Correcting User Guided Image Segmentation

Garrett Bernstein (gsb29)
Karen Ho (ksh33)
Advanced Machine Learning: CS 6780

Abstract

We tackle the problem of segmenting an image into planes given user input. Using a supervised learning approach, we develop an algorithm that adjusts the user input towards true plane segmentations, under the restriction that images have vertical boundary lines. We begin by collecting images for the training set consisting of hallways, buildings, stairs, rooms, etc. Users are then asked to draw dots on the image to indicate the locations of the intersection between neighboring planes. We then apply our supervised learning algorithm to predict the true location of the planes. Thus, we propose a method to infer the boundaries of planes from a single image given user input.

1. Introduction

There have been considerable efforts recently to generate realistic 3D models from single images. One approach is to create a surface layout for a single image that labels geometric classes of the ground and the sky as well as the relative position of additional objects [1]. Another common approach is to determine boundary lines using simple geometric constraints present in indoor and outdoor architecture structures. Using the user input of labeled parallel or perpendicular edges, a partial 3D model can be constructed after determining the vanishing points and camera calibration [2]. This algorithm, however, requires the user to define and label the exact edges in order to generate a suitable 3D model and furthermore, doesn't take advantage of learning techniques to train the algorithm. Our proposed algorithm intends to first generate thresholds that can fix slightly skewed boundaries drawn by the user and then, given an entirely new environment, use these thresholds to output more accurate divisional boundary lines.

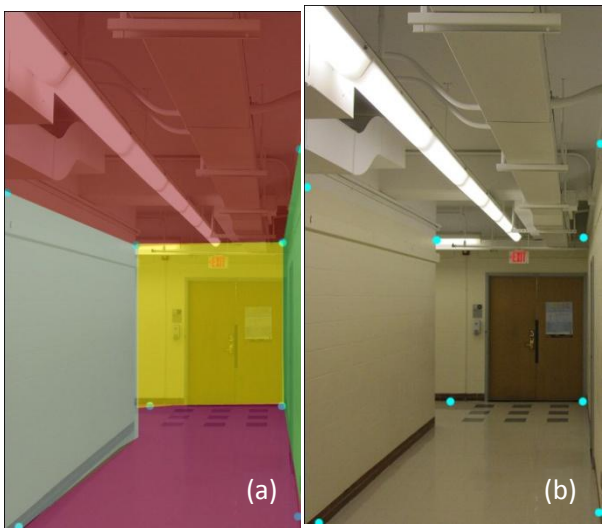

Figure 1a. Image of a hallway divided up into a series of planes that make up a polygon mesh that can be used to construct a 3D model. Figure 1b. Image of the same hallway with user input of the plane vertices.

Obtaining a 3D model from images is a difficult task in computer vision. Humans can easily envision the 3D structure from a single image using a variety of texture, geometric, color and contextual cues. One of the basic ways to generate a 3D model from a single image is to first break up the image into several planes. For example, in Figure 1a the color overlay illustrates the boundaries of planes that can be used to generate a 3D model. As shown, the walls, ceiling, and ground make up the planes of the image and this information can be simplified to just the corners of the planes. Our goal is to take user input in the form of dots that signify the vertices of planes and to adjust these vertices as necessary.

Users are not perfect at drawing points that define the boundaries between different planes such as in Figure 1b. The user may not be able to accurately determine the plane boundaries, whether from an occluding object or simply lack of skill. Another source of error is just not inputting the exact intended location on the image. Our learning algorithm attempts to correct the input by modeling the tendency of users' to input inaccurate points.

Our approach assumes that an image consists of several planes and that the ground is a straight horizontal line (ie. the camera was not tilted sideways.) With the second assumption we can assert one important dependency in our model; that the sides of planes will be vertical. Given this relation, our model will attempt to correct vertices in the image that have 'close' x-coordinates by bringing the x-values closer together for a more vertical line. How 'close' two points have to be will be determined from our training data and will act as a threshold as to when it is necessary to correct the locations of the vertices.

2. Prior Work

There have been many recent investigations of producing a 3D model from a single image. Some investigate texture distortion to infer the shape and orientation of objects [3] while others attempt deduce depth from the image from using relative sizes of objects [4] to recognizing properties of the structures in the image [5].

In Saxena, Chung, and Ng [6] and [7] produces depth maps using Markov Random Fields. Their approach is to use MRF to train two models (a jointly Gaussian MRF and Laplacian model) using a large training set of images and corresponding true depth maps. Our model takes on a similar approach to their model in how they capture depth relationships between neighboring patches and how we will attempt to simulate the relationship between neighboring vertices of planes. However, our problem focuses more on correcting user input as oppose to using features calculated inside the image. This presents an interesting problem not investigated by many papers: How accurate are users in isolating different planes in an image? While many can argue that it is easy to create a mental 3D model of the image, it is not as easy to specifically mark the boundaries of planes (e.g. an object occluding part of a boundary). Thus, we will investigate using supervised learning techniques how much our model needs to adjust user inputs to acquire the true vertices o the image.

