Interpreting 3D Scenes Using Object-level Constraints

Rehan Hameed
Stanford University
353 Serra Mall, Stanford CA
rhameed@stanford.edu

Abstract

As humans we are able to understand the structure of a scene without the need for a precisely calibrated camera system. We are able to employ a number of cues such as an approximate understanding of our height and field of view as well as the knowledge that we have accumulated about the world such as expected dimensions of objects in the world, their color and texture, their shape, and their expected location in a scene. Using these cues humans are able to get an approximate yet precise and detailed understanding of the scene including identification of various objects as well as their depth and height etc. In this project we explore the possibility of using a similar approach in our computer vision task of interpreting a 2D image to extract 3D information. In particular we implement a system which considers just a subset of these parameters, and identifies objects in the image purely by comparing geometric constraints on image elements with constraints on possible objects in 3D world.. Using some knowledge about visual system parameters and 3D geometry we compute constraints on possible dimensions and locations of each image 'element' in the 3D world and compare that against an 'object database' to find plausible matches. Moreover the system also attempts to use feedback from the scene geometry to improve our estimate of visual system parameters. Our particular case study defines a world with 5 distinct object classes and interprets an image containing some of these objects to reconstruct the 3D scene structure from a single 2D image. Our results show that in our limited universe we get a fairly accurate reconstruction of our target image even with a very rough estimate of visual system parameters.

1. Introduction

Human beings rely on a number of cues, which come together to aid in our understanding of this world. One big tool is off course our built-in stereovision using two eyes. However our visual system is not just a sophisticated and well-calibrated stereo system employing object correspondences and 3d geometry to extract depth at each point in the scene in front of our eyes. In fact from our everyday experience we know that even if we close one eye and shut down the stereo engine, our '3D' understanding of the world is still fairly complete. So what are some other helping aids we use in addition to the stereo engine? These could include, but are not limited to, the following:

1. An implicit understanding of how our height relates to height of other objects in the world
2. Implicit understanding of field of view captured by our eyes
3. Approximate understanding of our viewing angle (are we looking up or down and by what extent)
4. Are our eyes focused at a close distance or far?
5. What are the properties of objects in our world:
    a. Height and width
    b. Texture and color (sky is typically blue)
    c. Location in the world (cars don't fly)
6. Cues from the geometry e.g. vanishing points
7. Depth of field, etc.

Lets look at some images to for further elaboration. As first example consider the two images of the same cottage in Figure 1. Even though in one of the images the cottage occupies a much smaller part of the image, still we roughly get the same idea about its scale from both images. There are two important helping aids, which guide us here - the horizon line and the ground plane. The fact that the cottage top rises above the horizon line tells us that the object in question is taller than the viewer height. Also by comparing the distance from ground plane to horizon line and from horizon line to top of the cottage, we get a fair idea of its height. Combining this with an understanding of field of view captured in the image, our brain calculates that the smaller cottage is very far.

To extend this further let's look at the third cottage image where the horizon line as well as most of the ground has been taken away. Even though we no longer have an explicit reference, the sense of scale is still there though perhaps less precise. From a purely geometric perspective it can be anything from a huge structure really far away to a tiny box right in front of us. But our knowledge about his object is enough to still give us a sense of its scale. In other words only one interpretation "makes sense" out of all possibilities from 3D geometry.

