Improve 3D models from 2D images

T. Kostelijk
mailtjerk@gmail.com

October 24, 2011

Contents

1 Comments on this thesis

Dear Frans,

I write this in English so Isaac can also read it.
You are looking at an update of my thesis. Isaac read it and gave me feedback. His feedback and all feedback you gave me earlier are incorporated except for:

- the test on different datasets Because the 3D model isn't ready at the time of writing, I only used different datasets at the skyline detector.
- parametervalues Houghline (I'll do this on the end)
- large introduction In 'how to write a thesis' they advice to write the introduction at the end (when all chapters are finished) so I wait with this chapter.

Questions for you:

- Should I include result images of the related work?
- My new method of line-wall association (section 3.4.4) is much better and more intuitive then my old method (section 4). Therefor I am considering to discard the old method. However, I could also keep the old method and use it as a test against the new one. What is your advice about this?
- Could you give me feedback about the level of detail, where do I go in to much and where is more detail desired?
- Isaac told me to make the future work section shorter, do you agree with this? And if so could you tell me which part I should drop? Furthermore my discussion section is very short, I think it is best to add more discussion when the other datasets and results are ready. What do you think?

- I have tried to write down some formulas and symbols but didn't do it everywhere and didn't do it consistently. Would you give me feedback where to put (more) formulas or symbols and where to discard them?

- I have troubles with putting my work of chapter 3 in context with the work of others because I can't find any related work about this (specific) subject. How do I deal with that?

- I have difficulties with deciding wheter to put a text in the preliminaries, chapter or in an appendix. Could you give me feedback about this?

Please let me know when the feedback is done and let's make an appointment to discuss it. I think Isaacs presence will not be necesarry, right Isaac? I am available most of the days. Please let me know when you are available.

Thanks in advance for reading my thesis chapter and giving me useful feedback.

Kind regards,
Tjerk Kostelijk
mailtjerk@gmail.com
0614898323

2 Skyline detection

2.1 Introduction

If we take a regular image on which both sky and earth are present, there is often a clear separation between them. This separation is called the skyline. The detection of this skyline has proven to be a very successful computer vision application in a wide range of domains ranging from object detection, guiding flights, car localization, etc. In this project it is used at urban images to provide a contour of a building. The contour will be used to provide 3D information about the scene. This is a brand new purpose of skyline detection.
For our application the skyline detector must be accurate, robust and must operate without any user interaction. This makes it different from existing skyline techniques (e.g. [1],[5],[2]).
The organization of this chapter is as follows: First we give a summary of related work on skyline detection. Next we explain how we developed a new robust skyline detection algorithm. Then we present and discuss some results and, finally, we conclude.

2.1.1 Related work

Castano et al. [1] present a clear introduction of different skyline detection techniques.

Detection of dust devils and clouds on Mars In [1], mars Exploration Rovers are used to detect clouds and dust devils on Mars. Their approach is to first identify the sky and then determine if there are clouds in the region segmented as sky. The sky is detected by an innovative algorithm that consists of three steps. First they place seeds in a sliding window whenever the homogeneity of the window is high. Then they grow this seeds in the direction of edges which are estimated using a Sobel edge detector. Finally each pixel located above the grew seeds is classified as sky.

Of the discussed methods so far, this seed growing method looks like the most sophisticated one, as it is accurate and autonomous. However, we have a stable scene with sharp edges at the building contour so this method would be an implementation overkill.

Horizon detection for Unmanned Air Vehicles In this domain [5], scientists detect the horizon to stabilize and control the flight of Unmanned Air Vehicles.
S.M. Ettinger et all [5] use a horizon detector that takes advantage of the high altitude of the vehicle, in that way the horizon is approximated to be a straight line. This straight line separates the image into sky and ground. They use color as a measure of appearance and generate two color distributions: one for the sky and one for the ground. They use the covariance and the eigen values of the distributions to guide a bisection search for the best separation. The line that best separates the two distributions is determined to be the skyline.

This work is not applicable for detecting a building contour as the straight line assumption doesn't work. But it needs to be mentioned that some ideas for section 3.3 are created because the building has walls that have straight lines, an assumption is made about partially straight lines.

Planetary Rover localization Cozman et al. [2] use skyline detection in planetary rovers to estimate their location. To recover the rover's position they match image structures with a given map of the landscape (hills, roads, etc) and align both images. The matching process was first based on feature matching. In an improved version the matching process was done by searching for correspondences among dense structures in the image and on the given map, so called signal based matching.
The advantage of their algorithm is the simplicity and effectiveness, this could make their algorithm suitable for this project. A big drawback is that they prefer speed over accuracy. To increase accuracy, the detector is part of an interactive system where an operator refines the skyline. For our application the skyline detector must operate without any user interaction. Furthermore it has to be robust and accurate because it provides a basis for the extraction of straight lines which offer an estimation of the buildingwall heights.
We decided to use the Rover method [2] as a basis and build a custom algorithm with higher accuracy on top of that. This is explained in the next section.

2.2 Method

2.2.1 Situation and assumptions

Before we present the method let's define the situation and make some assumptions.