3. Probabilistic Model

There can be a wide variety of shapes and orientations for the image planes for different environments. Thus, the model must be able to globally reason the spatial structure of the image by modeling the relationship between vertices that define the boundaries of the plane, as opposed to defining set features over images from many environments. It can be inferred that the vertices of an image are not entirely independent of one another. Vertices that belong to the same plane will be highly correlated. For example, given the end of a hallway, the back wall will be a rectangular plane. Since we have restricted the scope of our problem to having the ground be horizontal, then it is much more likely that the x-coordinates of the plane's left boundary can construct a straight vertical line. The same applies to the x-coordinates of the plane's right boundary.

In fact, the majority of planes in images with a horizontal ground will have vertical edges. We cannot infer the same relationship with the y-coordinates of the vertices because of the geometric spatial properties of 3D objects, i.e. vanishing points mean the top and bottom boundary lines of planes will not necessarily be horizontal. Going back to the example of the hallway in Figure 1, there will also be planes of side-walls, which will have different y-coordinates to represent the plane tilted and sticking out in the 2D space.

The second conjecture the model can make relates to the expectation that, while not perfect, the user input will be close to the actual divisional boundaries. The model should be able to make a prediction that is close to the user input.

3.1 Gaussian Model

We employ a Gaussian Probabilistic Model denoting the probability that y is the desired plane boundaries, given a user input d parameterized by variances σ_1, σ_2 (1).

$$P(y|d;\ \sigma_1,\sigma_2) = \prod_{i=1}^{M}\left(\frac{1}{\sqrt{2\pi}\sigma_1}\exp\left(-\frac{\|y_i - d_i\|_2^2}{2\sigma_1^2}\right)\prod_{j\in Ne(i)}^{M}\frac{1}{\sqrt{2\pi}\sigma_2}\exp\left(-\frac{(y_{ix} - y_{jx})^2}{(2\sigma_2^2)}\right)\right) \quad (1)$$

Where y_i is the i^{th} point of the output, d_i is the i^{th} point of the input, and y_{ix} is the x-component of that point. *Ne(i)* is the set of all neighbors of point *i*, where neighbor is defined as within *neighborThreshold* pixels horizontally.

We use a Gaussian distribution to model the relationship between the user data points and the actual vertices locations. Given a set of *M* user points, the first Gaussian term models the fact that we believe the user input d_i to be close to the actual vertex location y_i.

The second Gaussian term involves the x-coordinates of neighboring output vertices of the image. This models the fact that the planes in the images have vertical boundary lines. Thus is the user draws two points within *neighborThreshold* pixels of each other horizontally we can assume they were meant to be drawn perfectly vertically, which maximizes that probability term.

3.2 Training the Variance Terms

There are two parameters for our model: σ_1 and σ_2. The first variance term represents how far from the true boundary points the users tend to input their points. The second variance term represents how far from vertical the users tend to place neighboring points. These variance terms are computed from the training data to maximize the conditional log likelihood $P(y|d;\ \sigma_1,\sigma_2)$. Since the model is Gaussian, the maximum log likelihood can be solved in closed form.

Using (1) we solve for the closed form of σ_1 and σ_2 by setting the derivative of the log-likelihood in terms of each variance to zero and solving for each variance term respectively (2).

$$\sigma_1 = \sqrt{\frac{\sum_{i=1}^{M}\|y_i - d_i\|_2^2}{n}} \qquad \sigma_2 = \sqrt{\frac{\sum_{i=1}^{M}\sum_{j\in Ne(i)}^{M}(y_{ix} - y_{jx})^2}{n}} \quad (2)$$

3.3 Adjusting User Input

Once we have learned the variance terms we can then take a user's input on an image from a new environment and find the adjusted output y that maximizes the probability in (1). We take the log of (1) and then manipulate to achieve quadratic programming form (2) to plug into MATLAB's solver.

$$f(y) = \arg\max_y \frac{1}{2}y^T H y + c^T y \quad (3)$$

subject to $y_i \geq 0\ \forall i$. *H* and *c* are shown in (4).

$$H = \left(-\frac{I}{\sigma_1^2} + -\frac{2(Ne - NumNe)}{\sigma_2^2}\right) \quad (4)$$
$$c = \frac{d_x}{\sigma_1^2}$$

Ne is the neighborhood table where the element *(i,j)* is 1 if the point y_i is a neighbor of y_j, 0 otherwise. *NumNe* is a diagonal *M*x*M* matrix with values

representing the number of neighbors of each point y_i. See appendix for derivation of (4).