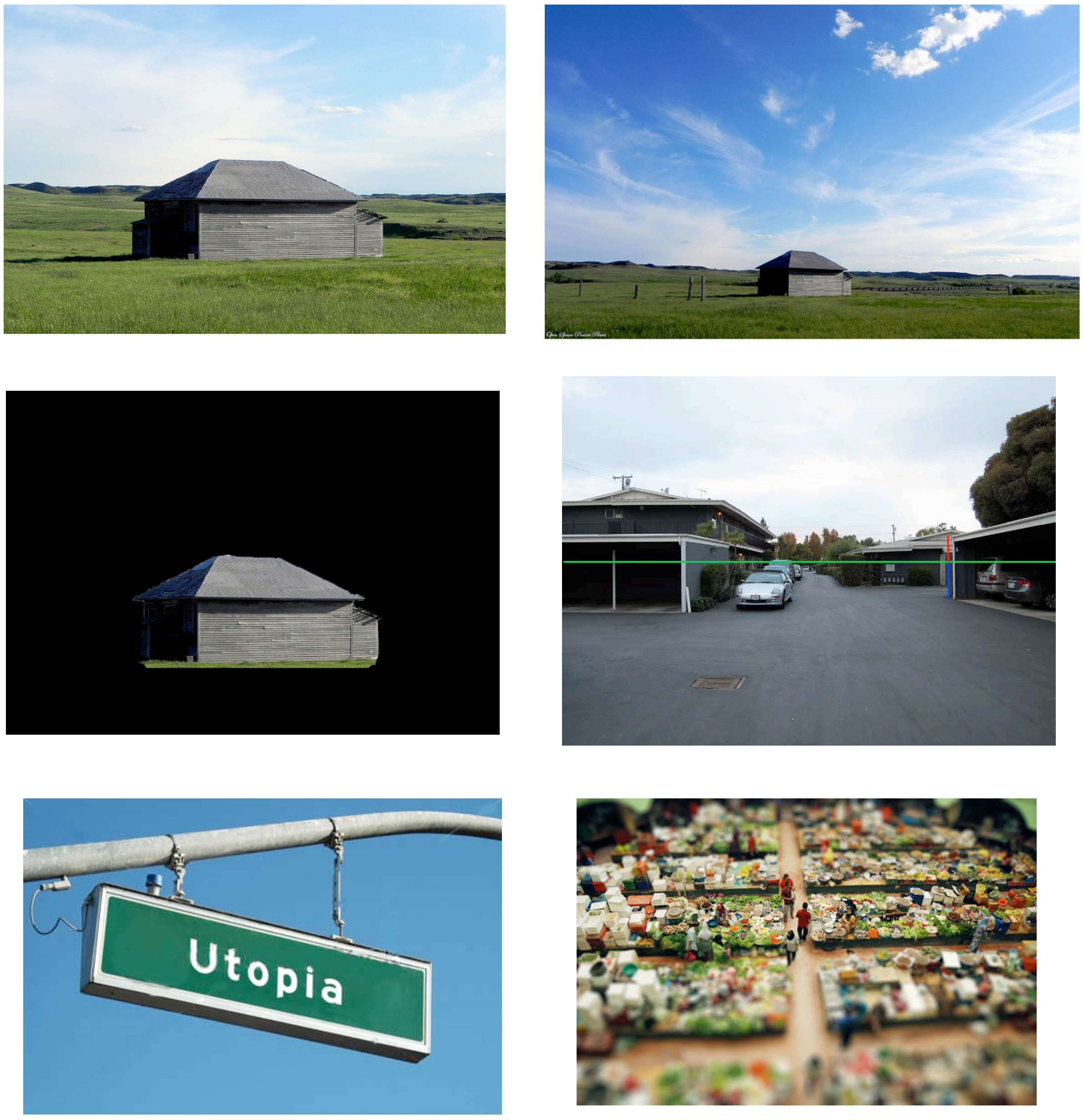


Figure 0 – Example images from the world

As a second example lets consider the street image next to this cottage. Again we know right away that anything above the middle of the image must be above our eye level. Which tells us that the structures that we are looking at in the image must be high things like walls and buildings. Moreover by the relationship between the ground plane, horizon line and the white pole, we can say that the white pole is just over 1.5 times the viewer height. Moreover we can see that if we were to try a find a car in this scene, we would only be looking in the lower half of the image, as we don't expect the cars to be up in the air.

We also note that even though we can't see an explicit horizon line in this image, the edge of the ground plane and the slanting lines at top and bottom of the walls give us a clear cue about where the horizon line lies. So in some sense it is a cyclic process where we use elements of the scene to calibrate our viewing system and then use that knowledge to interpret the same scene.

Object Class	Plane Type(s)	Vertical Location	Height Range	Width Range	Aspect Range
Building	Vertical	0m	2m-10m		0.25 - 4
Car	Vertical	0.25m – 1m	0.25m-0.4m	1.25m-2m	0.2 - 0.4
Box	Vertical	0m	0.5m – 1m		0.5 – 1.5
Street Sign	Vertical	4m-6m	0.2m-0.25m		0.2 – 0.3
Ground	Horizontal	0m			

Table 1 - Objects in our object database

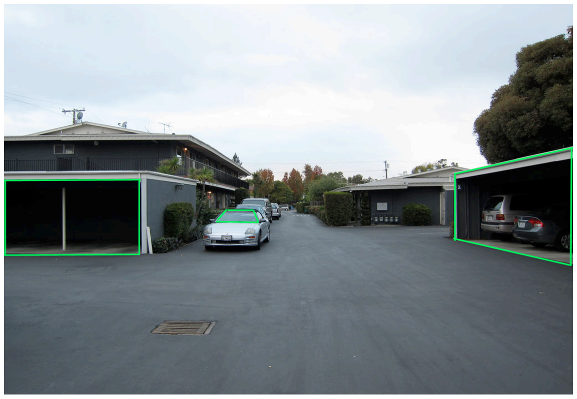


Figure 2 – Algorithm works on planes like the ones marked here with green boundaries

For next image (street sign) we have none of the cues like previous cases but we derive the sense of scale from the object itself because we know what we expect the size of this object to be.

As a last example lets look at a picture with so called miniature-model effect – i.e. a picture where we are tricked into believing that the picture is depicting miniature models of real-life objects, when in reality the picture has full-size real objects. The blurred areas trick us into believing that those areas are out-of-focus, as we would expect when we are focusing close. Moreover the downward angle is consistent with our belief where we expect to typically have only small objects in our field of view when we are looking downwards.

As last example shows, human visual system does not always get it right. However it works extremely well in so many cases because typically objects in the world remain consistent with how we expect objects in the world to be.

The motivation for this paper is to come up with an algorithm modeled around the same principles combining some knowledge about our visual system with an understanding of scene geometry and object knowledge to reconstruct 3D information for a 2D image. Of course it is beyond the scope of this project to have an object database anywhere as comprehensive and complex as employed by humans so our little universe consists of a very small set of objects which are characterized as consisting of horizontal or vertical 'planes' of various sizes. Figure 2 shows some planes associated with objects in test image.

The task of the algorithm is to match each 'plane' in the image to best possible candidate object that exists in our database of objects. These objects include "Buildings", "Cars", Boxes", "Street Signs" and "Ground". No explicit features such as SIFT etc are used for object identification and instead the matching is purely based on the "plausibility" that a plane could belong to a particular object class. This plausibility is derived from the constraints that we have on the size and location of each object class in the 3D world, as well as the information that we have about the visual system. In a nutshell the algorithm tries to find object-plain pairs which simultaneously satisfy the geometric constraints placed on the size and location of the plane as well as the size and location constraints on the object as specified by database.

The information about the visual system that is input to this system includes the following:

1. Approximate height
2. Approximate field of view
3. Range of possible vertical viewing angles.

The approximate values of first two parameters should be trivially obtainable in most visual systems. For the viewing angle a broad range can be specified as an input (such as +-20 degrees) and the algorithm tries to refine it as much as possible using feedback from the object matching process. So the algorithm again tries to mimic human visual system which can use cues from the scene to calibrate its understanding of current orientation.