Definition: skyline in urban scene
A skyline in an urban scene is a set of points of the size w (where w is the width of the image) where each point describes the location of the transition from the sky to an object (e.g. a building) which is connected to the earth.
The question is: how are we going to detect the sky-building transition point?
In general, the color of the sky is very different than the color of the building. The use of a color-based edge detector would be an intuitive decision. However, the sky and the building itself also contains edges (caused by for example clouds and windows). So how do we determine the right edge? The number of possible edges could be decreased by thresholding the intensity of the edge but it would still be a difficult task to determine the right edge. Furthermore the algorithm would not be robust to a change in the lightning conditions, influenced heavily by the weather.

To solve this problem we draw an assumption that is based on the idea of [2]. Instead of using the sharpest edge we take the most upper sharp edge and classify this edge as the skyline.
Top sharp edge assumption *The first sharp edge (seen from top to bottom) in the image represents the skyline.*

2.2.2 Related algorithm

To put our work in context, we first describe a related skyline detection algorithm as presented in [2].

To increase the difference between sharp and vague edges, and to let sharp edges stand out more and vague edges disappear, the images are converted to Gaussian smoothed images. The smoothed image is first divided in $\#w$ columns. Next, each column produces a new column that stores its vertical derivatives. This is called the smoothed intensity gradient. The values of this column are high when a big change in color happens (e.g. an edge is detected) at that location on the image. The system walks through the values of a column, starting from the top. When it detects a pixel with a

gradient higher then a certain threshold it stores its y-value (the location of the highest sharp edge of that column) and continues to the next column. The result is a set of y coordinates of length w, that represent the skyline.

2.2.3 Improved algorithm

Taking the smoothed intensity gradient is the most basic method of edge detection and has the disadvantage that it is not robust to more vague edges. It is not surprising that the algorithm in [2] was used in an interactive system where the user has to refine the result.

Our aim is to develop a autonomous skyline detector, the only user interaction that we allow is to provide the system some parameters. We will now discuss the adaptations that we developed with respect to the related algorithm.

The column based approach of the related algorithm seems to be very useful and is therefore unchanged. The related algorithm uses the smoothed intensity gradient as a method to detect edges. Because of the accuracy disadvantage of this method we took another approach in detecting edges. We tested different edge detecting types.
The output of the different edge detection techniques was studied on an empirical basis and the Canny edge detector came with the most promising results. This is probably because Canny is a more advanced edge detector. It uses two thresholds, one to detect strong and one to detect weak edges. It includes the weak edges in the output, but only if they are connected to strong edges. In table 2.2.3 we list Matlab's build in edge detectors together with the method explanation.
Because the optimal edge detector type can be scene depended, it can be set by the user as a parameter in our functions.

The Canny edge detector outputs a binary image, therefore the column inlier threshold is set to 1, which means that it finds the first pixel that is white. This is, as in the related algorithm, done from top to bottom for every column in the image.

Because we know we are looking for sharp edges we improved the algorithm by introducing two preprocessing steps. First the contrast of the image is increased, this makes sharp edges stand out more. Secondly the image undertakes an extra Gaussian blur, this removes a large part of the noise. Note

Table 1: Different edge detectors explained

Edge detecting type	method
Sobel	The Sobel method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of the image is maximum.
Prewitt	The Prewitt method finds edges using the Prewitt approximation to the derivative. It returns edges at those points where the gradient of the image is maximum.
Roberts	The Roberts method finds edges using the Roberts approximation to the derivative. It returns edges at those points where the gradient of the image is maximum.
Laplacian	The Laplacian of Gaussian method finds edges by looking for zero crossings after filtering the image with a Laplacian of Gaussian filter.
zero-cross	The zero-cross method finds edges by looking for zero crossings after filtering the image with a filter you specify.
Canny	The Canny method finds edges by looking for local maxima of the gradient of the image. The gradient is calculated using the derivative of a Gaussian filter. The method uses two thresholds, to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be fooled by noise, and more likely to detect true weak edges.

that depending on the edge detector type this could mean that the image is blurred twice.

The system now has several parameters which have to be set manually by the user:

- Contrast,
- Intensity (window size) of Gaussian blur,
- Edge detector threshold.

If the user introduces a new dataset these parameters need to be configured as the image quality and lightning condition are scene depended.

2.3 Results

Two different datasets are used.
The first dataset is the *Floriande* dataset, which is included in the Fit3d toolbox [4]. The dataset consists of eight images with resolution 1728x1152px.
The second dataset is named the *Spil* dataset and it contains 40 images with resolution 3072x2304px.

The output of the edge detector and skyline detector on the *Floriande* dataset [4] can be seen in Figure 2.3 We emphasize the effect of different thresholds of the edge detector on the *Spil* dataset in Figure 2.3.

