1.2.3 Sky Detection

In order to extract the skyline of the real image a sky detection algorithm based on graph cuts has been developed. Classifying the pixels into sky and rest can be seen as a segmentation problem with two labels. The features that can be used for this segmentation can vary from simple RGB colorspace components to a wide variety of features like gradients, straight lines, vanishing points, etc. The labeling can be formulated as an energy minimization problem and

solved by a discrete optimization algorithm like graph cuts. Graph cuts is successfully used in various vision problems like stereo, segmentation, and image restoration [34]. A parameter learning algorithm has been implemented in order to find the best parameters of the energy function to minimize and to avoid hand tuning. Our method is similar to the maximum-margin network learning method using graph cuts [35].

Few related works exist, in [36] a novel method has been proposed to segment and replace the sky in different images. The segmentation is achieved using graph cuts, where the features are learned using a classifier based on randomized forest [24]. Another related work, which already gives good results, is the geometric labeling problem to classify a given image into sky, buildings and ground [16]. However, some of their prior assumptions do not hold true in our model. First, the camera is not perspective. Second, the image is not taken by a person with the optical axis approximately parallel to the ground but perpendicular to the ground. Third, the algorithm expects the sky to be mostly on the top of the image. On the contrary, the images used mainly have the sky at the center.

1.2.4 Skyline Matching

Matching two shapes or measuring the similarity between shapes is a well researched topic in computer vision. Applications of shape matching include image retrieval, object recognition, object detection, pose estimation and registration. Hence, a wide range of techniques have been developed to solve these problems even in case of occlusion or noise as [22, 4, 3, 30]. In order to match the skyline robustly and efficiently a RANSAC based chamfer matching algorithm has been designed. It can find the precise position and orientation in the 3D model in the presence of outliers such as missing buildings or occlusions. The algorithm finds the location with the minimum cost between the real and the synthetic skylines by generated fisheye images for different orientations and locations in the 3D model. For each new generated image the algorithm computes the cost in $O(N)$, where N is the number of pixels of the image.

Overview of this thesis: In Chapter 2 the generic calibration and fisheye synthesis algorithm are presented. Next, the sky detection and matching algorithms are explained in Chapter 3. As this work involved a lot of software engineering and optimization, some details about the implementation are given in Chapter 4. Finally, promising experimental results are shown in Chapter 5.

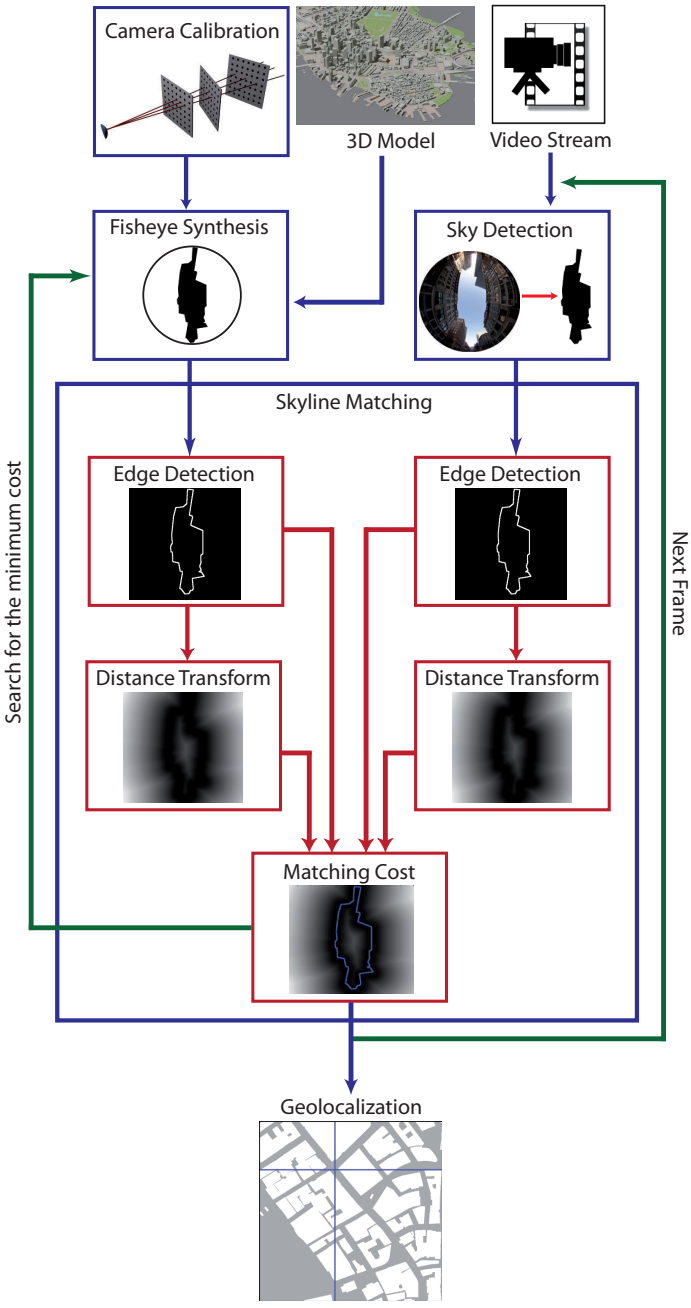


Figure 1.3: *The various stages of the algorithm.*

Chapter 2

Fisheye Lens Geometry and Synthesis

The fisheye camera is an important part of our setup as it allows to acquire images containing a wide skyline for the geolocalization. Nevertheless, the fisheye camera does not belong to the class of linear projection and hence need a different calibration method. In this chapter, firstly a quick overview of the geometry of a fisheye lens is given, secondly the method used to calibrate the camera is explained and finally the algorithm developed to synthesized fisheye images from a 3D model is presented.

2.1 Linear and Non-Linear Projection

A camera projection can be defined as a mapping from a 3D scene point to a 2D pixel $f : \mathbb{R}^3 \rightarrow \mathbb{R}^2$. In computer graphics and vision, linear projections such as perspective and orthographic are the most popular ones. For a 3D point $P(X, Y, Z)$ and the image plane at a distance f from the camera center C, the projection $p(x, y)$ is defined in the perspective case by $x = \frac{fX}{Z}$ $y = \frac{fY}{Z}$ (cf. Figure 2.1) and in the orthographic case by $x = X$ and $y = Y$ (cf. Figure 2.2). They can be formulated in a matrix form using homogeneous coordinates.

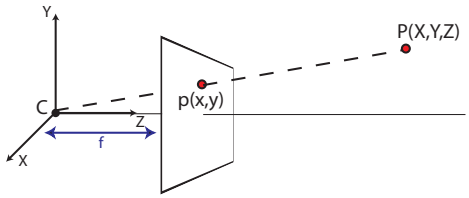


Figure 2.1: *For a 3D point $P(X, Y, Z)$ and the image plane at a distance f from the camera center C, the projection $p(x, y)$ is defined in the perspective case by $x = \frac{fX}{Z}$ $y = \frac{fY}{Z}$.*

$$Perspective: \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.1}$$

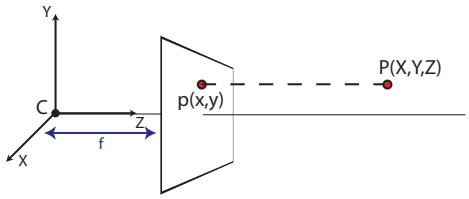


Figure 2.2: *For a 3D point $P(X, Y, Z)$ and the image plane at a distance f from the camera center C, the projection $p(x, y)$ is defined in the orthographic case by $x = X$ and $y = Y$.*

$$Orthographic: \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.2}$$

Numerous camera models are non-linear as fisheye or catadioptric and hence cannot be characterized by a linear system. Many mathematical functions have been derived to model different types of non-linear lenses. In the case of the fisheye lenses, several models exist assuming that the fisheye cameras have a single center of projection (in reality the rays pass through a small disk in space and not a single point). One class of model used for fisheye is called radially symmetric [10]. The radially symmetric models with a single center of projection

are defined by an equation of the form $r = k \cdot f(\theta)$ where k is the lens parameter, θ is the angle made by the projection ray and the optical axis and r is the distance between the 2D image point and the image center (cf. Figure 2.3). These models can be visualized by a surface of revolution as shown in the Figure 2.4 and defined by the equation:

$$z = \frac{r}{\tan\theta} = \frac{f(\theta)}{\tan\theta} \tag{2.3}$$

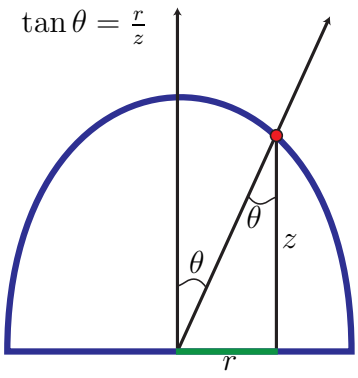


Figure 2.3: *The radially symmetric models with a single center of projection is defined by an equation of the form $r = k \cdot f(\theta)$ where k is the lens parameter, θ is the angle made by the projection ray and the optical axis and r is the distance between the 2D image point and the image center.*

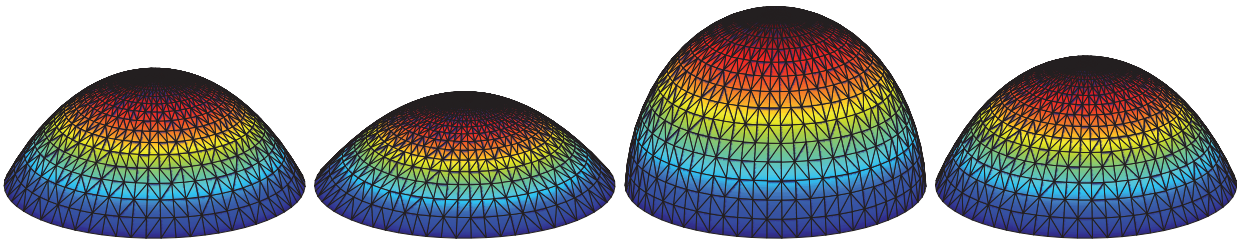


Figure 2.4: *(a) Equidistant projection: $r = k\theta$ (b) Stereographic projection: $r = k\tan\frac{\theta}{2}$ (c) Equisolid angle projection: $r = k\sin\theta$ (d) Sine law projection: $r = k\sin\frac{\theta}{2}$*

Projecting a 3D point in the case of radially symmetric models is equivalent to find the intersection of the projection ray with the surface of revolution and taking the orthographic projection of this point on the image plane as shown in the Figure 2.5.