The major tasks in the project then include:

1. Segmenting the image to extract the planes
2. First pass plane to object matching
3. Refinement of viewing angle using matching feedback and final match using this value.

Section 3 will discussed each of these in detail followed by evaluation in section 4. But before that next section looks at background work in related areas.

2. Related work

There has been a number of works in past which dealt the problem of recovering 3D information from a single 2D images. Work of Criminisi, et al. [1] for example looks for vanishing lines and vanishing points in a scene and uses that to measure distances between parallel image planes as well as distance between objects (up to a common scale factor), which lie on a given plane. They also use this information to estimate camera positions and orientation. Overall the work is based purely on perspective geometry of parallel lines and does not use any object-level knowledge. Our work also uses similar information if available. However we do not rely on the existence of vanishing lines and points and if they do exist, we primarily use them as one of many mechanisms built in the system to detect viewing direction with respect to ground plane.

Work by Hoiem et al.[4][5] is based on very similar considerations to what drove our work. Even though our work was initially conceived independently of work done by Hoiem et. Al, the similarities in underlying thinking nevertheless are very clear. Like us they strive to use knowledge about scene geometry to constrain where various subjects can live and in parallel use information about objects to constrain possible viewing direction for the camera. However their approach is fairly different from ours. Their algorithm uses a training based statistical approach to match the scene to some known templates of scene structures in the process identifying important scene elements like ground and sky. This then gives approximate information about 3D structure of the scene, which then works as a prior for feature-based detection of a known object in the 2D image. However unlike their work we actually do a full projecting of each image plane into the 3D world without assuming any specific structure to the scene and then use constraints on objects to find object matches in 3D and also to refine our knowledge of the 3D world. We do feel however that elements of their work can be combined with ours to create a more powerful system.

Other works such as that by Huang and Cowan [6], and Delage et al.[2], target scenes with known structure (specifically indoor scenes with walls, floor and ceilings in a well-composed structure).

Another interesting related work is done by sexana et al. [7] which is also based on the idea of using cues from the scene to understand its 3D structure in a manner analogous to how humans use such cues. However their approach is based on a statistical learning system which can 'learn' depth cues by using a set of test images and as such they don't make any explicit use of object level knowledge.

3. Approach

We first look at overall algorithm and then discus each component in detail. The inputs to the system include:

Input image

Visual system parameters including an estimate of the viewing direction

Object database

The output consists of the following:

Identification of any objects from the DB which exist in the scene

3D Location and orientation of the object

Dimensions of the object in the 3D world

A refined estimate of the viewing direction

The matching algorithm consists of the following steps:

Identify major edges and lines in the image

Segment the image

Find horizontal and vertical planes using 1 and 2

Find constrains on possible 3D locations of these planes in the 3D world

Attempt to match these planes with DB objects across the range of possible viewing directions

Use the feedback from matching process to refine the estimate of viewing direction

Find the final plane-to-object matching based on our best guess of the viewing direction

3.1. Objects Database

As discussed earlier, all objects in our world are characterized in terms of vertical and horizontal planes, and constraints on their potential location and size in the 3D world. Table 1 lists all 5 objects and their attributes.

The vertical location specifies the possible vertical position of the object in the 3D world. For example a building is not expected to exist up in the air so its only possible vertical location is at a height of. A street sign on the other hand can only be found at a height of 4-6m above ground as per our database. The height column gives the expected vertical dimensions of the objects. Fir example a street sign can be 0.2-0.3m high. As we can see most parameters span a range of possible values. Note that all of these values are 'loose' and are just based on 'eyeballing' by the author.

Lets also consider the object car further. While a car is not a planar object, for the purpose of our work we can characterize it in terms of its components such as windshield, which can be considered as planes.

Ground is the special object in the sense that it is the only object expected to be a horizontal plane. Note that this somewhat preferential treatment for the ground plane is purely for simplicity and does not significantly impact the matching process - we don't use the knowledge about ground plane as a factor in locating other objects.

3.2. Visual system input parameters

The visual parameters used as input include:

1. Approximate height (defaults to 5.5ft)
2. A measure of field of view. Currently this is specified in terms of camera sensor size and lens focal length. Can be replaced by FOV measured in degrees and does not need to be very exact.
3. An estimate of vertical viewing direction specified as a range. E.g a range of 0-15 degrees would specify that the camera / viewer is looking up by an angle somewhere in the range of 0-15 degrees.

Nest we look at each of the algorithmic steps.

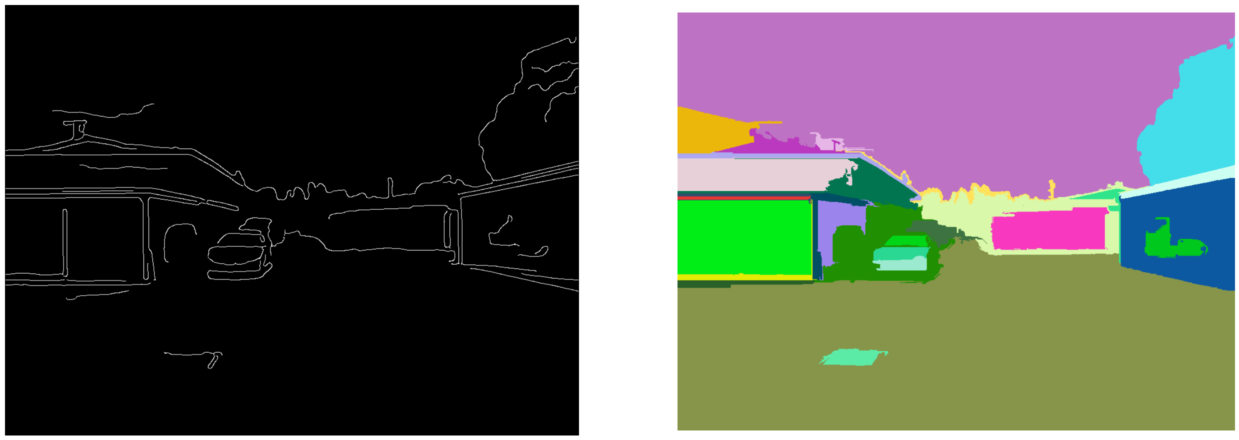


Figure 3 – Edges (after pros processing) & segments

3.3. Detecting major edges

The first step in the algorithm is to detect major edges in the image. The edge information is used in two ways. First the slanted and vertical lines in the image are used to form an estimate of the viewing angle. Second the edge information is used in combination with image segmentation output to determine plane boundaries.