2.4 Discussion

The largest part of the building edge is detected, this is a ... result given the algorithm operates without any user interaction.
The system assumes that the first sharp edge (seen from top to bottom) is always the building contour. As can be seen in Figure **??**, not every skyline element is placed on the building contour but placed on, for example, the streetlight or the tree. We define them as outliers. Other objects in the air that appear above the building but contain a sharp edge, for example an aircraft, will also turn into outliers. This is a drawback of the column based method.
The advantage of the method is its simplicity (and therefore low complexity) and autonomity. We didn't focus on the development of a more accurate skyline detector because this result is good enough for our purpose. We removed the outliers in a different module of the system, this is described in the next section.

2.5 Conclusion and Future work

A detailed research on related work research was done. We introduced a brand new application of skyline detection: the extraction of a building contour. We build an algorithm on top of a successful existing algorithm. The algorithm doesn't depend on human intervention and is robust enough to give (together with the module explained in the next section) accurate results.
It is interesting to denote that the skyline detector is a stand alone method and can be optimized individually without any knowledge of the other modules of the project.

Although the outlier removal procedure is done in a separate module, it would be interesting future work to develop a skyline detector which is more robust to outliers. Most of the related work is based on detecting parts that are classified as sky and parts that are classified as ground. The idea of detecting the sky and ground could be replaced by detecting the sky and a building. The distinctive textures of the buildings (repeating bricks) could be of great use for the classification. After that, a rough building contour could be estimated by using the highest building pixel for every column. Detailed edge detection could be done in this neighborhood. In this way the outliers, (e.g. the lamp and the tree) are filtered and no secondary outlier removal procedure is needed.

(a) The output of the edge detector

(b) The output of the skyline detector. The skyline elements are marked red

Figure 1: The output of the edge detector and the skyline detector.

(a) TODO

3 Improving the 3D building

3.1 Introduction

In the previous chapter we extracted the building contour with the skyline detector. The output was a set of 2D points and we collected this set for every view of the building. The aim of this chapter is to use this set of points to improve a basic 3D model.
The point cloud from the skyline detector included a lot of noise caused mostly by occluding objects like trees. How do we detect those outliers? And if we have an outlier free point cloud how can we use this information to improve a basic 3D model? And how do we know which point is associated to which part of the building? These questions are addressed in this chapter.
We present a stepwise solution. First *Openstreetmap* is used to generate a basic 3D model of a building. Secondly the set of points returned by the skyline detector is transfered to a set of lines. Then each line segment is assigned to a wall of the building. After this the lines are projected to these assigned walls in the 3D model. The projections are used to estimate new height values of the building walls. The 3D model is finally improved by updating the walls according these heights.
We will now elaborate on each step.

3.2 Generating the 3D model

The 3D model is generated using a basis (groundplane) which is manually extended. The basis (viewed from top) of the generated 3D model is originated from *Openstreetmap*.
Openstreetmap, see Figure 1,is a freely accessible 2D map generated by users all over the world. It contains information about streets, building contours, building functions, museums, etc. We are interested in the building contours. We take a snapshot of one particular area and extract this building contour. This is a set of ordered points where each point corresponds to a corner of the building. Next we link these points to walls.
Because the map is based on aerial images, it is in 2D and contains no information regarding the height of each wall.
The final 3D model is generated by starting with the the 2D building contour as its basis. We estimate the wall heights by hand and overestimate this height. After this we use the height to extend the 2D basis in the opposite gravity direction, resulting in a 3D model.

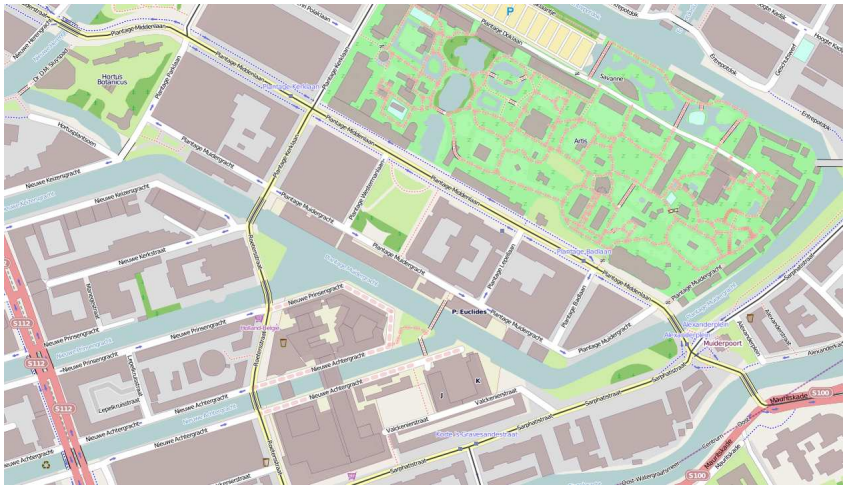

Figure 3:

Gravity aligned walls assumption
The walls of the building are aligned in the opposite direction of the gravity which is orthogonal to the 2D basis from ***Openstreetmap****.*

An example of the 3D model can be seen in Figure 4.

3.3 Extracting line segments

The skyline detector returned a set of points. If some of these points lie on the same line, they form a straight line. Straight lines are likely to come from the building contour. If we have a method that extracts these straight line segments, we can use these line segments to find parts of the building contour and finally use this to improve the 3D model.
Unfortunately a problem arises because some points are outliers. To discard these outliers we detect the inliers and consider the remainder as outliers. In this section we draw an assumption and we explain how straight line segments are extracted and how outliers are discarded at the same time.