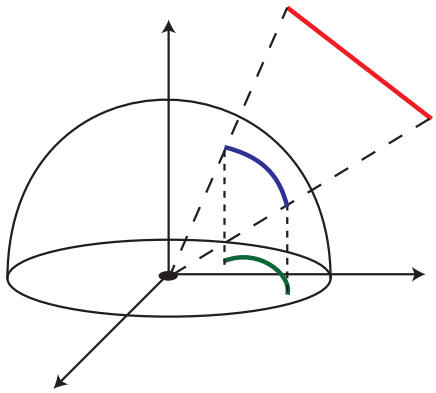
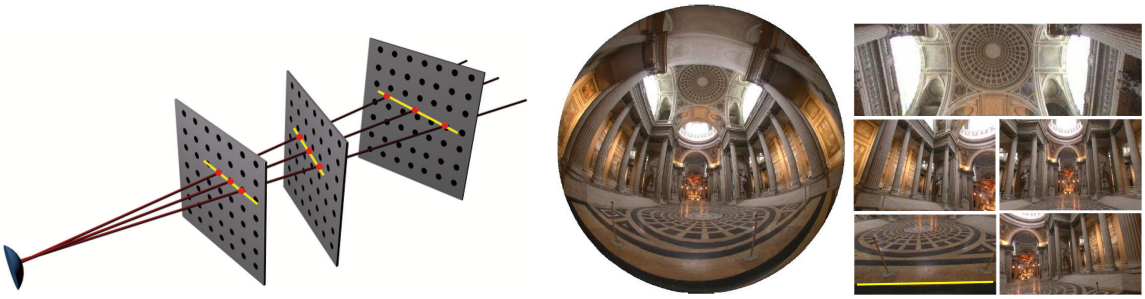


Figure 2.5: *Fisheye Projection of the red line. Firstly the line is projected over the surface of revolution (blue line), and then projected onto the image plane (green line).*

2.2 Calibration



(a) (b) (c)

Figure 2.6: *(a) Generic calibration using three calibration grids. (b) and (c) show an original fisheye image and samples of distortion corrected regions, obtained using the calibration result. The yellow line in (c) shows that the algorithm can correct even heavily distorted lines.*

The main idea behind the generic calibration approach [28] is explained in this section. Three images of a calibration grid are captured by the camera that needs to be calibrated. The images are taken from unknown viewpoints. Every image pixel observes three 3D points in different calibration grids as shown in Figure 2.6(a). These 3D points are obtained in three different coordinate systems. However, these points are collinear if they are expressed in the same coordinate system. That constraint enables us to compute the motion between the views and eventually the projection rays for the image pixels.

Let $\mathbf{Q}$, $\mathbf{Q}'$ and $\mathbf{Q}''$ refer to a known point on the first, second and the third calibration grids

respectively. Let the coordinate system of the first calibration grid be the reference frame and O represent the unknown optical center. The pose of the second and the third calibration grids with respect to the first one are given by $(\mathsf{R}', \mathbf{t}')$ and $(\mathsf{R}'', \mathbf{t}'')$ respectively. In order to estimate the pose of the grids, the three points $\mathbf{O}$, $\mathbf{Q}$ and $\mathbf{Q}'$ are stacked in a matrix after transformation to the reference frame and the collinearity constraint is used to extract the poses: if the estimation of the grid poses and the optical center is correct $\mathbf{O}$, $\mathbf{Q}$ and $\mathbf{Q}'$ are collinear. The collinearity constraint will force the determinant of any submatrix, whose size is size 3×3, of the following matrix to vanish:

$$\begin{pmatrix} O_1 & Q_1 & R'_{11}Q'_1 + R'_{12}Q'_2 + R'_{13}Q'_3 + t'_1Q'_4 \\ O_2 & Q_2 & R'_{21}Q'_1 + R'_{22}Q'_2 + R'_{23}Q'_3 + t'_2Q'_4 \\ O_3 & Q_3 & R'_{31}Q'_1 + R'_{32}Q'_2 + R'_{33}Q'_3 + t'_3Q'_4 \\ O_4 & Q_4 & t'_4Q'_4 \end{pmatrix} \tag{2.4}$$

In other words, we obtain four constraints by removing one row at a time. A similar matrix can be obtained from the 3D points on first and third calibration grids. Using all these constraints the motion variables $(\mathsf{R}', \mathbf{t}')$ and $(\mathsf{R}'', \mathbf{t}'')$ are extracted. After extracting the motion variables we can reconstruct these projections and thus obtain the necessary calibration information. The details of this extraction is shown in [28].

In practical calibration approaches, it is not possible to extract the projection rays for every pixel. The authors of [28] suggested the use of homography or bilinear interpolation for computing the projection ray for other pixels.

2.3 Fisheye Synthesis

As mentioned in the introduction, the methods to generate a non-linear projection can be classified in three different classes: vertex based, pixel based and mixed (processing on vertices and pixels). In order to synthesize an accurate non-linear projection of the 3D model in real time, an image based approach has been chosen. Many reasons have motivated this choice.

Firstly in a pixel based approach the accuracy of the result does not depend on the geometry of the model. Secondly, this technique is very suitable for the incorporation of the generic calibration. Finally it can be efficiently optimized with the new shader model and the parallel architecture of the GPU.

The main idea of pixel based algorithms is to render a raster representation of the scene using multiple perspective views (as a cube map) and to convert them to a non-linear projection. In order to generate this representation the camera is rotated in its local coordinate system and the views are rendered in a texture array

In Figure 2.7 the various stages in synthesizing a fisheye image from a hemicube are shown. The first step of the algorithm consists to calibrate the camera (as explained in the previous section) and to generate the table which maps the pixels in the fisheye image to the corresponding projection rays in space. Then, the mapping between the fisheye image pixels and the hemicube pixels is computed by using the calibration information. To do that the projection rays stored in the generic calibration table are intersected with the faces of the hemicube in order to associate every pixel in the fisheye image to a specific face and sub-pixel location in the hemicube. These associations between pixels from the fisheye image and pixels from the hemicube faces are stored in a floating point texture in order to be compatible with the pixel shader in charge of post-processing the perspective views.

To synthesize a fisheye image five binary perspective images of the 3D model corresponding to the views in the hemicube are generated. These views are synthesized in a single pass thanks to a layered rendering approach using a geometry shader. The binary views of the hemicube are generated simply by rendering a 3D model colored entirely in white, hence the resulting binary images are black for the sky regions and white for the buildings. The hemicube is then transformed to a fisheye image thanks to a pixel shader by a simple look-up in the pre-computed mapping table (floating point texture).

Our ray-calibrated view synthesis has the advantage that it does not “bake in” error that would arise from using a parametric lens model. Moreover, as a result of our optimization

techniques, accurate fisheye images are generated on the fly at a high frame rate removing the need to store images in a large database.