For this purpose we use a canny edge detector built into matlab with some enhancements. Canny is a good detector for our purposes since it implements edge linking to create well-connected edge lines, which is helpful when trying to find object boundaries. Also it allows control over the scale at which we want to detect the edges which is great since we only want to detect major edges along object boundaries and ignore the spurious edges within the objects. However Canny considers edge strength as the only criterion when deciding which edges to keep at a given scale. This gives rise to a large number of isolated small edge segments, which do not correspond to object boundaries and instead only represent isolated edges within objects. Increasing the threshold removes these but also eliminates some true object boundaries. To fix that we used a relatively low threshold to preserve all major edges and then added a post-processing step which iterates over all edges in the output and removes all edges smaller than a threshold (75 pixels).

Figure 3 shows an example output, which shows that this approach works very well. In fact we observed that once this post-processing step was added, the final output did not depend too heavily on the exact threshold used during the canny stage. The supplemental section at the end includes a couple more example outputs before and after the post-processing step.

3.3.1 Isolating Straight Edges

We use edges primarily to find lines leading to vanishing point and to help find plane boundaries, both of which only rely on straight edges. So we add a second post-processing step, which looks at all edge segments remaining after refinement and retains only those edges, which correspond to straight-line segments. In the canny output a single stretch of connected edge pixels might cover multiple straight and curved lines segments. Thus we need to run a window over each edge line looking for local straight segments and cataloging the whole set. During this process we also bin these straight edges into “horizontal”, “vertical” and “slanted” edges.

3.4. Image segmentation

The next step is to perform image segmentation. The particular scheme that we used is a graph based scheme the developed by Felzenszwalb and Huttenlocher [3]. The algorithm can automatically decide the number of segments and also considers color and texture instead of relying only on intensity. Thus it can effectively distinguish the variability within a ‘textured’ object like a tree from the variability between two different objects such as a tree versus sky.

We used the freely available public domain code for this algorithm. The algorithm has only 3 parameters – we used a blur sigma of 1.0, left the threshold at a default value of 500 but set the minimum component size to a high value of 1200 to again eliminate small segments and retain only large object level segments. By trying it across different images, this consistently gave good results.

Figure 3 shows the output from this step. The overall segmentation quality is very good. However we note that the algorithm often creates a number of thin segments on the boundary between two objects. This is apparently because the boundary region between some objects is a blurred combination of both objects and the algorithm is unable to associate this to any one of the two objects. Like

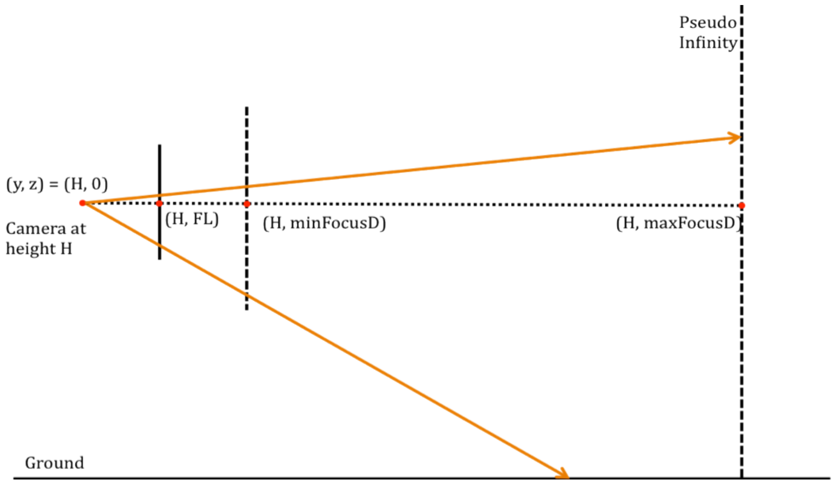


Figure 4 – Deriving constraints on image points

edge detection we added a post-processing step to try and remove any thin segments, distributing the pixels to neighboring objects on both sides. Nevertheless we did not invest a lot of time on this and thus the resulting algorithm is not very effective at removing all thin segments. Mostly this is not a problem because these segments are not expected to match any objects in our database and get eliminated during matching step. Nevertheless we still 'manually' remove some of these segments by passing an 'segments-to-ignore' list to the system to simplify matters. Given more time it should be fairly easy to automatically eliminate most such segments.

3.5. Extracting planes

This step now uses the segmentation and edge data to extract planes in the scene. We assume that the scene can contain three types of planes – horizontal planes, vertical planes that are front-facing and vertical planes which are side facing. Looking back at picture 2 the large plane on the right marked by a green boundary is a side facing vertical plane, the plane on left again marked by a green boundary is a front-facing vertical plane and the ground plane is a horizontal plane. Note that the distinction between front-facing versus side-facing vertical planes is arbitrary and only a result of how we process these planes. The object descriptions of Table 1 do not distinguish between these two vertical plane types.