4. Experiments

4.1 Data Collection

We used a camera to collect images of 20 different environments around the campus of Cornell ranging from hallways to staircases. For each of these environments we took 5 different images. We labeled truth points by hand. We then acquired 5-7 users input per image and collected the locations of the plane vertices. This completes our data set of over 500 images, each containing a range of 6 to 15 points to indicate the vertices of the image. For a given image, we required the user inputs to be consistent. In other words, users must label the same number of vertices in order for the comparison between the user input and the actual image output to evaluate correctly. We therefore eliminated extraneous points that were not included in the actual image output. We used this image collection to train our data.

5. Results and Discussion

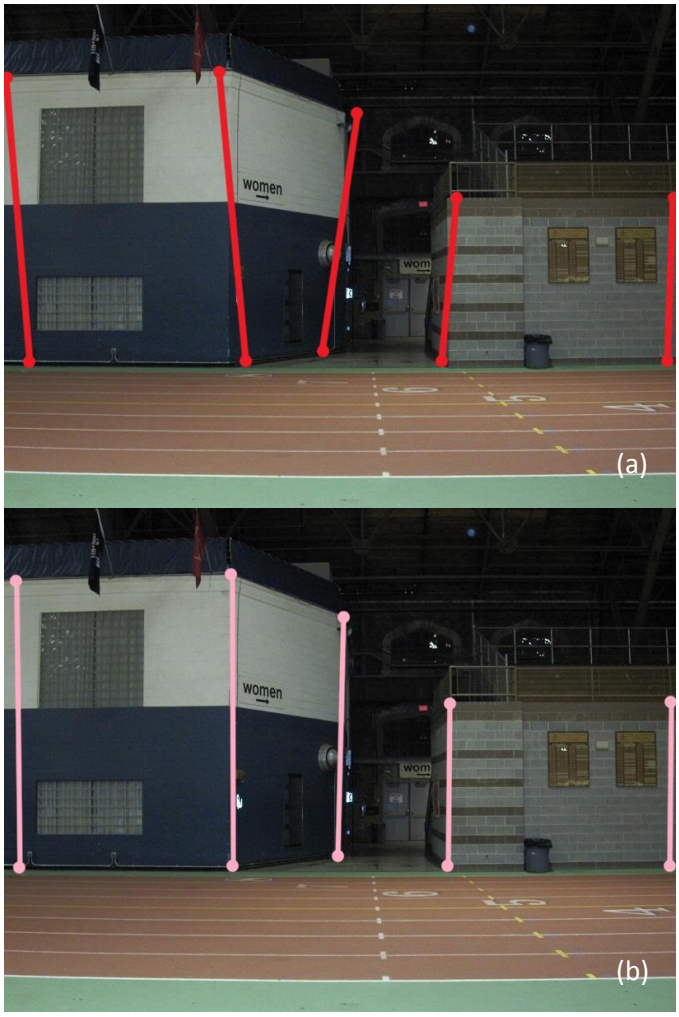

Figure 2a. Shows a user input where the plane boundaries are slightly skewed. Figure 2b. shows the output of our model which corrects the skewed boundaries.

We tested our model on several new environments. Figure 2a shows the original user input for an image and Figure 2b illustrates the model's adjusted mapping of the data points. The output of our algorithm is visibly more vertically aligned.

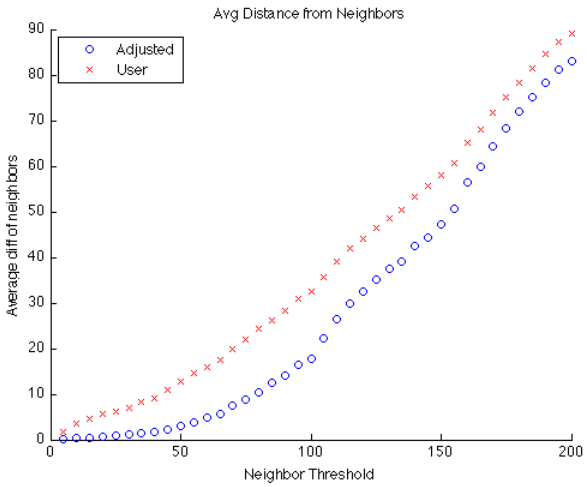

Figure 3. Finding the Optimal Neighbor Threshold.