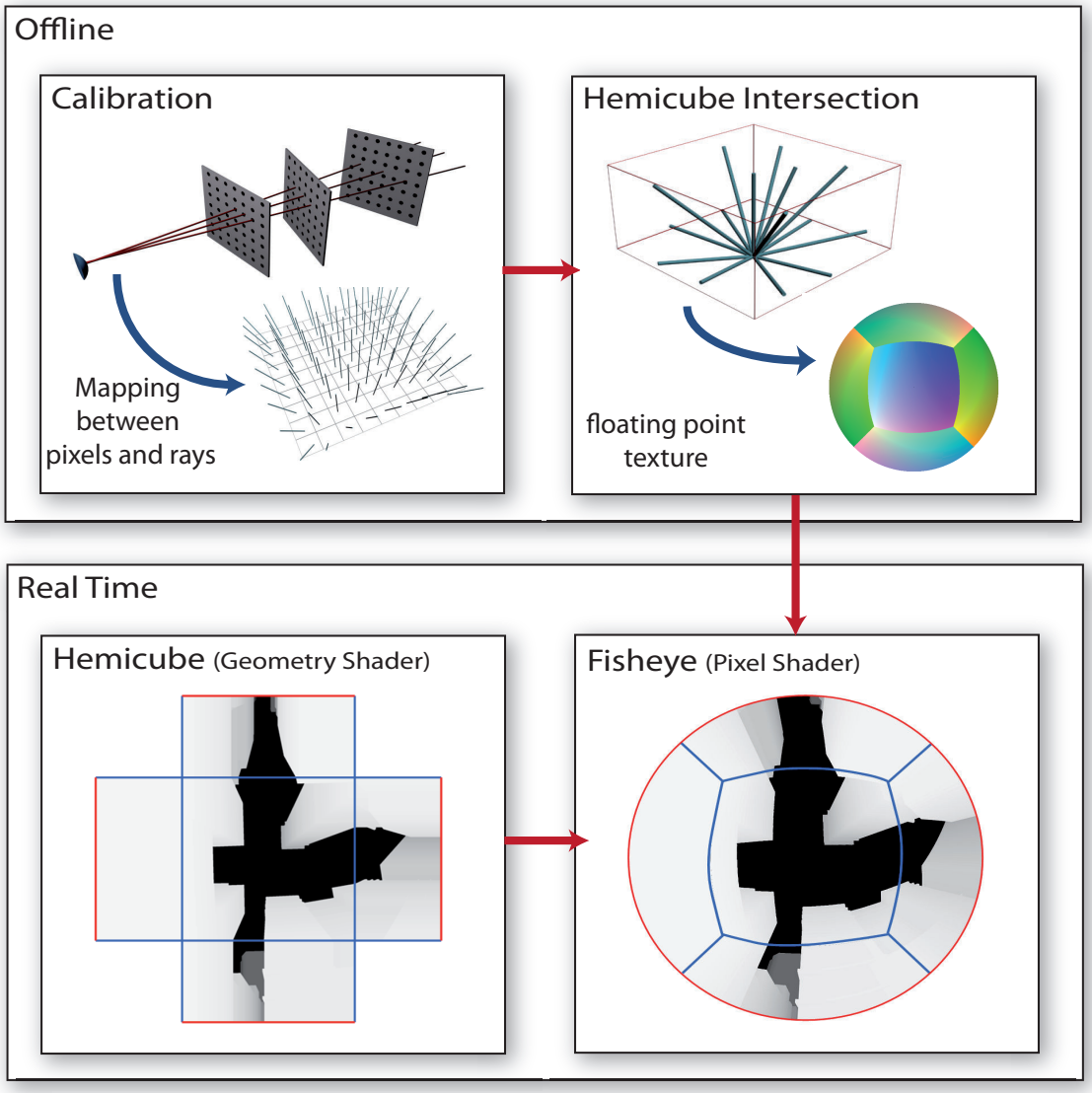


Figure 2.7: *The various stages of the fisheye synthesis algorithm by transformation of a hemicube.*

Chapter 3

Geolocalization by Matching Skyline

This chapter explains how the skyline is extracted from the fisheye images and how the matching between synthetic and real skylines is done. This chapter introduces the parameter learning method used for the sky detection and the RANSAC based matching algorithm.

3.1 Segmentation

The labeling of the fisheye images in sky and rest is a segmentation problem with two labels which can be formulated as an energy minimization problem and solved by a graph cuts algorithm. An energy function, involving binary variables in unary and pairwise terms, is represented using a weighted graph whose minimum cut yields an energy-minimizing partition of the variables. The minimum cut (st-mincut) separates the set of nodes (a node symbolizes a pixels) in the graph into two sets, one belonging to a source node (sky) and one belonging to a sink node (rest). Formally, let $G = (V, E)$ be a directed graph which consists of a set of non-negative directed edges E and of a set of nodes V containing two special nodes, namely, the source S and the sink T. The st-mincut algorithm partitions the set of vertices in V into two disjoint sets V_S and V_T, such that $S \in V_S$ and $T \in V_T$. The special nodes S and T correspond to sky and rest respectively as shown on Figure 3.1. The cost of the cut is computed by the sum

of the weights corresponding to directed edges linking a node $p \in V_S$ to a node $q \in V_T$. It has been proved by Ford and Fulkerson [11] that finding the value of the maximum flow from the source to the sink is equivalent to finding the value of the minimum cut.

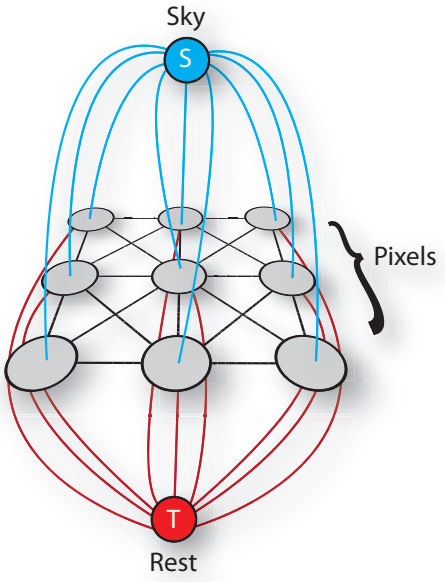


Figure 3.1: *In the graph the source node S represents the sky, the sink node T represents the rest and the other nodes represent the pixels of the image. The minimum cut separates the pixels into two sets, one belonging to the sky S and one belonging to the rest T.*

In our algorithm a quadratic pseudo-boolean function is used to represent the energy function to minimize for labeling the sky and the rest (cf. Equation 3.1). It is a energy function of boolean variables that maps a boolean vector to a real value. Let $x_i \in \mathbb{B} = \{0, 1\}$ where $i = 1, 2, ..., n$ represent boolean random variables. These variables represent the boolean labels (sky and rest) of the pixels in the image. Let θ denote the parameters in the energy function. The parameter vector θ consists of unary terms $\theta_{i;a}$ and pairwise terms $\theta_{ij;ab}$, where $i, j = 1, 2, ..., n$ and $a, b \in \mathbb{B}$. These parameters are also referred to as unary and pairwise potentials. The unary parameter $\theta_{i;a}$ can be seen as a pseudo-boolean function $f : \mathbb{B} \rightarrow \mathbb{R}$ that gives the cost when $x_i = a$. Similarly, a pairwise parameter $\theta_{ij;ab}$ is a pseudo-boolean function $f : \mathbb{B}^2 \rightarrow \mathbb{R}$ that

gives the cost when $x_i = a$ and $x_j = b$. The function mapping partitions to energies is then

$$E(x|\theta) = \sum_{i \in V} [\theta_{i;0}\overline{x_i} + \theta_{i;1}x_i] + \sum_{(i,j) \in E} [\theta_{ij;00}\overline{x_i x_j} + \theta_{ij;01}\overline{x_i}x_j + \theta_{ij;10}x_i\overline{x_j} + \theta_{ij;11}x_i x_j] \quad (3.1)$$

It is important to note that minimizing $E(x|\theta) + C$ (where C is a constant) is similar to minimizing $E(x|\theta)$: the same boolean vector $[x_1, ..., x_n]$ minimizes $E(x|\theta) + C$ and $E(x|\theta)$.

In order to minimize this function a graph is constructed with a node for each boolean random variable $x_i \in \mathbb{B}$. The minimal cut will give the binary labeling of all the vertices of the graph which minimize the function. If the function is submodular the energy can be represented by a graph [21]. A function is called submodular if and only if for each couple (i, j) $(\theta_{ij;00} + \theta_{ij;11}) \leq (\theta_{ij;01} + \theta_{ij;10})$. The submodularity condition is the discrete analogues of convex functions in continuous domains. It can been seen on the Figure 3.3 that if the submodularity condition is not respected the pairwise cost of the graph becomes negative and hence the min-cut problem becomes NP-hard and cannot be solved by a maximum flow algorithm. The construction of the graph for the quadratic energy function is described bellow.

Firstly let us construct a graph for an energy function containing only unary terms:

$$E(x|\theta) = \sum_{i \in V} [\theta_{i;0}\overline{x_i} + \theta_{i;1}x_i] \quad (3.2)$$

The function can be factorized:

$$E(x|\theta) = \begin{cases} \sum_{i \in V} [\theta_{i;0} + (\theta_{i;1} - \theta_{i;0})x_i] & \text{if } (\theta_{i;1} - \theta_{i;0}) \geq 0 \\ \sum_{i \in V} [\theta_{i;1} + (\theta_{i;0} - \theta_{i;1})\overline{x_i}] & \text{if } (\theta_{i;0} - \theta_{i;1}) \geq 0. \end{cases} \quad (3.3)$$

The above energy function is represented in a graph shown in Figure 3.2.

Then let us construct a graph for an energy function containing only pairwise terms:

$$E(x|\theta) = \sum_{(i,j) \in E} [\theta_{ij;00}\overline{x_i x_j} + \theta_{ij;01}\overline{x_i}x_j + \theta_{ij;10}x_i\overline{x_j} + \theta_{ij;11}x_i x_j] \quad (3.4)$$

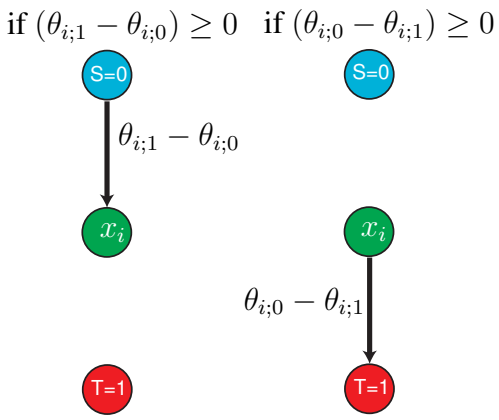


Figure 3.2: *Graph construction in the case of an energy function containing only unary costs.*

The function can be factorized as follows:

$$E(x|\theta) = (\theta_{ij;01}+\theta_{ij;10}-\theta_{ij;00}-\theta_{ij;11})\overline{x_i}x_j+(\theta_{ij;10}-\theta_{ij;00})x_i+(\theta_{ij;10}-\theta_{ij;11})\overline{x_j}+(\theta_{ij;00}+\theta_{ij;11}-\theta_{ij;10}) \tag{3.5}$$

The graph construction for the above pairwise energy function is shown in Figure 3.3.

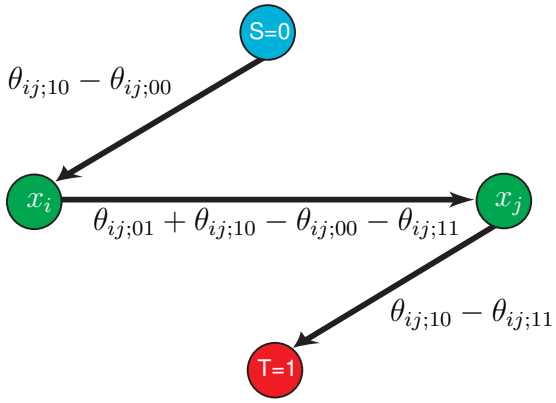


Figure 3.3: *Graph construction in the case of an energy function containing only pairwise costs.*

The energy function is the sum of the unary and pairwise terms, hence the final graph used for minimizing the energy function is constructed by merging the graphs for unary and pairwise terms as per the additivity property shown in [21].