We use relatively simple criteria to categorize each segment as either one of these three plane types or as type "no-plane". First we isolate the boundary points of each segment and then fit horizontal, vertical and slanted lines through these segment boundaries. For slanted lines we use the set of slanted lines that we have already identified during edge detection as the candidates. In some cases not all edges of a plane are visible so we require that each segment is bounded by at least two straight lines to declare it as a plane. For categorization we use a simple system where each segment, which is bounded by at least one vertical, and one horizontal segment is considered a front-facing vertical segment. A plane bounded by at least one vertical and one slanted line is considered a side-facing vertical plane, and a plane having at least one slanted and one horizontal boundary line is considered a horizontal plane. This is not comprehensive and ignores some cases but can be improved easily.

3.6. Plane-to-Object Matching

Once planes have been isolated we come to the core task of matching these to objects in the database. The task consists of two parts – first we use our knowledge about visual system parameters and location of planes in the image to derive constraints on possible location, orientations and dimensions of each plane in the 3D world. Then the matching tasks become finding for each plane one or more object-plane pairs which simultaneously satisfy these geometric constraints placed on the planes as well as the size and location constraints on the object as specified by the object database of Table 1

For this discussion we initially assume that the viewing angle is known (within a small error margin). Later we will extend this to the case when the knowledge about the viewing angle is not precise.

3.6.1 Finding 3D constraints on the planes

To derive constraints in the planes, lets first study the constraints placed on a single image point as depicted in figure 4. For now we assume a 1D vertical imaging sensor and vertical viewing direction of 0 degrees. Our camera's 'eye point' sits at a height of H and at a depth of 0 (y = H, z = 0). The imaging sensor is approximately a focal length away at (H, FL) (actual distance is not quite equal to focal length unless we are focused infinity but we are looking for approximate measures only). We consider the constraints on the actual 3D locations of two image points on the sensor. The direction vector (indicated in orange) from the camera to the lower image point represents all the 3D world locations where that point could have existed in the real world. Assuming that the lower image point is y units below the sensor center, we can find that direction vector as:

$$\vec{d} = \frac{(-y, FL)}{\|(-y, FL)\|}$$

(Of course the image points are defined in terms of pixels but given the sensor size, it is trivial to convert from pixel height to sensor height). Given this direction vector, we now impose further constraints on the point. The first constraint comes from the assumption that every point in the 3D world resides at or above ground level – i.e. we don't have object at lower than ground level. This gives us a constraint on the points minimum height in the world and the corresponding depth can be computed by first computing the scaled vector $\vec{d}_{ground}$:

$$\vec{d}_{ground} = -\frac{\vec{d}}{d_y} \bullet H$$

The z-component of $\vec{d}_{ground}$ then gives the maximum depth of the point in the scene:

$$z_{\max} = d_{ground_z}$$

To find the other extreme we look at the closest point to the camera where we expect to find an object. We call this point as closest focusing distance (and use a somewhat arbitrary value of 0.5m for it), though in reality it is just a proximity constraint on how close the object can be to the viewer and does not have to depend on camera system's minimum focusing distance. Since here we already have depth so we can find the height as:

$$\vec{d}_{close} = \frac{\vec{d}}{d_z} \bullet \min FocusD$$

$$y_{\max} = \vec{d}_{close_y}$$

For the second image point on top, we use a similar process to find the height at closest depth point, except that that image point is now the minimum-y point instead of maximum-y point. Since the upward going vector is not bounded by ground plane so the maximum height constraint has to come from the farthest possible depth in the scene. Of course the largest possible depth is infinity but for our purposes it is sufficient to use a moderately large value of depth such as 1000m to represent infinity. That then gives the maximum height for the upper image point as:

$$\vec{d}_{far} = \frac{\vec{d}_{upper}}{d_{upper_z}} \bullet \max FocusD$$

$$y_{\max} = \vec{d}_{far_y}$$

Now we incorporate the 2D sensor and the viewing angle. To incorporate a viewing angle for the camera we note that viewing up / down by an angle θ is equivalent to the camera sensor rotating up / down by an angle θ around the –axis (with center of rotation at the camera point (0, H)). The points on the un-rotated sensor have 3D co-ordinates (x, y, FL) with respect to the camera location and after rotation each point on the sensor moves to a new location R. (x, y, FL) where R is the rotation matrix describing rotation around the x-axis by an angle θ. Thus the direction vector for any point (x, y) on the rotated sensor can be calculated as:

$$\vec{d}_{\theta} = \frac{R(x, y, FL)}{\|(x, y, FL)\|}$$

Where x, y represent distances from the sensor center.

With that setup, give a point in the image and a viewing angle θ for the camera we can compute the constraints ($y_{\min}$, $y_{\max}$) corresponding to the minimum and maximum possible height of that point in the 3D world. And we can also derive the corresponding z and x locations. Alternatively we can look at it as constraints on minimum and maximum depth along with a direction vector which can give us the y and x co-ordinates at each depth.

Now we use this to find constraints on the vertical planes in the scene. Looking again at figure 4, lets now assume that the two image points are top and bottom points of a front-facing vertical plane. In that scenario the top and bottom points would be at the same depth in the 3D world. Thus in this particular example the overall depth of the plane would then be constrained by the lower edge of the plane which cannot go below ground plane as per our assumption. And the minimum depth constraint simply comes form the *minFocusDistance* parameter as before. From this constraint on the minimum and maximum depth of the plane we can now derive constraints on vertical position, height, and width of the plane. Note that we define the 'vertical position' as the y-coordinate of the lower edge of the plane and height as the vertical dimension of the plane (i.e. difference between the y-coordinated of its upper and lower edge). For example a plane, which extends from 3m to 7m above the ground level has a vertical position of 3m and height of 4m.

