Predicting 3D Geometric shapes of objects from a Single Image

Tamer Ahmed Deif :: Savil Srivastava CS229 Final Report

Introduction

Automatically reconstructing a solid 3D model from a single image is an open computer vision problem. Fully automatic solutions such as Make3d [1] and PhotoPopup [2] are only able to make 2.5D models from images. On the other hand, [3] are able to use manual intervention (namely, drawing edges of the object) and symmetry to reconstruct the 3D shape. This work attempts to use machine learning to learn the solid 3D structure of an object in an image. It assumes objects can be broken into building blocks of cuboids, pyramids, etc. We were only able to manage to investigate for the case of cuboidal-only objects.

Our approach is as follows. We pose the learning of edges of the object in the image as a Linear Regression problem. We use these edges to predict what kind of 3D cuboidal shape could have given the object its particular 2D projection in the image. This is defined as an energy optimization problem, which is solved by gradient descent.

Data Set:

Our data source is Google 3D Warehouse Sketchup models. There are several motivations for that choice, amongst which are:

- The free and abundant availability of data.
- The models represent actual buildings and they are rated by their quality, so realistic models can be chosen.
- The models allow future extension of the project for more complex, as well as non-cuboid bodies.
- Models had texture, which can be used to give the 3D model a more realistic look.
- Google Sketchup allows to manipulate those models and look at them from different angles, as well as, take snap shots from different view angles, which translates into 2D input images for the algorithm.

In order to prepare our Data set, we downloaded 50 models of different buildings from Google 3D Warehouse. We also wrote a ruby script that takes 6 different snap shots of the model from 6 different viewing angles. In total we had 300, different 2D images (each of size 558x404 pixels). Sketchup allows its users to remove texture and shadows from the model. So we exploited these features to take an additional snap shot at each of the 6 positions with only the edges of the model (refer Fig. 1). These extra images represent the ground truth data for the edge detection-learning algorithm.

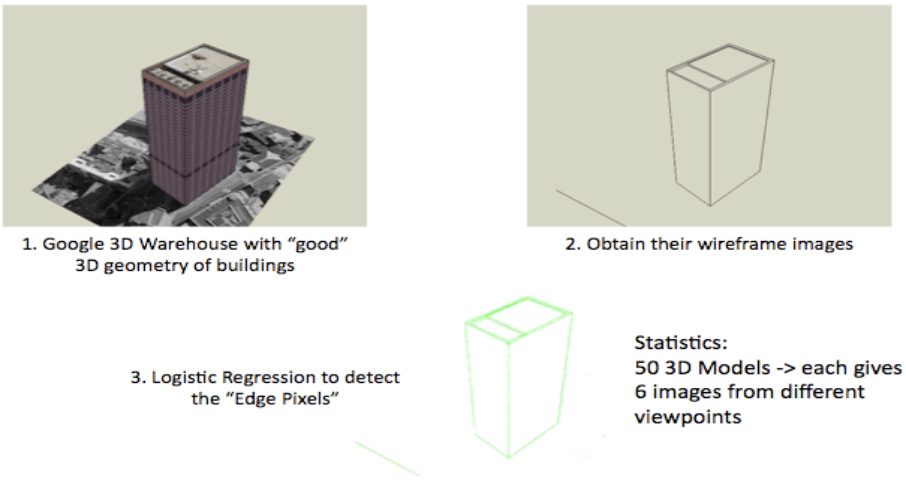

Fig 1: Shows the textured image of the 3D model, the edge map that we generate from sketchup to use as ground truth data, and the edge map we detect after linear regression

Method and Results:

Part 1: *Significant* Edge Detection

We designed features and performed linear regression to determine the geometrical shape edges of the object in the image. In particular, for any pixel we want to classify it as either edge or non-edge.

We selectively sample which pixels to train on, since considering all would result in a large feature matrix with which MATLAB would run out of memory. Currently, we are sampling 90% edge-pixels and 2% of non-edge pixels. Our final sample has a ratio of edge: non-edge pixels of 3:1.

Last, the result of the linear regression gives us predictions on whether the pixel is edge or not as a Real number in the range [0,1]. For each image, we adaptively select what threshold to use, for classifying as edge or non-edge, by choosing the threshold that minimizes the error.

The following table summarizes the average overall error for training, cross validation and testing:

Features/Error %	Training	Cross Validation	Testing
Y, Cb, Cr	6.83	8.37	10.08
Y, Cb, Cr, Hough	6.78	6.85	8.1
Y, Cb, Cr, Hough, Percentage	6.71	8.44	9.6

Features and Results

We consider 101 features for any given pixel. First, we consider a 9x9 pixel neighborhood patch in YCbCr color space. This is the able to capture the difference in Luminance, and blue/red color space. For Luminance, we consider a 9x9 patch and for the color components a 3x3 patch. This smaller patch was considered since MATLAB would run into memory issues while training on this data. The training and test error for this is shown in Figure 2.