3.2 Parameter Learning

In order to avoid the hand tuning of the parameters of the energy function, the parameters are learned automatically from a hand labeled training set. The discriminating power of the model is optimized by estimating parameters that maximize the difference between ground truth labelings and all other labelings. Near-miss labelings are generated, and a parameter vector that maximizes the margin separating true labelings from the near misses in a convex optimization is estimated. This operation is done repeatedly and the process converges to an optimal parameter vector in a small number of iterations [35]. In the current implementation the unary likelihood for sky and rest are obtained using their color values. A Gaussian model is first estimated for the classes sky and rest and the mean and covariance are computed using manually segmented ground truth images. For a new test image the unary likelihood is obtained by computing the Mahalanobis distance to the sky and non-sky classes as shown in Figures 3.4. The energy

(a) (b) (c)

Figure 3.4: *(a) Original image. (b) Likelihood for the sky. (c) Likelihood for the rest of the image. The brighter values correspond to higher likelihood.*

function is reformulated by decomposing the unary parameters $\theta_{i;a}$ as follows:

$$\theta_{i;a} = \theta_a^p + l_i \theta_a^l \tag{3.6}$$

Once the likelihood cost l_i is computed for every node the unary parameters θ_a^p and θ_a^l are only dependent of the label a. Similarly the pairwise parameters $\theta_{ij;ab}$ are also assumed independent of the associated nodes i and j and are replaced by θ_{ab}. Due to the problem nature, the pairwise matrix is assumed symmetric. i.e. $\theta_{01} = \theta_{10}$. We denote the new parameter vector which we want to learn as Θ.

$$\Theta = \begin{bmatrix} \theta_0^p & \theta_0^l & \theta_1^p & \theta_1^l & \theta_{00} & \theta_{01} & \theta_{11} \end{bmatrix} \tag{3.7}$$

The parameter vector Θ is then estimated via the standard convex program for linear support vector machines (SVMs), using vectors of unary and pairwise statistics from true labelings and near miss labelings as positive and negative examples. In order to guarantee that the estimated model supports optimal inference, the convex program is augmented with a constraint on the pairwise terms in the parameter vector that guarantees submodularity. As mentioned before, in the binary case the constraint is simply

$$\theta_{00} + \theta_{11} \leq 2\theta_{01} \tag{3.8}$$

θ_0^p	1
θ_0^l	-1
θ_1^p	-0.03708
θ_1^l	-0.0935
θ_{00}	-0.5638
θ_{01}	0.2487
θ_{11}	0.3151
Accuracy (%)	94.2

Table 3.1: *The learned parameters of the energy function. Note that the pairwise parameters satisfy the submodularity condition, as imposed in our learning framework. The accuracy figure in the last row is the percentage of correctly classified sky pixels in the training set.*

3.3 Matching and Distance Transform

In order to match the real skyline with the synthetic one, a RANSAC based chamfer matching has been proposed. The chamfer distance is generally used to measure similarity between two shapes and can be used to find the location of a template shape in an edge map. The idea of the chamfer matching is to minimize the mean distance between the edge pixels of a template shape and their closest correspondences on the edge map. Formally, let T be a template image, E an edge map and p_T and p_E the location of an edge pixel respectively in T and E. The chamfer matching consists of finding the position that minimizes the chamfer distance ($\mathcal{C}_d(T, E)$):

$$\mathcal{C}_d(T, E) = \frac{1}{|T|} \sum_{p_T \in T} \min_{p_E \in E} \|p_T - p_E\|_2 \tag{3.9}$$

The chamfer distance can be efficiently computed with the distance transform map of the edge map image. The distance transform ($\mathcal{D}_t$) stores for each pixels p the distance to the closest edge pixel p_E and can be computed in linear time [9].

$$\mathcal{D}_t(p) = \min_{p_E \in E} \|p - p_E\|_2 \tag{3.10}$$

Hence the expression for chamfer distance shown in Equation 3.9 can be written using the distance transform as:

$$\mathcal{C}_d(T, E) = \frac{1}{|T|} \sum_{p_T \in T} \mathcal{D}_t(p_T) \tag{3.11}$$

The chamfer matching algorithm already gives good geolocalization when the sky detection and the 3D model are reasonably accurate. However the algorithm fails in the presence of occlusions from trees, sun, etc. In order to handle these problems, we propose a robustified version of the chamfer matching using RANSAC. With the conventional chamfer distance, the outliers contribute to the estimation of the location and the orientation. As a result, depending

on the number and nature of these outliers the correct solution can drift as shown in Figure 3.5(c) and Figure 3.6(a).

Our goal is to identify and remove these outliers in the matching algorithm. In order to do this, a RANSAC based chamfer distance has been developed. Our algorithm finds the position that minimize the number of pixels that have their distance to their closest correspondences above a threshold τ. Hence, it consists of finding the position that minimize the following cost function:

$$\mathcal{C}_{rd}(T, E) = \frac{1}{|T|} \sum_{p_T \in T} f(\mathcal{D}_t(p_T)) \tag{3.12}$$

where

$$f(x) = \begin{cases} 1 & \text{if } x > \tau \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

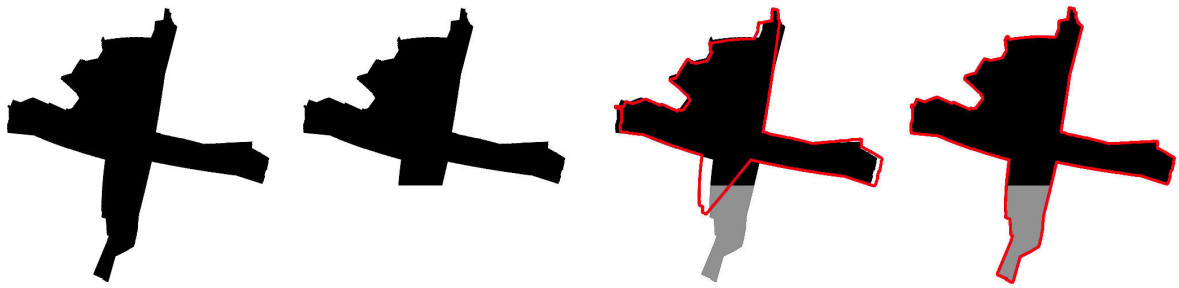
It is also possible to consider the orientation of the edge pixels in deciding whether a pixel is an inlier or not. First, we identify the pixels closest to the template pixels considering only the distance and refer to them as their correspondences. For each pixel p, the location of the corresponding edge pixel p_E is stored in an argument distance transform ($\mathcal{A}_{dt}$):

$$\mathcal{A}_{dt}(p) = \arg\min_{p_E \in E} \|p - p_E\|_2 \tag{3.14}$$

A template pixel is considered to be an outlier if the distance to its corresponding pixel is greater than τ and the difference of orientation with its correspondence above η. Let us represent the normalized gradient information of an edge pixel p as $\overrightarrow{g}(p)$. The final RANSAC cost function embedded in our algorithm is given below:

$$\mathcal{C}_{rd\theta} = \frac{1}{|T|} \sum_{p_T \in T} h(\mathcal{D}_t(p_T), |\overrightarrow{g}(p_T) \cdot \overrightarrow{g}(\mathcal{A}_{dt}(p_T))|) \tag{3.15}$$

where



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

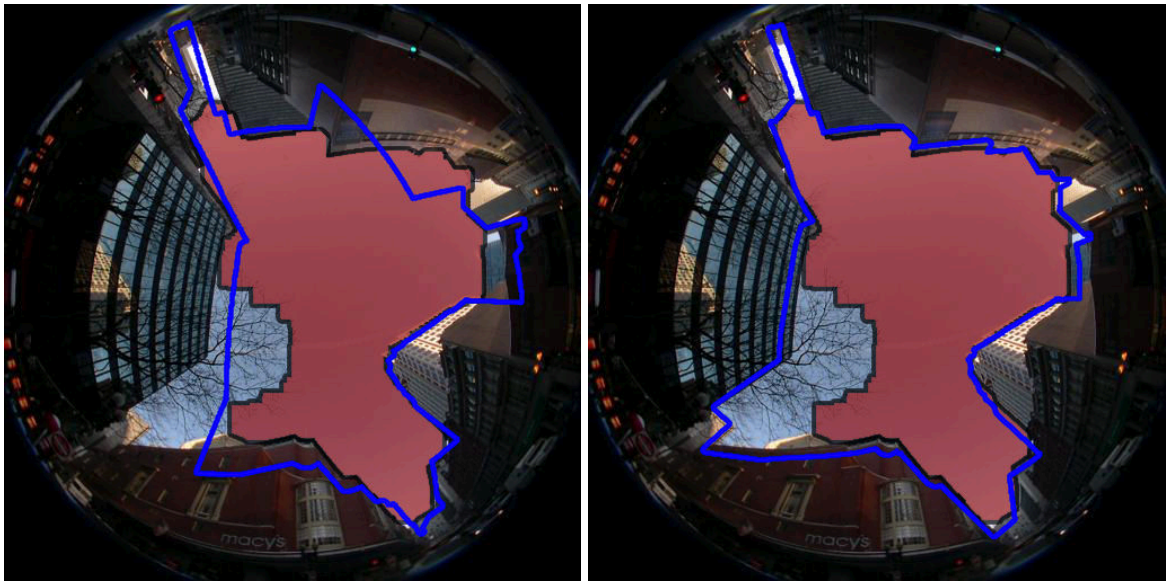
Figure 3.5: *We show the comparison of skyline matching using conventional chamfer distance and our robustified approach. The original skyline before perturbation is shown in (a). A significant part of this skyline is removed as shown in (b). The matching results for the perturbed skyline using conventional chamfer matching and our algorithm are shown in (c) and (d) respectively. Note that there is a big drift from the correct location in (c). On the other hand, our algorithm gets the correct location without any drift as shown in (d).*

$$h(x,y) = \begin{cases} 1 & \text{if } x > \tau \text{ or } y > \eta \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

In our implementation a symmetric cost function ($\mathcal{C}_{rd\theta}(T,E)$+$\mathcal{C}_{rd\theta}(E,T)$) is used where T and E refer to the synthetic and real skylines respectively.