To understand what does this really mean in terms of our ability to identify the plane as an object lets consider the front-facing vertical plane on the left in figure 2 (marked by a green boundary). Assuming that the camera was pointed up by 1 degree, we learn that this plane be

	Vertical Location	Height	Width	Aspect Ratio	
Plane	0 - 1.65m	4cm – 3.2m		0.32	

Table 2 – Constraints on the 3D presence of an image plane

Matching	Common Vertical Locations?	Common Heights?	Common Widths?	Match?
Plane- Street Sign	None			NO
Plane - Car	0.25m – 1m	None		NO
Plane - Box	0m	None		NO
Plane - Building	0m	3.2m	0.32	YES

Table 3 – Matching the plane to objects

anywhere from 0.5m to 36m away from the camera. We also learn that at 0.5m away, it would represent a tiny object, which is only around 4cm in its vertical size and is located up in the air at a vertical position of around 1.65m. We can now imagine looking up the object database to see if we know any object class in our little universe, which can be as small as 4cm and can be found at a height of 1.5m in the air.

The procedure for measuring the dimensions and location of "side-facing" vertical planes like the right most slanted plane in figure 2 (marked by a green boundary) is conceptually similar but involves a bit more math due to the added complexity that for these planes different points on the plane are at a different depth. The first step in that case is to find the plane orientation i.e. the angle of the plane with respect to the image plane, which can be found out by using the slanted lines on top or bottom of the plane by using a bit more 3D geometry. Once that is know, we can then find the constraints on height, vertical location and width in a manner similar to the front-facing. We haven't included the equations of that case for the sake of brevity and instead refer the interested reader to the code.

We note that the plane has are two slanted bounding lines available which can both be independently used to find an estimate for plane orientation and if there is disagreement in both measurements then that tells us that our current assumed viewing direction is probably incorrect – a fact that we will use later when we try to refine viewing direction estimates.

3.6.2 Matching

Now we have all the pieces in place to try and match each plane to one or more of the object classes. Lets again look at the example of front-facing plane from figure 2. Table 2 specifies the constraints that the system has computed for this plane assuming a vertical viewing angle of 1 degree. Now the system takes the plane and for each object in the DB tests the hypothesis that this plane can be an object of that class. The process to test the hypothesis involves testing a constraint on the plane against the corresponding constraint on the object and finding if there exists a region of overlap. If an overlap exists then we tighten the constraints to match the overlap region and test next constraint. For example lets look at the test for the hypothesis that our test plane is from a car:

1. Do the possible vertical locations for the plane overlap the possible vertical locations for a car plane? Yes – the common range is 0.25m-1m.
2. What is the range of heights the plane will have at vertical locations 0.25m – 1m? – [1.25m to 2.7m].
3. Does this range overlap with possible heights for a car plane. NO – Match failed

Table 3 shows the results of matching the plane to each object class. The street sign does not work because there is no overlap in possible vertical locations. The Box has a possible match at vertical location 0. However for this plane to be at vertical location 0, it has to be 3.2m high, which means it cannot be a Box. The building object on the other hand meets all criteria and thus our best guess for this plane is that it belongs to a building.

We must emphasize here that looking at individual constraints a match was possible with many objects, however the hypothesis failed in most cases because it was not possible to find a set of parameters where all three constraints on vertical location, height and aspect ratio were matched. So at the end of the process we not only know that the plane is a building plane, we have narrowed down the large range of its possible depths and heights to exactly one possible height, depth and width using the object-level knowledge.

As another example let's look at the plane 10 from figure 5. Based on the constraints computed on its height and vertical locations, the system concludes that it can either be a box lying at ground with a height of around 0.95m, or it can be a plane from a car which is around 0.5m above the ground and has a height of 0.38 – 0.4m. However since it is a long and thin plane with an aspect ratio of around 0.22 so we reject the hypothesis that it is a box (which in our world have a minimum aspect ratio of 0.5) and this conclude that it is a car plane which is around 18-19m away and has a vertical size of about 0.38-0.4m.

3.7. Matching Summary

To summarize this section the whole matching mechanism is based on the belief that while a plane might

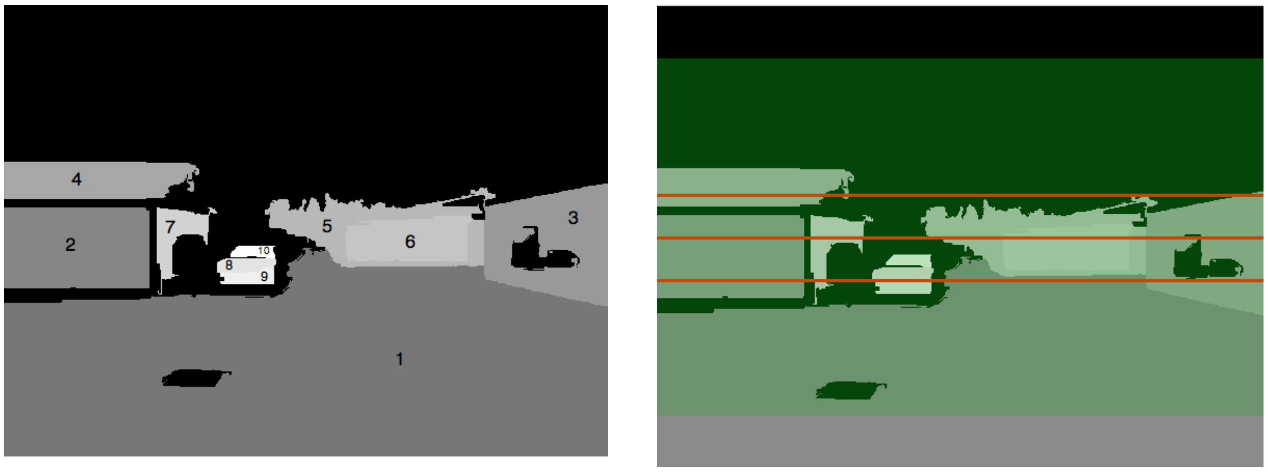


Figure 5 – (a) Segments identified as planes, (b) Horizon lines for various assumptions on viewing angle

have a large range of possible locations in the 3D world based on the geometric constraints alone, only a small subset of those locations would 'make sense' given what we know about the objects in the world. The algorithm thus provides one way to combine the geometric information from the image and viewing system with object level knowledge where each alone would not have been enough to argue about the objects in the image.