Assuming a flat roof Many urban areas contain buildings with a flat roof. This means that the contour of the building is mostly formed by straight lines. We use this fact to simplify our problem:

Flat roof assumption
We assume each building has a flat roof, implicating that each building wall has a straight upper contour. The walls may have different heights but the roof should be flat.

This assumption is very useful as it let us focus on finding the height of the building walls from the building contour without having to concern for (complex) rooftypes. This doesn't mean that the method described in this thesis is unusable for buildings that contain roofs. E.g. with a small adjustment the method could be used to at least gain the building height which is a useful application.
Ideas about how to handle rooftypes explicitly can be found in the Future work section.

3.3.1 Hough transform

A widely used method for extracting line segments is the Hough transform [3] We regard this as a suitable method because it is used a lot for this kind of problems. This is probably because it is unique in its low complexity (compared to other (iterative) methods like $RANSAC$).
In the Hough transform, the main idea is to consider the characteristics of a straight line not as its image points $(x1, y1)$, $(x2, y2)$, etc., but in terms of the parameters of the straight line formula $y = mx + b$. i.e., the slope parameter m and the intercept parameter b.
The input of a Hough transform is a binary image, in our case the output of the skyline detector (chapter ?).
If a pixel is classified as a skyline pixel (a pixel that lies on the skyline according the skyline detector), the Hough transform increases a vote value for every valid line (m,b pair) that crosses this particular pixel. Lines (m, b pairs) that receive a large amount of votes contain a large amount of skyline pixels.
Because the algorithm detects straight lines containing only skyline pixels it is most likely that it returns parts of the skyline and therefore the building contour.
The Hough transform is implemented in *Matlab* and has some useful extra functions.The algorithm can optionally return the start and endpoint of the found lines which is very useful as it helps us to associate which part of the building is described by the line.

Furthermore it has the parameter *FillGap* that specifies the distance between two line segments associated with the same m,b pair. When this inter line segment distance is less then the *FillGap* parameter, it merges the line segments into a single line segment. In our application this parameter is of particular interest when we want to merge lines that are interrupted by for example an occluding tree.
Results of the Hough transform on the 2D output of the skyline detector are displayed and evaluated in the Result section.

3.4 Project the skyline to the building

The Hough transform of the previous section returned a set of 2D line segments which present parts of the skyline. If we can find a way to project these lines to the building we can improve our basic 3D model.
This involves three steps, the calibration of the camera, the projection to the building and a way to associate a skylinepart with a specific buildingpart.

3.4.1 Camera calibration

Extrinsic parameters How do we find the the camera centers (positions) and camera viewing directions. In some systems this is known on forehand by measuring the cameras position and orientation at the scene. In other systems, as in ours, this is calculated afterwards. We calculated the values afterwards and used the *Fit3d toolbox* [4] for this. See preliminaries () for detail.

Intrinsic parameters As it is our own dataset we also know what camera took the picture so we can calculate a camera specific Calibration matrix. The calibration matrix is retrieved by making images of a chessboard in different positions and orientations together with the Bouget toolbox. More on this in the preliminaries.

3.4.2 From image point (2D) to possible points in scene (3D)

The line segment that was returned by the Hough transform consists of two endpoints v and w. These endpoints are in 2D but are recorded in a 3D scene and therefore present a 3D point in space. The value of the 3rd dimension represents the distance from the 3D point to the camera that took the picture. Unfortunately this value is unknown, however because we calibrated the camera we can reduce the possible points in 3D space to a

line.
See Figure
We know:

- The 2D properties about the pixel, (the location of the pixel in the image).
- The camera's internal parameters (Calibration matrix)
- The translation of the camera
- The rotation of the camera

We can now define the line of possible points in 3D space by two coordinates:

- C, the center of the camera, and
- $K'p$, the point that lies on the retina of the camera, where K is the Calibration matrix and p is the homogeneous pixel coordinate.

Now we have the two required coordinates, we can set up an equation of the line of possible points in 3D. This is done as described in preliminaries (Transform coordinates to a line equation).

3.4.3 Intersecting with walls

As described in the 3D building model is divided into different walls. These walls are described by two ground coordinates and a direction which is the y-direction as assumed by the *Gravity aligned walls assumption.*
As you can read in preliminaries (Planes and walls) the walls are transformed into planes. This is done for two reasons: first this transformation is required to calculate the intersection properly. Secondly, because the 3D model is an estimate, the walls maybe just to small which could result in no intersection which is unwanted.
Now we have the building divided op into planes we can intersect the lines. For every line segment we have two endpoints which are now projected on the building walls This results in 2xlxw points in 3D, grouped by the line segments, where l is the number of lines and w is the number of walls.
The next challenge is to reduce the number of intersection for every line segment to one. In other words, to determine the wall that is responsible for the line segment. This is later on used to update the height of the wall of the 3D model.