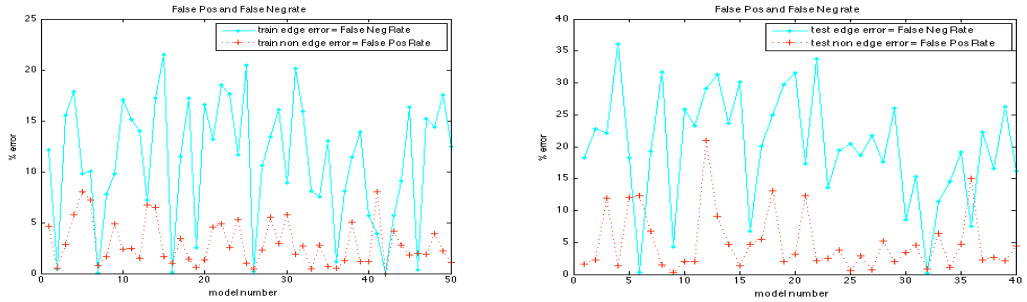

Fig 2: False and positive classifications error for training (to the left) and testing (to the right) data when using Y, Cb, Cr features

We then considered adding two more features, which had a more explicit relationship with edges.
1) Consider whether the pixel lies on a Hough line. For this, a Hough transform was done on each input image and the 8 most likely-to-be lines were selected. Pixels lying these lines were tagged with this feature.
2) Percentage of edge pixels in the 9x9 neighborhood patch.

The results from this can be seen in Fig. 3 and Fig. 4.

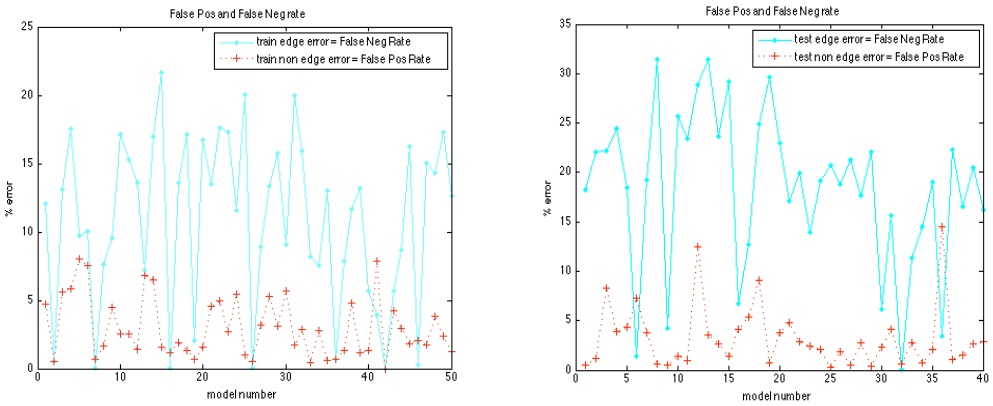

Fig 3: False and positive classifications error for training and testing data when using Y, Cb, Cr and Hough lines features

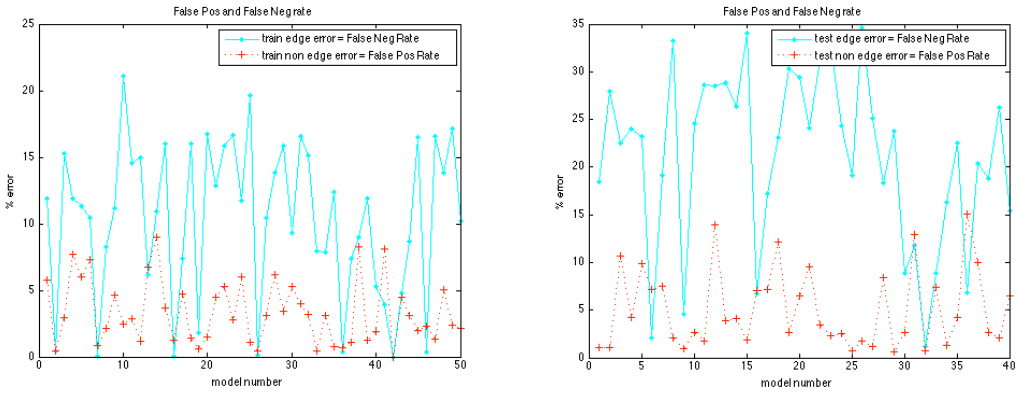

Fig 4: False and positive classifications error for training and testing data when using Y, Cb, Cr, Hough lines and Edge-pixels percentage as the input features

We note that we are most interested in lowering the false positive rate. The false negative rate can be tolerated to be high because (1) the ground truth labels have thick edges (2) losing a few of the edge pixels does not affect the energy function in Part 2 as much as having noisy pixels.