3.8. Refining the viewing angle

Previous sections assumed that the viewing angle of the camera is known with some degree of accuracy. However the second part of the system deals with performing the matching when the viewing angle is not precisely known. The algorithm accepts the viewing angle with a degree of uncertainty – e.g we can say that the viewing angle can be anywhere from -10 to +10 degrees or 15 to 25 degrees etc. The algorithm then tries to refine that estimate along with the matching process.

Initially we assume that all viewing angles are equally probable and we associate the same "confidence" to each angle, There are then 4 sources of feedback, which help in refining the confidence to find the best estimate of viewing angle. First, we believe that an incorrect viewing angle estimate will often result in such constraints on the image planes which do no make sense in the context of the objects in our world, and thus will result in low matches. In other words "the image will only make sense when we interpret it with a reasonable estimate of the viewing angle". So the first form of feedback is to just try all viewing angles (with a steps size of 0.5 degrees) and compute a confidence measure based on how many planes are successfully matched to unique objects in the database and how many are unmatched or are ambiguous.

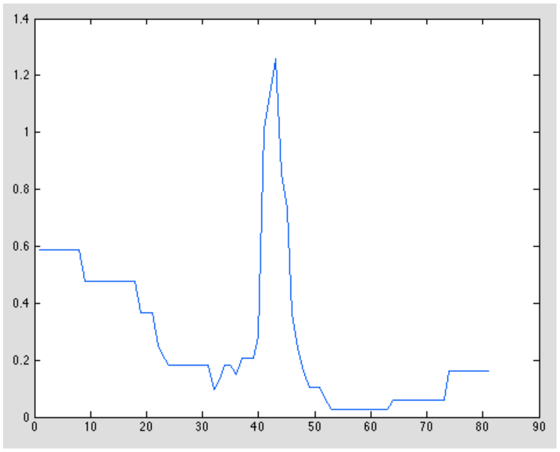
The second source of feedback comes from using the slanted lines, which we isolated during the edge detection stage, and finding their intersection to locate possible vanishing points for parallel lines in the world. A simple huff transform mechanism is used and the height of any detected vanishing points is then used to find the corresponding viewing angle. The output of this process is also a confidence measure if say two possible vanishing points are detected at slightly different heights then it gives confidence that the horizon line lies somewhere in that range and the range of corresponding viewing angles then get a vote of confidence. The supplemental section at the end gives some example outputs.

The third source of feedback also relies on vanishing points but in this case it is applied after we have identified vertical planes like plane 3 in figure 5. Here we know with greater certainty that the top and bottom lines of this plane should intersect at a vanishing point unlike the previous case where we did not know which of the slanted edges are true parallel lines and which are just angled edges.

The 4th feedback comes from any horizontal plane, which has been identified as possibly being the ground plane. The ground plane gives the constraint that the horizon line should be above the edge of ground plane in the image, which than limits the possible range of viewing angles.

We note that none of these measures is considered as a "proof" for true location of the horizon line / viewing angle. Instead each of these just provides a vote / confidence measure for one or more viewing angles and we then combine the votes from each source to come up with the best guess.

At the last step we then use this best guess viewing angle and find the object-plane matching suggested by this viewing angle. This is then the final refined matching.



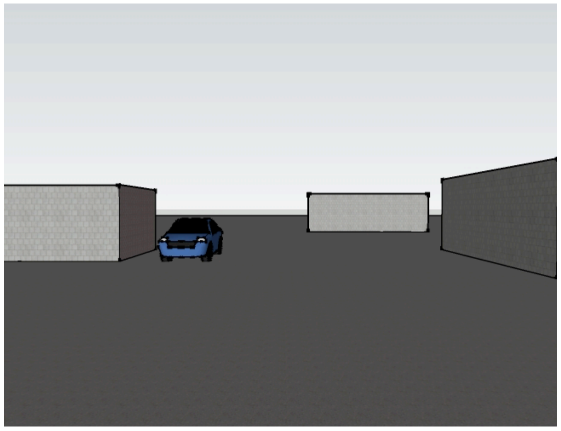


Figure 6 (a) Confidence for various viewing angles from -20 to 20 (b) 3D Model reconstructed from the output

4. Evaluation

For evaluation we use the test image in figure 2, which we have already been using throughout the paper to discuss various parts of the algorithm. We assess 4 aspects of the performance for this image including correct detection of planes, object matching when a correct viewing angle is specified, object matching when an incorrect assumption is used for the viewing angle and finally ability to automatically determine the correct viewing angle and perform matching when a broad range of viewing angles are specified.

We have already seen in figure 3 all the segments identified by the segmentation algorithm. Table shows the number of segments as they get filtered at each step. We start with a total of 29 segments identified by the segmentation routine. After removing the thin segments by a combination of automatic and manual means, we are left with 20 segments. The plane finding algorithm then discards 10 more which it cannot identify as being a plane. And Figure 5 shows the final 10 segments, which remain at this stage.

Our simple criteria for finding plane types is unable to confidently label plane '4' in the figure 5 as a vertical plane as it does not have any vertical lines bounding it and it gets labeled as a "vertical or horizontal plan". It can eventually be identified as a vertical plane when object criteria are applied (there is no know object in our world which is horizontal and sits above the horizon line). However our current implementation does not implement this and thus this plane gets eliminated during matching. Plane 5 also gets discarded and we are left with a total of 7 vertical planes and 1 horizontal plane.