3.4.4 Associating line segments with building walls

Assumptions We consider each building consisting of seperate walls and associate each line segment with a wall of the building that is most likely responsible for that line segment.

Unique wall assumption:
We assume that the output of the Hough transform are line segments that each represent a single wall of the building, e.g. if the Hough transform finds 3 line segments there are 3 walls present.

Building wall appearence assumption:
We assume that every line segment in the output of the Hough transform represent (a part of) the upper side of a specific wall of the building.
If a line segment is assumed to represent a single wall then the projection to that wall should have a large overlap with this wall. To be more precise, let's define l in $\mathbb{R}2$ as a line segment that is generated by the Hough transform. If we project l to the plane spanned by a wall W we get a line l_{proj_W} in $\mathbb{R}3$. If we assume l to come from contour of wall W, then l_{proj_W} should have a large intersection with this wall W (as we overestimated the height of the building). We call this the line-wall overlap value, *lwo*. A mathematical definition of *lwo* follows. Note that the projection of l with the other walls should have a small *lwo* value.
Largest line-wall overlap assumption:
A line segment describes the contour of the wall where its projection has the largest overlap with.

Having defined the assumptions, the situation and the idea behind the line-wall association, we can now explain the line-wall matching algorithm.

A line segment is projected to all walls and the amount of line-wall overlap, *lwo* is calculated. The wall with the largest overlap with the specific line segment is classified as the most likely wall for that line segment. Next the line segments are projected to their most likely wall and the algorithm outputs this set of lines in $\mathbb{R}3$.
This line-wall overlap is calculated in different steps. First, different types of overlap are explained. Secondly the algorithm determines the *overlap type*, then the overlap amount is determined and finally the amount of overlap is normalized.

l_{proj_W} can overlap W in four different scenarios, this is explained in Figure 10. The wall W is spanned by $abcd$, and l_{proj_W} is spanned by vw.

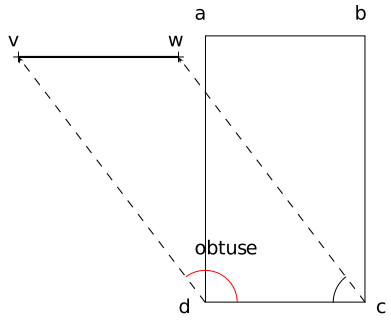

Figure 10a

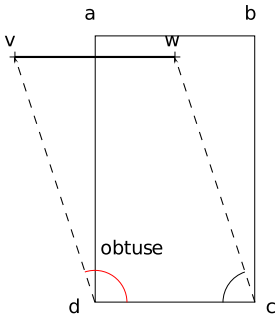

Figure 10b

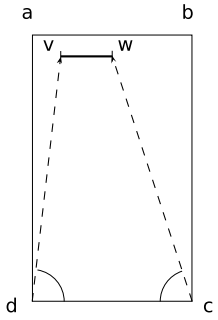

Figure 10c

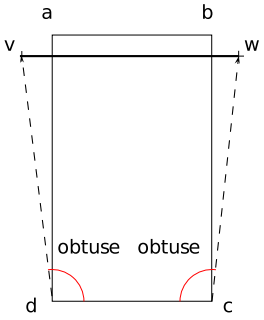

Figure 10d

Figure 4:

The type of overlap is defined by exposing the endpoints of the line segments to an *in polygon* test, where the polygon represents a wall of the building (e.g. *abcd* in Figure 10).
The table below represents the types of overlap with the corresponding number of points that pass the *in polygon* test and their possible line-wall overlap value.

Type of line-wall overlap	Points in polygon	Line-wall overlap	Figure
No overlap	0	0	10a
Partial overlap	1	[0..1]	10b
Full overlap (included)	2	1	10c
Full overlap (overextended)	0	1	10d

No overlap If the point in polygon test returns 0, the line-wall overlap calculation is skipped and 0 is returned. The remaining overlap types, partial and full, are treated individually:

Partial overlap Let's first consider the partial overlap type (Figure 10b), the *in polygon* test returned 1, that means that one of the line segments endpoint lies inside and one lies outside the wall.
To calculate the amount of line-wall overlap, the line segment is cropped to the part that overlaps the wall and the length is measured.
The cropped line has two coordinates, first of course the point that passed the *in polygon* test and secondly the intersection of the line segment with one of the vertical wall sides (*da* or *cb* from Figure 10b).
We assume the walls to be of infinite height, therefore the partial overlapping line segment always intersects one of the vertical wall sides.
To determine which of the two vertical wall sides is crossed, we determine on which side the point that doesn't lie in the polygon (v) is on. This is done by an angle comparison (as in section ?).
First, two groups of two vectors are defined: dv, dc and cw, cd (see Figure 10b). We measure the angles between the vectors and call them $\angle d$, and $\angle c$. Because one of the line segment endpoints lies outside the wall $\angle d$ or $\angle c$ is obtuse, in this case $\angle d$ is obtuse. (Note that this holds because the walls are orthogonal to the basis which we assumed in the *Gravity aligned walls assumption*
To be more precise:

- Ff $\angle d$ is obtuse, the left vertical wall side da, is crossed.