The improvement between the RANSAC and the chamfer distance can be interpreted in terms of l_0 and l_1 norm minimization. The cost function of the conventional chamfer matching as shown in Equation 3.11 is an l_1 norm minimization of the closest distances. On the other hand, the cost function in Equation 3.12 is an l_0 norm minimization of the closest distances. The l_0 norm minimization is known to be more robust to outliers than l_1 norm minimization because the outliers are avoided in the cost function. As a l_0 norm minimization is used, we are very restrictive about our inlier selection. Hence, both distances and orientations are used in choosing the inliers rather than one of them.

The robustness of our proposed matching algorithm is shown in a simple simulation (cf. Figure 3.5) where a correct skyline is manually perturbed and matched using both the conventional chamfer matching and our algorithm. In the case of the chamfer matching, there is a drift in the location and orientation due to the outliers. On the other hand, our approach gives the



(a) (b)

Figure 3.6: *We show the matching results for a real image with slightly incorrect sky detection (red zone) due to an occluding tree. In (a) we show the incorrect matching using a conventional chamfer matching. In (b) we show a relatively better localization using our RANSAC based algorithm.*

correct solution in spite of the perturbation. The robustness is also shown in a real experiments using fisheye images. In Figure 3.6, the sky detection is inaccurate due to the presence of a tree and contrary to the chamfer distance the RANSAC approach is able to get the correct localization in spite of this problem. A detailed analysis of our experimental results is given in Chapter 5, and some sky detection and matching results are shown in Figure 5.6 for Boston and in Figure 5.7 for New York.

Our RANSAC based cost can be efficiently computed in linear time. Moreover in our software, we have implemented a Sobel edge detection with non maxima suppression in GLSL and the distance transform with CUDA and OpenMP. Hence, the matching process is very fast, and a wide number of synthetic fisheye images can be generated and matched in few seconds.

Chapter 4

Rendering of 3D Cities

This chapter presents the software engineering of our geolocalization system. An object-oriented methodology is used to develop the system. All the components are designed as classes and methods for re-usability. The program is developed in C++ and rely on different API: OpenGL (Open Graphics Library) and the shader programming language GLSL (OpenGL Shading Language) are used for graphics rendering and image processing, OpenMP (Open Multi-Processing) and CUDA (Compute Unified Device Architecture) are used to parallelize functions which cannot be implemented efficiently in GLSL. The algorithms have been parallelized between the multi cores of the CPU and the GPU. The code has been cleverly engineered to give the best performances for a wide range of architectures and data size. The discussion in this chapter will mainly focus on the rendering engine.

4.1 Mesh Rendering

In this section the techniques used to pre-process the 3D models and to render them is explained. The pre-processing step is mandatory for our 3D engine because the models are downloaded from internet. These models are usually available in several 3D formats and their scales, positions and orientations in the world coordinate system are unknown and can be widely

different from one to another model.

4.1.1 Coordinate transformation of the 3D model:

The downloaded 3D models are transformed in order that the ground plane of the 3D model corresponds to the world coordinate system XY plane and that the vertical orientation of the buildings corresponds to the Z axis (cf. Figure 4.1). In order to achieve this, the 3D model is rotated and translated. First, a principal component analysis (PCA) is performed on the vertices of the 3D model. The three principal components correspond to the ground plane and vertical orientation of the buildings. Using these principal components the vertical orientation is aligned with the Z axis of the world coordinate system. By doing this we already bring the ground plane of the 3D model parallel to the XY plane of the world coordinate system. Next, the ground plane of the 3D model is centered on the Z axis and translated in order to have the minimum Z value of the model equal to 0. Finally, as the downloaded models are not very precise, the heights of various buildings (expressed in meters) are used to compute the scale factor. Using this scale factor, the entire 3D model is scaled such that 1 meter in the real world corresponds to 1 unit in the 3D model.

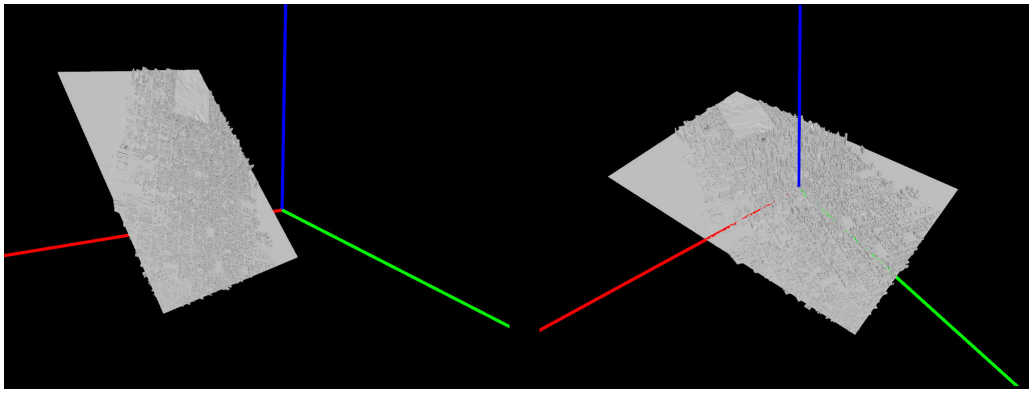


Figure 4.1: *The mesh is automatically translated and rotated in order to be aligned with the XY plane of the world coordinate system.*

4.1.2 Frustum and Back-face Culling:

To render the mesh containing billions of triangles efficiently, the 3D model is directly loaded in the graphic card memory by using a vertex buffer object (VBO) which avoid the repetitive transfer of data between the RAM and the VRAM. Second, the model is batched to reduce the number of API calls for the drawing. Finally, two techniques have been implemented in a geometry shader in order to speedup the frame rate: the frustum culling and the back-face culling. The frustum culling allows to render only the triangles inside the frustum of the camera and the back-face culling allows to render only the triangles facing the camera. With these two techniques, a large amount of triangles are not send to the rasterizer stage saving a lot of operations. Due to the recent development of the geometry shader both frustum and back-face culling can be implemented efficiently. Most of the time our geolocalization system synthesizes fisheye images under various poses. In this operation few triangles from the buildings are inside the frustum. In addition, because of the buildings topology only few triangles face the camera. As a result, less than 20% of the triangles are sent to the rasterization stage. A schematic diagram demonstrating the advantages of frustum and back-face culling is shown in Figure 4.2.

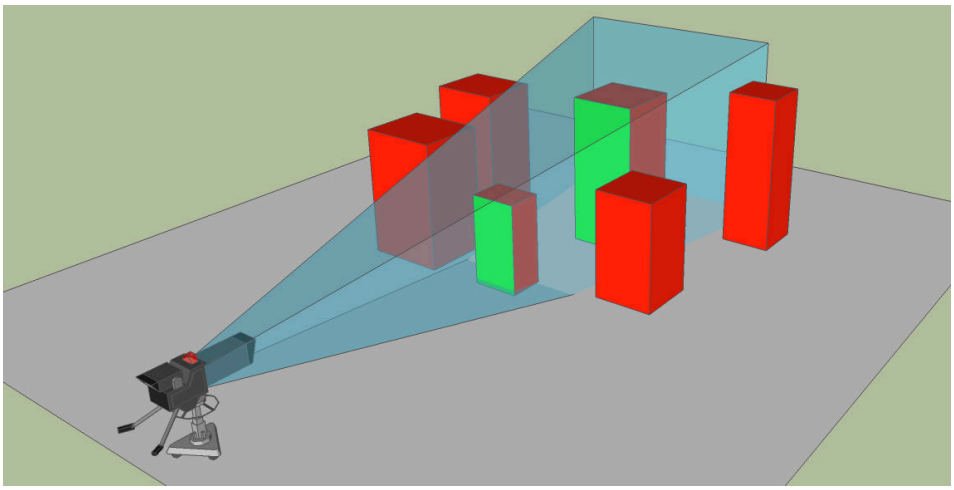


Figure 4.2: *The frustum and the back-face culling remove for each of the perspective views a large number of faces (in red) and keep only few faces (in green).*

4.2 Graphical User Interface:

The interface of the software has been designed for best visualization and algorithmic analysis as shown in Figure 4.3. The interface provides the following functionalities.

- The user can freely navigate in the 3D model.
- For each position, the matching result between the real and the synthetic skylines can be visualized.
- The position of the user is displayed in an aerial view of the city.