Figure 3 shows the average horizontal distance between neighboring points in our test set for both user input and our adjusted points. At a *neighborThreshold* of 100 pixels, the users specify points with an average of 32.5 pixels of vertical alignment error, while our algorithm adjusts that to an average of 17.8 pixels of error. This 14.7 pixel improvement represents the largest error reduction our algorithm can achieve. At this optimal neighbor threshold the first variance term $\sigma_1 = 24.2$ and $\sigma_2 = 62.7$. Our adjusted points have an average of 9.7 pixels from truth while the user input has error of 5.7 pixels from truth.

Our adjustment algorithm does slightly move the users' points farther from the truth points on average. The truth points, however, were defined by us and may not perfectly define the boundaries of the image planes. Thus it is more important that the adjusted points are closer to vertical lines as that more accurately defines the plan boundaries. Hence, our algorithm is successful in adjusting user input to create more accurate plane boundaries.

Conclusion

We have developed an algorithm that adjusts user input so that vertices marked in an image are more accurately positioned vertically. Therefore, if a user skews the plane boundaries, our model will readjust the plane boundaries so they are more vertical, given the ground is horizontal. Our model was given a training set of over 500 images each with a series of data points and using this image sets we applied a supervised learning technique to determine two variance parameters. The first variance parameter expresses how far the user tends to draw a point from the true vertices of an image. The second variance parameter examines how much the users tend to skew neighboring vertices so that the plane boundary is slanted. Testing our model on new environments, we observed that our model corrects slanted boundaries by correcting the x-coordinates of points if they are within a specific threshold.

Using the vertices of the image, a 3D model can be constructed by dividing the image into planes based on adjusted user inputs and applying a 3D modeling algorithm such as [6] and [8]; eliminating certain steps such as boundary and edge detection.

Our model has many exciting potential applications. With tablets and touch-screen hand-held devices becoming more and more popular, our model can be integrated with other 3D model algorithms to help generate the final 3D output. With a touch-screen, the user doesn't have to accurately pinpoint the location of the vertices in the image because our algorithm will readjust if necessary. Thus, our work has many promising use for many other applications in vision.

Appendix

Derivation for finding the quadratic programming form of the Gaussian Probabilistic Model (1).

$$\log\left(P(y|d;\,\sigma_1,\sigma_2)\right) = \sum_{i=1}^{M} -\log\left(\sqrt{2\pi\sigma_1}\right) - \frac{(y_i - d_i)^2}{2\sigma_1^2} + \sum_{j\in NE(i)} -\log\left(\sqrt{2\pi\sigma_2}\right) - \frac{\left(y_i - y_j\right)^2}{2\sigma_2^2}$$

$$= \frac{-y^T y + 2d^T y - d^T d}{2\sigma_1^2} + \frac{-y^T \# y + 2y^T (Ne) y - y^T \# y}{2\sigma_2^2}$$

$$= \frac{1}{2} y^T \left(-\frac{I}{\sigma_1^2} + -\frac{2(Ne - NumNe)}{\sigma_2^2} \right) y + \frac{d_x}{\sigma_1^2} y(3)$$

Where # is a diagonal matrix with values corresponding to the total number of neighbors of each point. *Ne* is a neighborhood matrix where cell *(i,j)* index is 1 if the point y_i is a neighbor of y_j. Using (3), we solved for the elements in the quadratic problems subject to $Ax \leq b$ where A is the negative identify matrix and b is zeros vector of size M so that points cannot have negative values.

References

[1] Derek Hoie, Malexei A. Efros, Martial Hebert. “Recovering Surface Layout from an Image.” International Journal of Computer Vision (2007): 151–172.

[2] R. Cipolla, T. Drummond and D. Robertson. “Camera calibration from vanishing points.” BMVC99 (1999): 382-391.

[3] Rosenholtz, J. Malik and R. "Computing Local Surface Orientation and Shape from Texture for Curved Surfaces." Internation Journal of Computer Vision (1997): 149-168.

[4] Li Bing, Xu De, Feng Songhe, Wu Aimin and Yang Xu. "Perceptual Depth Estimation from a Single 2D Image Based on Visual Perception Theory." Advances in Multimedia Information Processing (2006): 88-95.

[5] Antonio Torralba, Aude Oliva. "Depth Estimation from Image Structure." IEEE Transactions on Pattern Analysis and Machine Intelligence (2002): 2002-2015.

[6] Ashutosh Saxena, Min Sun, Andrew Y. Ng. "Make3D: Learning 3-D Scene Structure from a Single Still Image." IEEE Transactions on Pattern Analysis and Machine Intelligence (2008).

[7] Ashutosh Saxena, Sung H. Chung, Andrew Y. Ng. "Learning Depth from Single Monocular Images." Neural Information Processing Systems (2005).

[8] A. R. J. Francois, G. G. Medioni. " Interactive 3D Model Extraction from a Single Image." Image and Vision Computing. (2001): 317-328.