Figure 5(b) shows the position of horizon lines corresponding to various estimates of viewing angles. The red lines correspond to -5, 0 and 5 degrees (top line = -5 degrees) and the green region shows a range of -20 to +2-degrees which pretty much means that the horizon line can be anywhere in the image.

Table 4 shows the estimated object types and dimensions for each of the 8 planes when a viewing angle estimate of 0 degrees is used. We note that the algorithm is able to find exact objects matches for each plane. Moreover for all "building objects" it is able to narrow down the height and depth range to exactly one possible value due to the constraint that a building plane can only be at a vertical location of 0m. For car object there are multiple possible locations and sizes of the plane consistent with the belief that it is a car plane. However even then it is able to narrow down the possible configurations to a relatively small range. We also note that the assumed viewing angle value of 0m is actually slightly off from the correct viewing angle.

Tables 5 and 6 show what happens when we assume viewing angles of -5 and 5 degrees. With a -5 degree assumption most vertical planes can't find a suitable match. Just one plane fits a possible configuration where it can be a car plane (an incorrect match). With a 5 degree assumption a couple of planes find a match as a building (correct matches) – however most planes again don't find any possible configuration in which they can be one of the objects in the database.

These results are fairly interesting and reinforce the two key ideas (i) that object-level constraints are very powerful in narrowing down the large range of possible 3D configurations to a narrow range, and (ii) that the objects in a scene just don't make sense when we try to interpret them with an incorrect assumptions on viewing parameters. In other words the object knowledge guides us towards a correct interpretation even when we don't accurately know the viewing parameters.

	Object	Depth	Height
Plane 1	Ground	-	-
Plane 2	Building	14m	2.32m
Plane 3	Building	17m	2.6m
Plane 6	Building	26m	2.3m
Plane 7	Building	12.5m	2.3m
Plane 8	Car	10-14m	0.28-0.4m
Plane 9	Car	11-13m	0.33-0.4m
Plane 10	Car	13-19m	0.28-0.4m

Table 4 – Matches assuming 0 degree angle

	Object	Depth	Height
Plane 1	Ground	-	-
Plane 6	Car	4.6m	0.4m

Table 5 – Matches assuming 5 degree angle

	Object	Depth	Height
Plane 1	Ground	-	-
Plane 2	Building	52mm	8.7m
Plane 7	Building	36.8m	6.64m

Table 6 – Matches assuming 5 degree angle

And now we come to the final stage where we give the system a broad range of -20 to 20 degrees as the possible viewing angle. As figure 5(b) shows this means that the horizon line can be pretty much anywhere in the image. Figure 6(a) shows the confidence that we get for various viewing angles using a combination of criteria discussed earlier. There is a clear trend which shows that only a narrow range of viewing angle make most sense. The final estimate of the viewing angle comes out to be 1 degree, which is consistent with a manual analysis of the image. Figure 6(b) shows the 3D model crated from parameters suggested by the final refined match at 1 degree (with some manual fixes).

5. Conclusion

This work has shown one possible approach to combine object level knowledge with geometric information to recover the information about the original 3D scene. From a purely geometric perspective a given 2D image can be the result of infinitely many 3D configurations in the real world. However the key point is that only a few of these configurations make sense in the context of the objects in the world, as we know them. And our results show that this knowledge is a powerful tool to incorporate when interpreting 3D scene as it greatly filters down the space of possible 3D interpretations of the image. We have also seen that an image only "makes sense" when we interpret it using the right parameters. Humans all the time use familiar objects configurations in the world to calibrate their visual system. A similar approach can be adopted in the compute vision task of understanding the 3D scene, where the information flows both ways i.e. geometric information constrains possible object matches and object matches constrain possible geometric configurations and the final solution is the best possible reconciliation of both.

While a significant part of this work (and the write-up) was focused on image segmentation and extraction of image plane, in retrospect we would have chosen a different approach. We feel that we should have focused more on trying our various configurations of objects in the images to see how well our framework works across various scenarios. The image segmentation task in the end was just means to an end, and if we do this again we would instead use 3d scenes generated from 3d modeling software such that the segmentation task is straightforward. Or alternatively we will just manual mark the planes in the image and let the algorithm concentrate on finding plausible object matches and viewing system parameters.

Other future work could include incorporating the coarse focus distance information as suggested in the original proposal.

Finally, this work did not make use of any knowledge about object features as used in typical object recognition tasks. In fact all objects in our world looked identical (planes) from a features perspective, which is essentially a harder matching task. But at the same time our world had only 5 possible objects. In a practical system, we would expect the 'object database' to contain features in addition to height and location etc, which can then be a really powerful framework for matching.

References

[1] A. Criminisi, I. Reid and A. Zisserman. Single View Metrology, International Journal of Computer Vision (IJCV), 2000.

[2] Erick Delage, Honglak Lee, and Andrew Y. Ng. Automatic Single-Image 3d Reconstructions of Indoor Manhattan World Scenes, In ISRR, 2005.

[3] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. IJCV, 59, 2004.

[4] D. Hoiem, A.A. Efros, , M. Hebert, Putting Objects in Perspective, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2006

[5] D. Hoiem, A. A. Efros, and M. Hebert. Geometric context from a single image. In Proc. ICCV, 2005.

[6] Jingyuan Huang, Bill Cowan. Simple 3D Reconstruction of Single Indoor Image with Perspective Cues. CRV'2009. pp.140~147

[7] Ashutosh Saxena, Min Sun, Andrew Y. Ng, Learning 3-D Scene Structure from a Single Still Image, , In ICCV workshop on 3D Representation for Recognition (3dRR-07), 2007