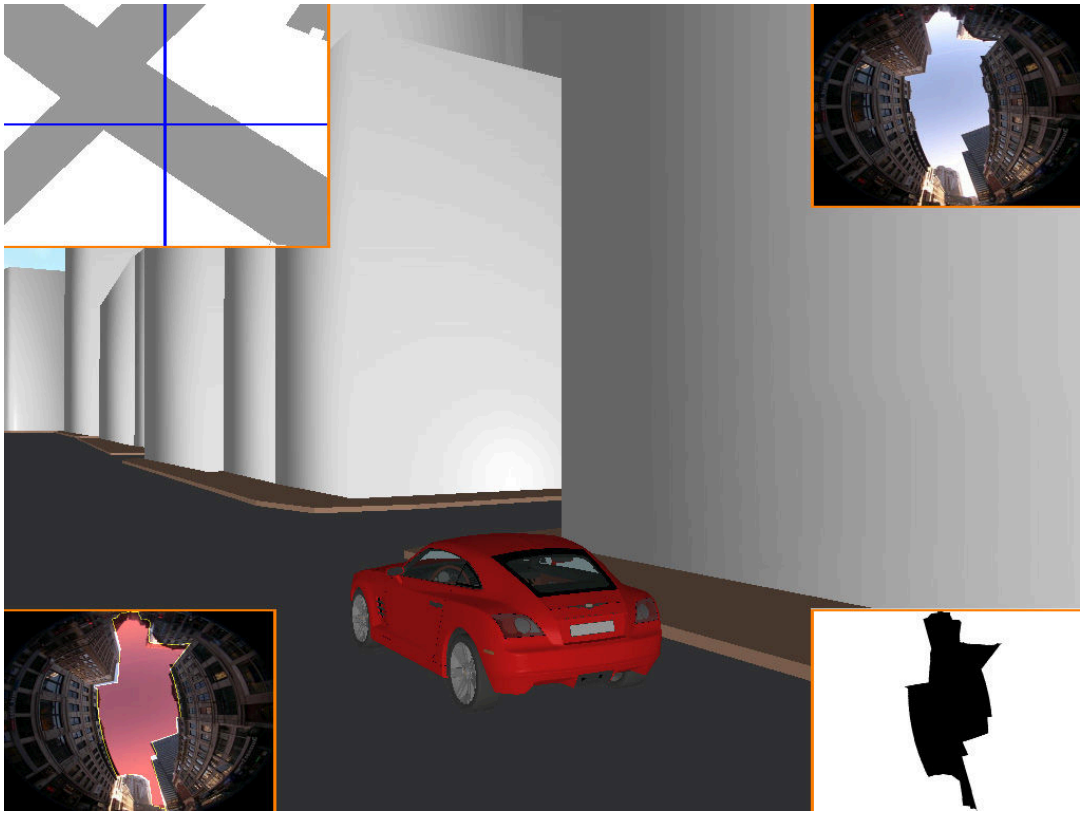


Figure 4.3: *On the top left, the location of the car is displayed (using a blue cross) on an orthographic aerial image of the city where the user can move and zoom in/out. In the aerial map, the road (gray) and the buildings (white) are classified using the Z values. More details are given in the Figure 4.4. On the top right, the real fisheye image is shown. The synthesized fisheye image and the skyline matching are shown on the bottom right and left respectively. The user can freely navigate the 3D model displayed in the middle and simultaneously observe the associated images in all the four corners.*

(a) (b) (c)

Figure 4.4: *The various steps in creating an aerial image displaying buildings and roads from the 3D model of a city (a) Orthographic depth map (b) Detection of buildings using the Z values (c) Localization of the car on the aerial image.*

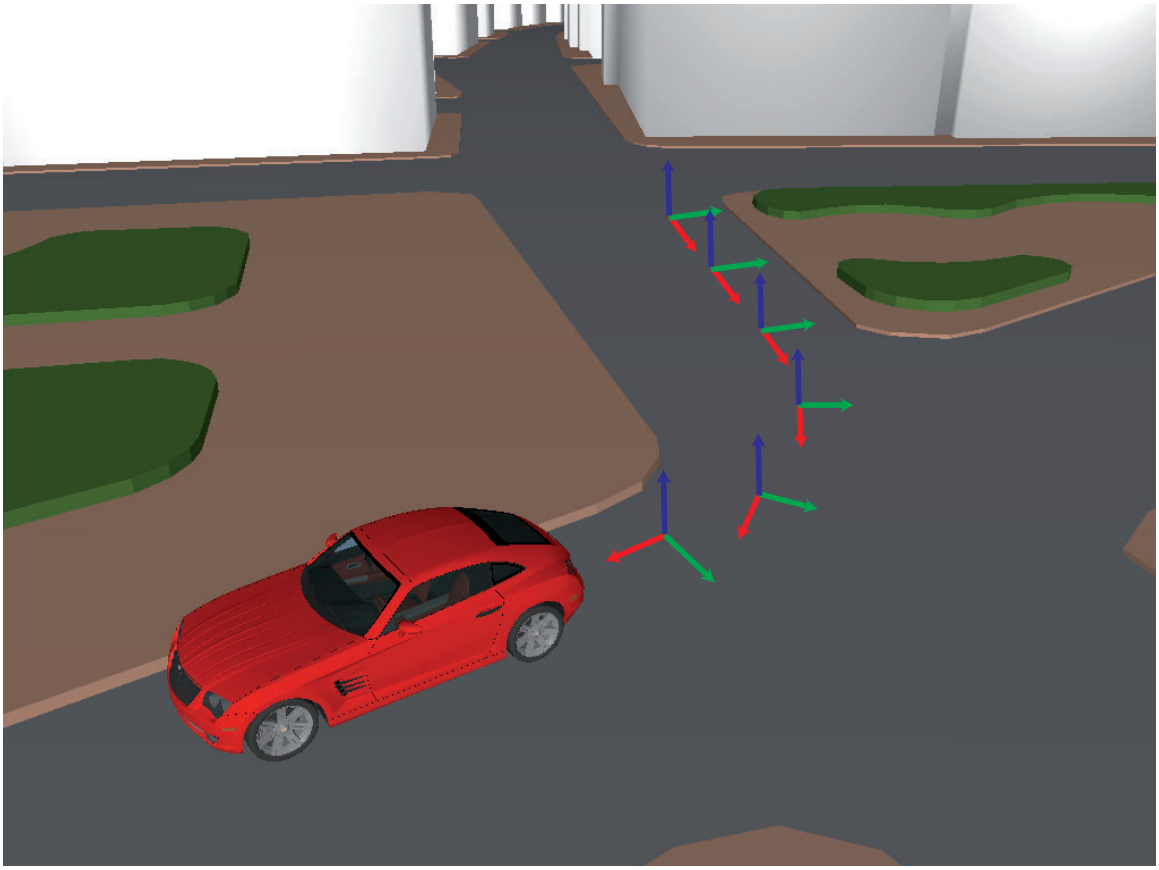


Figure 4.5: *The matching locations and orientations founded by the algorithm are display in the 3D model.*

Chapter 5

Experimental Results and Discussion

In this section we discuss the various simulations and real experiments conducted on the 3D models of Boston Financial district, Boston's North End and New York in order to validate our algorithm.

5.1 Dataset

The real fisheye images are captured using a Nikon Coolpix E8 fisheye lens in Boston and New York. The Boston 3D models used for the experimentation is downloaded for free from:

www.bostonredevelopmentauthority.org/BRA_3D_Models/Index.html

And the New York 3D models has been purchased from the following website:

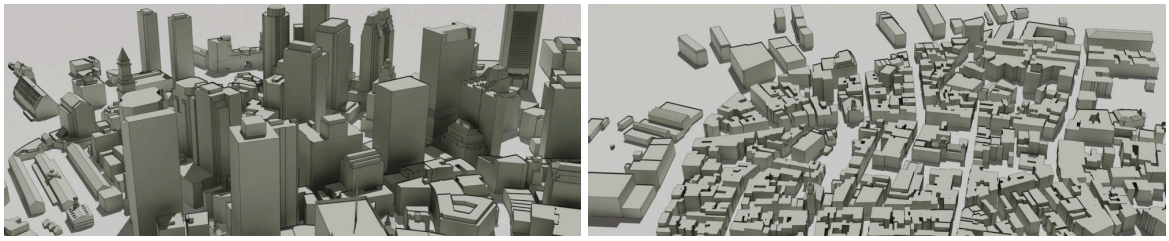
www.3dcadbrowser.com

On this website several other 3D models of cities (Boston, Las Vegas, San Francisco, etc) can also be purchased.

5.2 Simulations

Uniqueness of the skylines: In order to investigate how unique the skylines fingerprint is, 16 random locations are selected on the 3D model of the Financial District of Boston where tall

building can be founded (cf. Figure 5.1(a)). For each of these locations 27 synthetic fisheye images are generated by translations along X, Y and Z directions, separated each by 25cm. For the total of 432 images, a confusion matrix is obtained by computing the chamfer distance for all possible pairs (cf. Figure 5.3(a)). For every point, the first 27 closest matches correspond directly to its nearby locations. The same experiment is conducted in Boston's North End where the buildings are short (cf. Figure 5.1(b)). The experiments demonstrate that the skyline matching algorithm can be robust even if the buildings are short. Surprisingly, the confusion matrix of the North End locations (cf. Figure 5.3(b)) indicates even better location fingerprinting than the confusion matrix of the Financial District.



(a) (b)

Figure 5.1: *The 3D models of (a) Boston's Financial District which has tall buildings and of (b) Boston's North End which has smaller buildings.*

To study more precisely how a skyline varies for small translations and rotations from the correct location, 100 images has been generated for translations on the XY, YZ and XZ plane and rotations around X and Y axis, Y and Z axis and X and Z axis (cf. Figure 5.2). The translations vary from -10 to 10 units where one unit corresponds to 25cm and the rotations vary from -10° to 10°. The result shows that the correct location has the lowest chamfer score, and that the skyline is more perturbed with the rotation than with the translation.