Method Part 2: Cuboid Fitting

In this step, we predict what 3D cuboid could have been projected onto the image to give the object shape (as determined by the edges predicted in Method Part 1). A parameterized 3D cuboid is projected onto the image plane with correct perspective, and we calculate an energy function based on this projection. We generate this energy function by blurring the edge image generated in Part 1 such the maximum energy lies on the actual edge pixels. Then we perform gradient descent on the following function:

$$J_1 = \min_{\psi} \left| E_{true} - E(\psi) \right| \qquad where\ \psi \in R^{10}$$

The problem of optimizing this function suffers from a lot of local minima. Hence what we initialize the parameters our cube to is crucial to success. To mitigate this problem, we run gradient descent initially on 50 different sets of initialization parameters for 5 iterations. We then pick the 10 best models, and perform more iterations. We repeat this funnel-selection method till we get the most optimal model from our 50 initializations. As seen in Fig. 5 below, where the initialization was done automatically, the optimization falls into local minima.

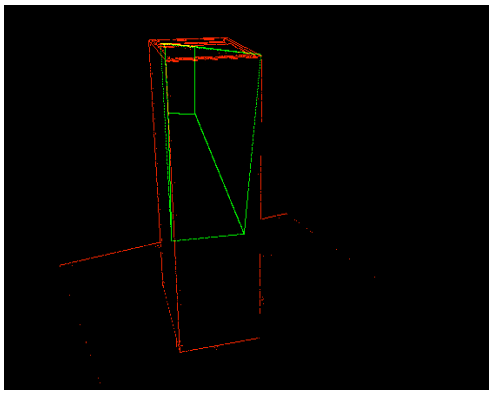

Fig 5: Fitting stuck at a local minimum

However, in some cases, the optimization is able to snap the initialized predicted model onto the predicted edges (Fig. 6).

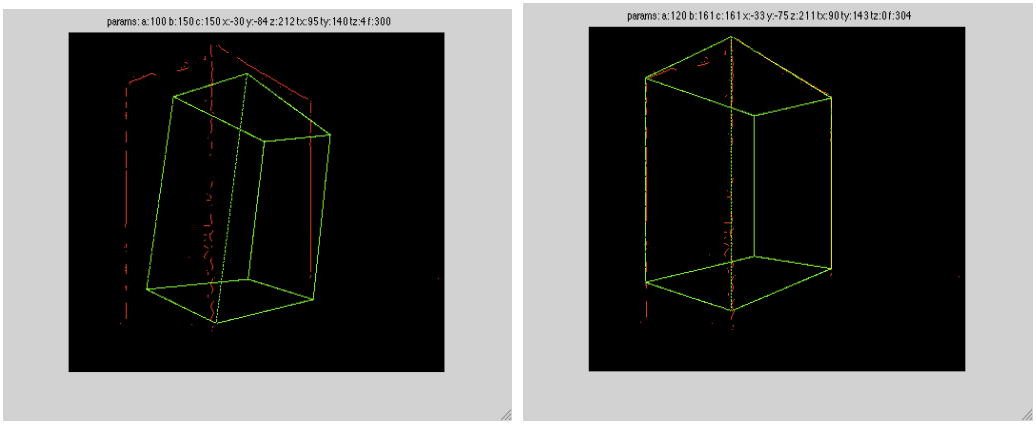

Fig 6: When initialized close to the right position of the building it snaps in and fits accurately

We also tried adding an extra energy term; proportional to the number of edge pixels the cuboid was on. However, in our tests this resulted in less optimal solutions.

Limitations and Future Work

The Cuboid Fitting step fails for a number of reasons:

1) The energy function has a lot of local minima.

A) Currently, when we project the cube onto the image, we draw all the edges of the cube. However, by calculating which side of the cuboid is visible and only projecting the edges of those sides, we could have a better energy function.

B) Once we have a final result, then one can try perturbing it and see whether it snaps back to where it was originally.

2) While the algorithm can predict edges from the input well, it has a false positive rate that can results in quite a few non-edges being classified as edges. This "noise" affects the energy function and even when the cube is initialized such that it should reasonably snap into place.

Furthermore, in the edge-learning phase, the artificial (Sketchup Model) images that we test on have only one object of prominence in the image. For real images, there may be more peripheral objects whose edges also get detected. One would have to design a UI where a user can indicate Region-Of-Interest (ROI) and consider edges only within this ROI.

Acknowledgments

We thank Ashutosh Saxena, Christian Theobalt and Siddharth Batra for their guidance and helpful discussions while planning the project.

References

1. A Saxena, M Sun, AY Ng - Learning 3D Structure from a single still image. Computer Vision, 2007. ICCV 2007. IEEE 11th International ..., 2007

2. D Hoiem, AA Efros, M Hebert - "Automatic Photo Pop-up" Proceedings of ACM SIGGRAPH 2005, 2005

3. PE Debevec, CJ Taylor, J Malik - Modelling and Rendering Architecture from Photographs: A hybrid geometry and imaProceedings of the 23rd annual conference on Computer ..., 1996