- If $\angle c$ is obtuse, the right vertical wall side cb, is crossed.

The angles are acute or obtuse if the dot product of the vectors involved are respectively positive or negative. The advantage of this method is that it's

simple and has low computational costs.

Line-wall overlap calculation
The amount of line-wall overlap is calculated by cutting of the point where l intersects the determined vertical wall side (da or cb) and measuring its remaining length.

Full or no overlap Now let's consider the overlap types where the *in polygon* test returned 0. As you can see in Figure 10a and 10d this resulted in either full or no overlap. Again we analyze the vector angles to determine the remaining overlap-type. If only one of the angles is obtuse with no points in the polygon, like in Figure 10a, the whole line segment lies outside the wall: an overlap value of zero is returned.
Otherwise, if both angles $\angle d$ and $\angle c$ are obtuse or acute (Figure 10d), both endpoints lie on a different side of the wall, and they cross the wall somewhere in between. Full overlap is concluded here.
The amount of overlap is now calculated by measuring the length of the line segment which is cut down by his intersections with da and cb. In this case this is the same as line dc, but its easy to see that this is not the case when vw is not parallel to dc.

Line-wall overlap normalization Finally the line-wall overlap is normalized by the line segments length:

$$\alpha_l = \frac{lwo}{|l|} \tag{1}$$

Where α_l is the normalized line-wall overlap, lwo is the unnormalized amount of line-wall overlap, and ($|l|$) is the total length of the line segment (uncut). The intuition behind this is that line segments that are likely to present a wall not only have a large overlap but also have a small part that has no overlap, the missing overlap should have a negative effect. By calculating the relative overlap, both amounts of overlap and -missing overlap are taken into account.
The maximum of the normalized line-wall overlap is used to associate a line segment with its most likely wall. To summarize, the overlap type is defined by calculating the numbers of in polygon points and evaluating two dotproducts. Next the line segment is cut off depending on the overlap type

and the line is normalized.

Now we have determined the normalized line-wall overlap, we use this to search for the correct line-wall association. This is achieved by associating a line segment with the wall that has the largest line-wall overlap.

3.4.5 Improving the 3D model by wall height estimation

In the previous section we associated the line segments with their most likely wall. In this section this information is used to estimate the heights of the walls which will eventually be used to update the 3D model in the next section.
Now all line segments are associated with a certain wall, we re-project the line segment from the different views on their associated wall. The re-projection is done by intersecting both endpoints of the line segment to the plane that is spanned by the associated wall.
Next the 3D intersection points are collected and averaged, this gives us an average of the midpoints of the projected line segments. We do this for every wall separately, returning the average height of the line segments. These averages are then used as the new heights of the walls of the building. Note that this is only permitted in presence of the *flat roof assumption*.
The new individual heights are used in the 3D model by adjusting the location of the existing upper corner points of the walls. We copy the bottom left and right corner points and add the estimated height from the previous section to its y-value. The y-value is the direction of the gravity which is permitted by the *Gravity aligned walls assumption*.

3.5 Results

Let's return to the output of the skyline detector in Figure 1. Figure 3 shows the top 3 longest Houghlines, the endpoints are denoted with a black and blue cross. All three line segments lie on the building contour. The left line segment covers only a part of the building wall. The middle line segment covers the full wall. The left and middle line segment are connected. The right line segment covers the wall until the tree occludes.
Figure 3 displays the line segments (originated from different views) projected on to their associated walls. For a clear view we've only selected the lines that were associated with three specific walls of the building. The red cross in the middle of the line represents the average of its endpoints.

Figure 5:

Figure 5 displays the updated 3D model. The corner points of the walls are adjusted according the calculated wall heights. The green plane displays the augmented wall. The left and middle wall are extended and the right wall is shortened.

3.6 Discussion

As can be seen in Figure 3 the left line segment doesn't cover the whole building wall. This is caused by the use of strict parameters in the Hough transform (like a small line thickness parameter). If some ascending skyline pixels fall just outside the Houghlines, a gap is created and the line segment is cut down at that point. This is however not a big problem because the lines are long enough to produce a good wall height estimate. Furthermore there are at least 5 other lines (originated from different views) that support the estimate for this wall.

Figure 6:

3.7 Conclusion

To conclude, we showed that a Houghline transform is a useful method to detect outliers and find prominent structure in the contour of a building with a flat roof. We introduced a method to pair up line segments with their associated walls. This was used to produce new wall heights which were propagated to the 3D model. Existing and novel AI computer vision techniques were powerfully combined resulting in an accurate 3D model based on only a few calibrated 2D images.

3.8 Future work

As can be seen in Figure 4 two line segments appear on the same single wall. This means that they have a double influence on the average wall height, which is unjustified. A simple solution would be to add a normalization pre-process step, so each view has only one wall height vote per wall. A more decent solution would be to merge the two (or more) line segments to a single line segment. This could be achieved with an iterative Hough transform where the *FillGap* parameter is increased in each iteration. E.g.