Advantage of Omni-directional camera: In order to assess the benefit of omni-directional cameras, the same experiment of computing the confusion matrix is conducted with a perspective camera in the Financial District. A perspective camera model with a field of view of 90° is used and the images are synthesized at exactly the same locations in the financial district

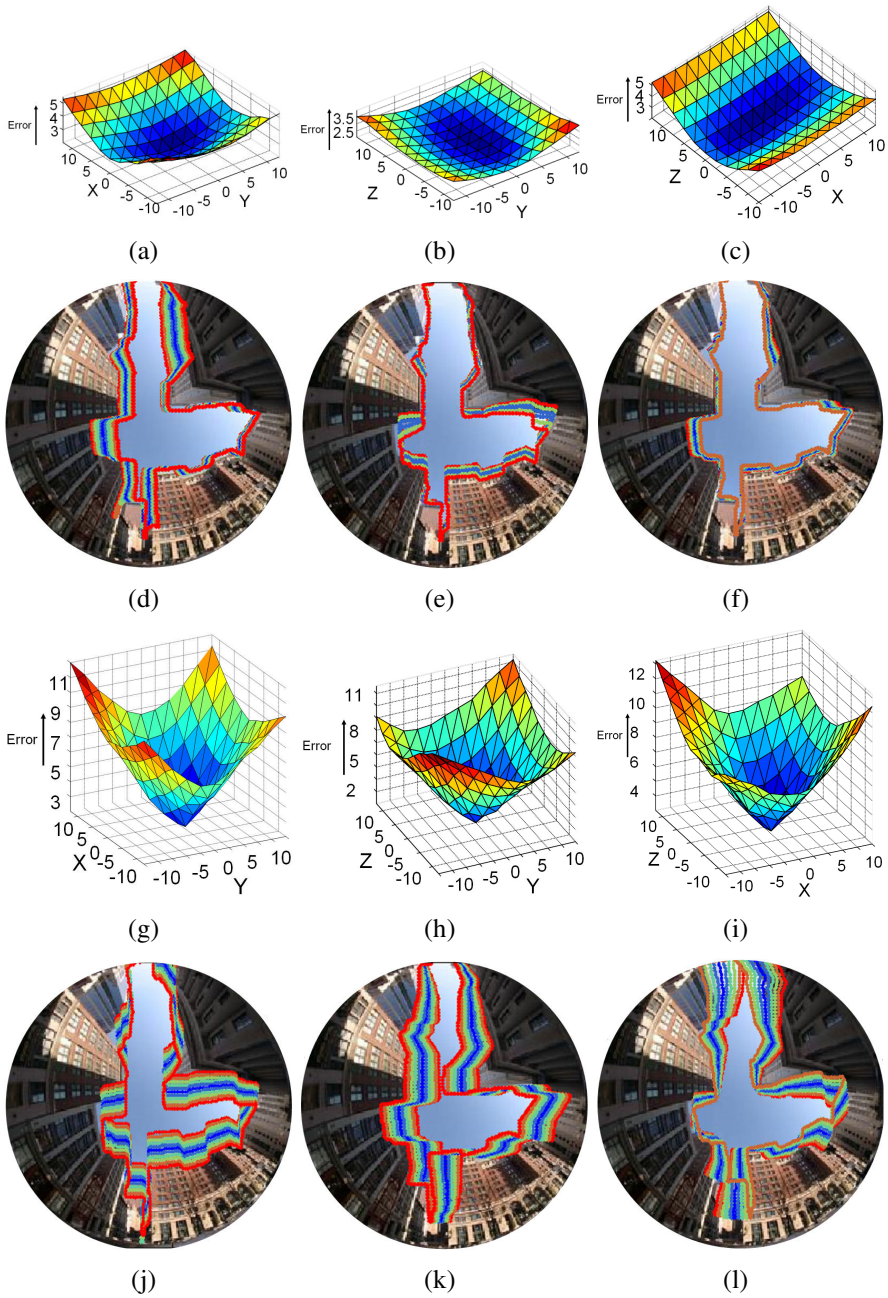
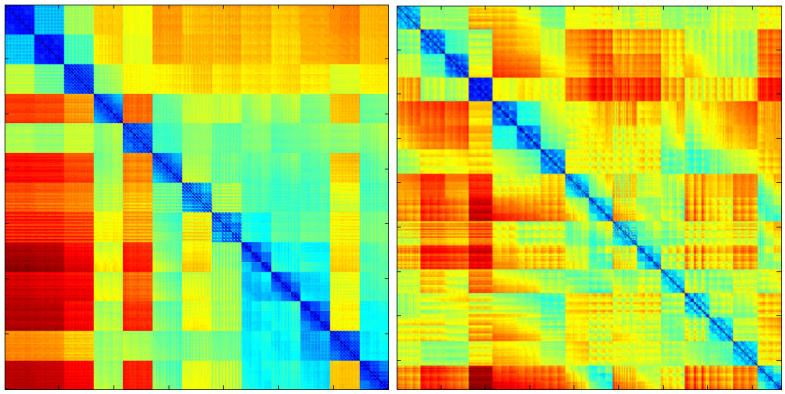


Figure 5.2: *Chamfer distance error (average pixel distance) between the skylines from real and synthetic images for translation (a-c) and rotation (g-i) from the correct location. The translations vary from -10 to 10 units where one unit corresponds to 25cm and the rotations vary from -10° to 10°. The change in shape of the skylines is also studied for small perturbations in translation (X in (d),Y in (e), Z in (f)) and rotation parameters (R_x in (j), R_y in (k) and R_z in (l)). A color-coding is used to display the change in shape of the skylines. The skylines drawn in blue, green and red show the initial, intermediate and final shapes.*



(a) (b)

Figure 5.3: *Confusion matrices computed using 432 fisheye images synthesized at various locations in Boston's financial district (a) and in Boston's north end (b).*

that were chosen for the fisheye experiment. The confusion matrix (cf. Figure 5.4) indicates very clearly the large number of ambiguities between different skylines. The primary difference comes from the fact that the skylines imaged on the fisheye model covers a much longer stretch of the street and thus of the skyline, when compared to the perspective case.

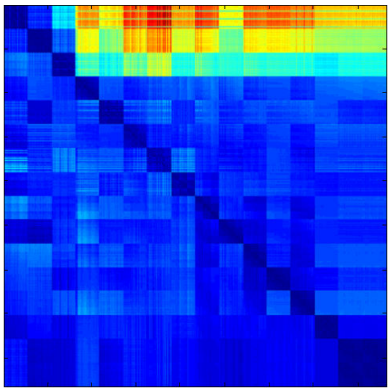
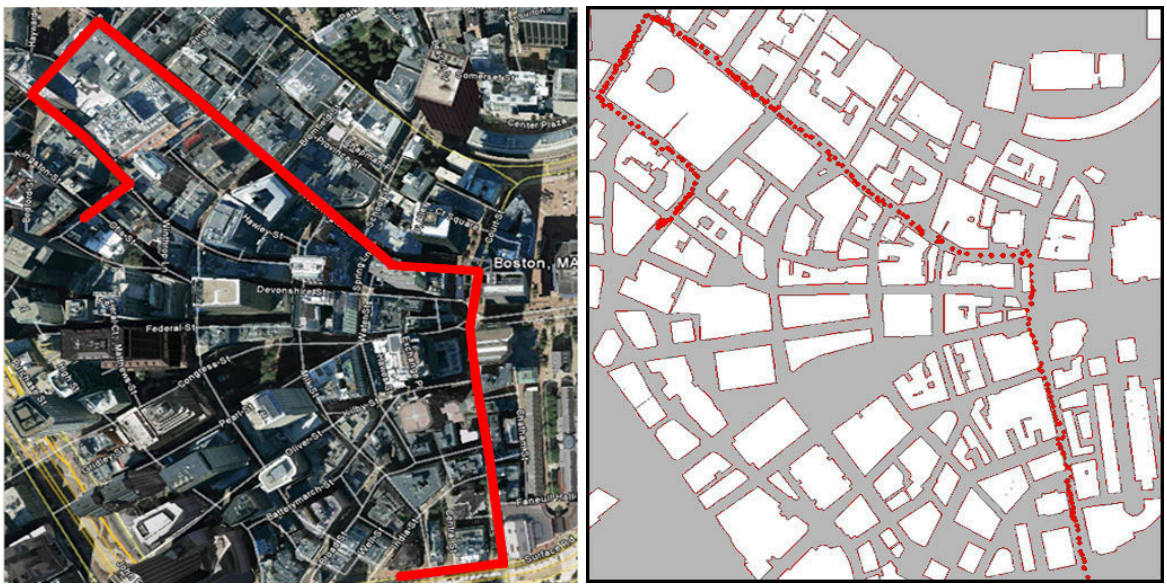


Figure 5.4: *Confusion matrix computed for synthesized perspective views with a field of view of 90° in the financial district of Boston at exactly the same locations where the one shown in Figure 5.3(a)*

5.3 Real Experiments

The real experiments were conducted by capturing 250 fisheye images for approximately 2 kilometers in the Financial district of Boston as shown in Figure 5.5, and 1 kilometer in

New York city. Some of the sky detection and matching results for these cities are shown in Figure 5.6 and Figure 5.7.



(a) (b)

Figure 5.5: *Result of the matching algorithm in Boston Downtown, the experiment was conduct over 2km where 250 images have been taken.*

The accuracy of the algorithm is evaluated by comparing it with a commercial GPS unit "Garmin Nüvi 255W". Upward facing fisheye images and GPS measurements were collected for 30 locations in a street of the Financial District. The average height of the buildings in this street is around 50 meters and the average width of the street is 15 meters (cf. Figure 5.8). The images were collected at the boundary between the sidewalk and the road. We register the real trajectory (boundary between road and sidewalk), GPS measurements and our localization results on an aerial image obtained from Google Earth. In order to compare the result of our algorithm the aerial map of the 3D model is registered over Google Earth by manually clicking corresponding points and by computing the transformation matrix with a least square solution (cf. Figure 5.9). The mean error for the GPS is 29 meters and the mean error for our algorithm is 2.8 meters (cf. Figure 5.10). One of the reasons for the degraded performance of GPS is due to poor satellite reception (because of tall buildings). By comparing the heights of 20 buildings in our 3D model with the ones in Google Earch a discrepancy up to 12 meters is measured.

Figure 5.6: *Some results of the matching algorithm for the Boston data set. The rows show in order the original, the sky detected and the skyline matched images (ordered from left to right).*

Figure 5.7: *Some results of the matching algorithm for the New York data set. The rows show in order the original, the sky detected and the skyline matched images (ordered from left to right).*

Our algorithm was able to achieve good results even with such imprecise 3D models because of the usage of RANSAC in our framework. Thus the precision of our method can even improve further with more accurate 3D models.

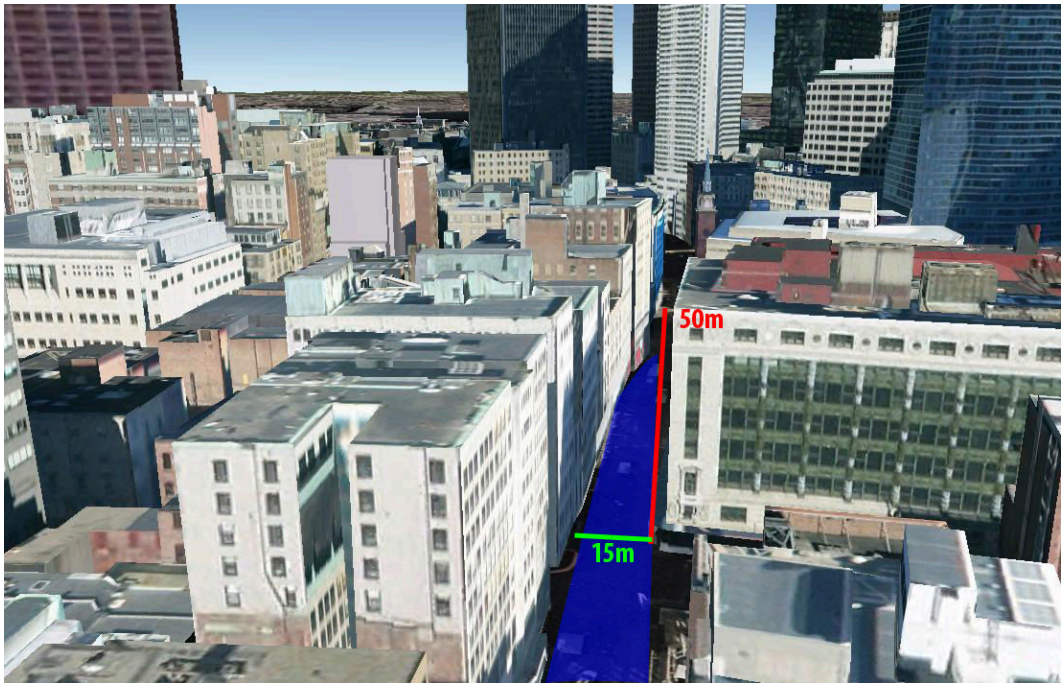


Figure 5.8: *Google Earth view of the street used for the experimentation of the GPS accuracy.*

(a) (b) (c)

Figure 5.9: *Result of the registration (c) of the binary orthographic aerial view of the 3D model (a) over the Google Earth map (b).*

Our results clearly demonstrate that it is possible to outperform GPS measurements in urban scenes, which are known to be extremely problematic for commercial GPS units. Moreover, contrary to the GPS, our algorithm also provides the orientation (cf. Figure 4.5). In the

Figure 5.10: *We compare the accuracy of commercial a GPS units and our fisheye-based system in the Financial District (tall buildings) using 31 sample points. The blue line shows the ground truth (although not completely accurate as it is manually marked on the boundary between the road and sidewalk), and the red and green lines mark the piece-wise linear curves of 31 points obtained from GPS measurements and skyline-based geolocalization respectively.*

future we expect to improve speed and robustness of our algorithm by using other vision algorithms: interest point matching/tracking between consecutive frames, structure from motion algorithms, Kalman filtering, and other priors based on street map information. It is important to note that the current GPS systems have been improved and robustified for several years by addressing various complicated issues that even includes theory of relativity. In a few years, we strongly believe that it is possible to build an inexpensive, robust and accurate vision based GPS.

List of Figures

List of Tables

Bibliography

[1] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *International Conference on Computer Vision (ICCV)*, 2009.

[2] 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 *3D Data Processing Visualization and Transmission (3DPVT)*, 2006.

[3] H.G. Barrow, J.M. Tenenbaum, R.C. Bolles, and H.C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 1977.

[4] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence (PAMI)*, 2002.

[5] T. Brodsky, C. Fermüller, and Y. Aloimonos. Directions of motion fields are hardly ever ambiquous. In *European Conference on Computer (ECCV)*, 1996.

[6] John Brosz, Faramarz F. Samavati, M. T. Carpendale Sheelagh, and Mario Costa Sousa. Single camera flexible projection. In *Non-Photorealistic Animation and Rendering (NPAR)*, 2007.

[7] N. Cornelis, K. Cornelis, and L. Van Gool. Fast compact city modeling for navigation pre-visualization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[8] A.K. Dunne, J. Mallon, and P.F. Whelan. A comparison of new generic camera calibration with the standard parametric approach. *Machine Vision and Applications (MVA)*, 2007.

[9] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance transforms of sampled functions. *Technical Report, Cornell University*, 2004.

[10] M.M. Fleck. Perspective projection: The wrong imaging model. *Technical Report, University of Iowa*, 1995.

[11] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics (CJM)*, 1956.

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

[13] Jean-Dominique Gascuel, Nicolas Holzschuch, Gabriel Fournier, and Bernard Péroche. Fast non-linear projections using graphics hardware. In *Interactive 3D graphics and games (I3D)*, 2008.

[14] M.D. Grossberg and S.K. Nayar. The raxel imaging model and ray-based calibration. *International Journal of Computer Vision (IJCV)*, 2005.

[15] J. Hays and A.A. Efros. Im2gps: estimating geographic images from single images. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[16] D. Hoiem, A. A. Efros, and M. Hebert. Recovering surface layout from an image. *International Journal of Computer Vision (IJCV)*, 2007.

[17] Xianyou Hou, Li-Yi Wei, Heung-Yeung Shum, and Baining Guo. Real-time multi-perspective rendering on graphics hardware. In *Special Interest Group on Graphics and Interactive Techniques (SIGGRAPH)*, 2006.

[18] N. Jacobs, S. Satkin, N. Roman, R. Speyer, and R. Pless. Geolocating static cameras. In *International Conference on Computer Vision (ICCV)*, 2007.

[19] K.Daniilidis and H.H.Nagel. The coupling of rotation and translation in motion estimation of planar surfaces. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 1993.

[20] Oliver Koch and Seth Teller. Wide-area egomotion estimation from known 3d structure. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[21] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *(Pattern Analysis and Machine Intelligence (PAMI))*, 2004.

[22] Y. Lamdan and H.J. Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In *International Conference on Computer Vision (ICCV)*, 1988.

[23] J. Meguro, T. Murata, H. Nishimura, y. Amano, T. Hasizume, and J. Takiguchi. Development of positioning technique using omni-directional ir camera and aerial survey data. In *International Conference on Advanced Intelligent Mechatronics (AIM)*, 2007.

[24] Frank Moosmann, Bill Triggs, and Frédéric Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Neural Information Processing Systems Conference(NIPS)*, 2006.

[25] J. Neumann, C. Fermüller, and Y. Aloimonos. Polydioptric camera design and 3d motion estimation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[26] T. Pajdla. Stereo with oblique cameras. *International Journal of Computer Vision (IJCV)*, 2002.

[27] R. Pless. Using many cameras as one. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[28] S. Ramalingam, P. Sturm, and S.K. Lodha. Towards complete generic camera calibration. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[29] D. Robertson and R. Cipolla. An image-based system for urban navigation. In *British Machine Vision Conference (BMVC)*, 2004.

[30] Jamie Shotton, Andrew Blake, and Roberto Cipolla. Multiscale categorical object recognition using contour fragments. *Pattern Analysis and Machine Intelligence (PAMI)*, 2008.

[31] Martin Spindler, Marco Bubke, Tobias Germer, and Thomas Strothotte. Camera textures. In *International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE)*, 2006.

[32] F. Stein and G. Medioni. Map-based localization using the panoramic horizon. In *International Conference on Robotics and Automation (ICRA)*, 1995.

[33] R. Swaminathan, M.D. Grossberg, and S.K. Nayar. A perspective on distortions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[34] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *Pattern Analysis and Machine Intelligence (PAMI)*, 2008.

[35] Martin Szummer, Pushmeet Kohli, and Derek Hoiem. Learning crfs using graph cuts. In *European Conference on Computer Vision (ECCV)*, 2008.

[36] Litian Tao, Lu Yuan, and Jian Sun. Skyfinder: Attribute-based sky image search. In *Special Interest Group on Graphics and Interactive Techniques (SIGGRAPH)*, 2009.

[37] S. Teller, M. Antone, Z. Bodnar, M. Bosse, S. Coorg, M. Jethwa, and N. Master. Calibrated, registered images of an extended urban area. *International Journal of Computer Vision (IJCV)*, 2003.

[38] Matthias Trapp and Jürgen Döllner. A generalization approach for 3d viewing deformations of single-center projections. In *International Conference on Computer Graphics Theory and Applications (GRAPP)*, 2008.

[39] Yonggao Yang, Jim X. Chen, and Mohsen Beheshti. Nonlinear perspective projections and magic lenses: 3d view deformation. *Computer Graphics and Applications (CGA)*, 2005.

[40] W. Zhang and J. Kosecka. Image based localization in urban environments. In *3D Data Processing Visualization and Transmission (3DPVT)*, 2006.