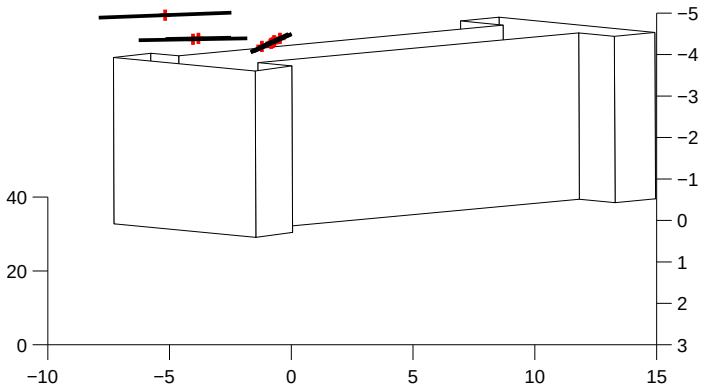

Figure 7:

for the right wall of the building in Figure 4 two iterations would be enough, the *FillGap* parameter needs to be at least as big as the occluding tree in the second iteration.

In this thesis little is discussed about the computational costs. This is because the computations are done efficiently (e.g. using matrix multiplications in Matlab) and off line, making the calculation process accomplishable in reasonable time. To make the application real time the next speedup would be useful.
To determine the best line-wall association the line segments are now projected to every wall and for every wall the amount of line-wall overlap is calculated. This is computational very expensive and looks a bit like an overkill.
It would be a significant speedup to reduce the set of walls to only the walls that contain the middle point of the line segments. To be more concrete the middle point needs te be calculated by averaging the line segments endpoints, this middlepoint is used in the *in polygon* test for every wall. Next the line-wall association algorithm only treats the walls that pass this test. The downside of this method is that it will be inaccurate, resulting in more false negatives: a linesegment that overlaps the wall with only 1/3 could

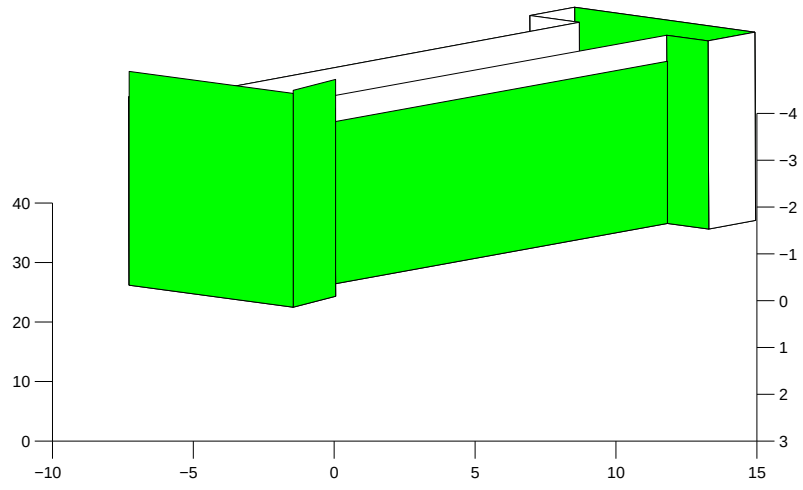

Figure 8:

be of use in the height estimation but instead it is discarded. What can be concluded is that there is a trade of in the accurateness of the height estimation and the computational costs.

3.8.1 Alternative roofs

To make the algorithm more generic, the flat roof assumption could be stretched or even discarded. We'll now consider other roof types and discuss what adaptations the system should require to handle these. In Figure 5, 6 different roof shapes are displayed.

Consider the *Gable Roof* in Figure 5, it is a roof consisting of two planes which are not parallel with the facade of the building. This makes the problem of extracting the 3D model more complex, but not infeasible.
Because we assume that the roof images are taken from the ground, the skyline detector will always detect the top of the building. In case of a flat roof this is also the top of the building walls. In case of an alternative roof,

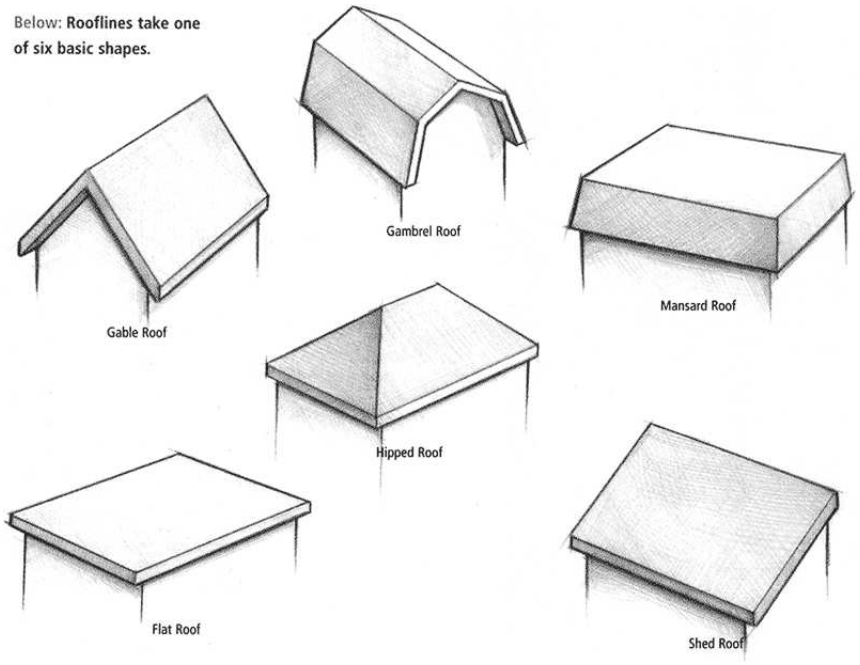

Figure 9:

this will be just the top of the building. The building walls however could lie a lot lower, therefore something else needs to be developed to find the wall heights. It would be useful to develop a method that can detect which roof type we are dealing with, what the wall heights are, and finally generate an entire 3D model.
Some ideas about this are now proposed:

- Use an object detector to detect doors, windows and dormers so the number of floors, the location of the wall-roof separation and the exclusion of some roof types (e.g. a dormer is never located on a flat roof) could be determined.

- Use the Hough transform to search for horizontal lines to detect the wall-roof separation, and use the the ground plane and the top roof line to guid the search. Some building have a gutter, because of this the number of horizontal lines on the wall-roof separation will be larger which could be of great use.

- Use geographic information (a database of roof types) with gps location to classify the roof type.

- The skyline detector detects the building height, if we could use predefined information about the ratio between the wall height and total height of the building, the wall heights could be estimated.

Assuming we determined the roof type,the building height and wall heights, the 3D model could easily be generated. For the *Gable* roof for example this will involve connecting two surfaces from the upper side of the walls with the high roof line (returned by the skyline detector). For the other roof types, the building height and wall height together with a template structure of the roof could be used to generate the 3D model.

4 My old method of line-wall association

It would be straightforward to use the method of the skyline detector of chapter ?Instead of a skyline pixel we could take the line segments endpoints and project it onto the building walls. The wall with the shortest distance will be assigned to the line segment.

And to update the specific wall we first need to know with a high probability of being correct which wall the line segment presents But this method introduces a problem: some of the line segments have endpoints that lie at the corner of the building. These line segments could easily be associated with the neighboring wall. Because the 3D model is a rough estimate this could lead to bad results. In the corner case it is not clear to which wall the line segment belongs because both endpoints do not agree on the same wall. To solve this problem some heuristic methods are developed and tested. The following heuristic is both simple and effective. The heuristic uses the importance of the middle point of the line segment. This middle point has a low change of being on a building corner and the on average biggest change of being on the wall we are looking for. Therefor we discard the endpoints and use the middle point the endpoints to determine the right wall. As in the previous section this middle point is intersected with all planes spanned by the walls. The line segment is stored to the wall with the shortest distance. The output of this part of the algorithm is for every wall a bunch of

associated line segments originated from different views.

in section Results was this text:
The left and middle line segment of Figure 3 are a good example of the corner problem. Both endpoints that lie on the corner could easily be associated with the wrong wall (even if the rough 3D model is very accurate). Fortunately we use the middle point of the line segment to determine the correct wall. This works well as its 100% accurate (for this dataset).

5 Preliminaries (DRAFT)

5.1 Fit3d toolbox

The fit3D toolbox [4] TODO explain structure by motion etc.

5.2 Transform coordinates to a line equation

TODO, explain 2 coords to an line equation

5.3 Coordination systems

The camera can be described in the same coordinate system as the 3D point we are looking for, we call this the world coordinate system. The camera center is a value in $\mathbb{R}3$ that represents the camera's position in the world coordinate system. If the camera rotates, it rotates around this point. TODO elaborate

5.4 Homogenous coordinate

TODO explain how to add extra homogeneous dimension (x,y,1) and why this is

5.5 Planes and walls

TODO How to span plane by wall coords

5.6 Getting the camera centers

TODO Explain here about fit3d toolbox

5.7 Calibrating the camera with the Bouget toolbox

TODO

6 References

test1

References

[1] Andres Castano, Alex Fukunaga, Jeffrey Biesiadecki, Lynn Neakrase, Patrick Whelley, Ronald Greeley, Mark Lemmon, Rebecca Castano, and Steve Chien. Automatic detection of dust devils and clouds on mars. *Mach. Vision Appl.*, 19:467–482, September 2008.

[2] Fabio Cozman, Eric Krotkov, and Carlos Guestrin. Outdoor visual position estimation for planetary rovers. *Auton. Robots*, 9:135–150, September 2000.

[3] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15:11–15, January 1972.

[4] I. Esteban, J. Dijk, and F.C.A. Groen. Fit3d toolbox: multiple view geometry and 3d reconstruction for matlab. In *International Symposium on Security and Defence Europe (SPIE)*, 2010.

[5] Michael C. Nechyba, Peter G. Ifju, and Martin Waszak. Vision-guided flight stability and control for micro air vehicles. In *IEEE/RSJ Int Conf on Robots and Systems*, pages 2134–2140, 2002.

test2