Recursive Grammars for Scene Parsing *a.k.a. RANSACing Offices*

Caspar Anderegg (cja58), CS '12
Bill Best (wpb47), MEng '12
with Abhishek Anand

May 15, 2012

Abstract

Our goal was to use recursive grammars to construct parse trees within segmented point-clouds. Similar methods have been successfully used in flat images, but never with 3D data. We have written a virtual scanner to convert Wavefront object files to point-clouds. Additionally, we created a module to segment point-clouds using a generalized RANSAC algorithm. Both were incorporated in a ROS package that we are contributing to the official listing. Our RANSAC module interfaces directly with Abhishek's system for parse-tree construction and labeling.

1 Introduction

Many algorithms for object identification rely on single pixels or small collections of pixels identified as important features. These systems usually do not take into account the relations between features, or the shape of the overall objects. When these systems segment an image, there is generally not a one-to-one correspondence between segments and objects in the scene. Multiple different objects might be included in a single segment, or a single object may be split into many segments. Cognitive science research in human perception has shown that there is ongoing feedback between recognition areas and segmentation areas of the brain.

After splitting an object, we use a recursive grammar to merge segments. Atomic segments are merged into contiguous components (for example a plane forming the front of a computer tower), these components will be merged into complete objects. Such a segment hierarchy allows for a wholre new class of features to be used in object identification.

2 Related Work

Previous projects [Girshick, R., P. Felzenswalb, and D. McAllester. 2011] and [Y. Zhao, S.Z. 2011] have successfully used recursive grammars for object recognition in two dimensional images. These projects generated many possible candidate parse trees, then used maximum likelihood estimation to select a most probable candidate given the scene. Neither project attempted to use 3D data.

Abhishek Anand and Hema Swetha Koppula have done work on scene understanding from Kinect point-cloud data. They are able to use local visual appearance and shape cues to label objects in home and office scenes. Abhishek has created a graphical interface to simplify the graphical construction and labeling of parse trees, given segmented point-clouds. As part of the project, we interface with this existing software.

3 Approach

3.1 Data Acquisition

One of the major challenges was data acquisition. We wanted to use digital models from the Google Sketchup 3D Warehouse, converted into point-clouds. These files are downloaded in a proprietary Sketchup format, but can be exported into a Wavefront Object Files (`.obj`) and Material Libraries (`.mtl`). We had intended to find a ROS package that would convert `.obj` files into Point-cloud Library `.pcd` files, but were unable to find a suitable converter. The best we could find was a python script that simply stripped out face information and left vertices.

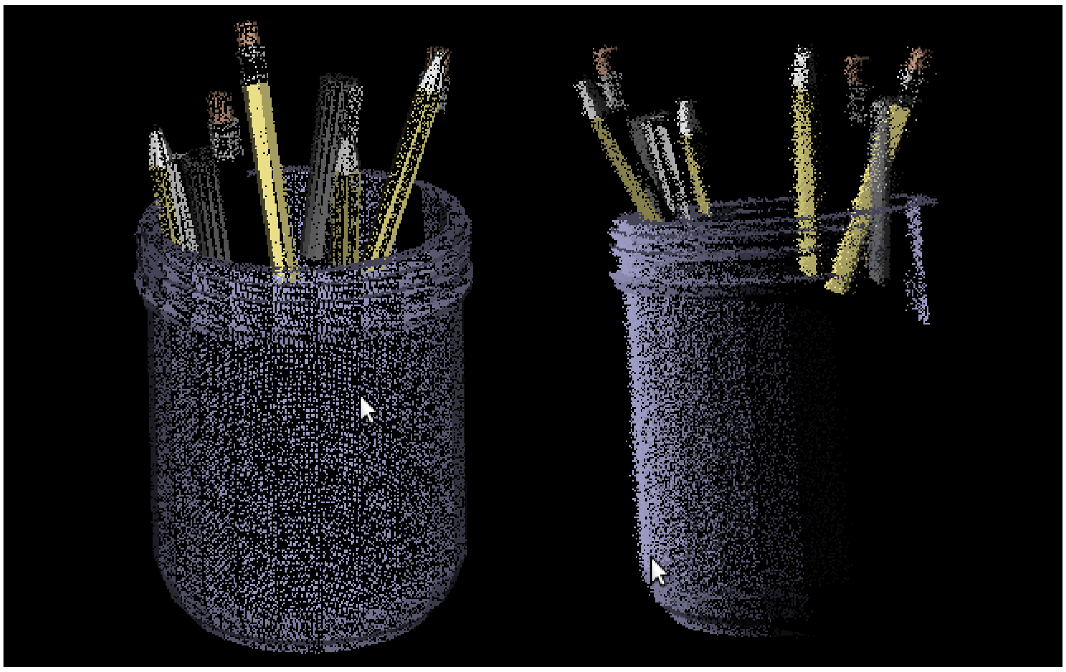

Figure 1: A synthetically generated point-cloud (from several angles). The cloud was rendered from model taken from Google Sketchup 3D warehouse.It uses all three occlusion models, and

Thus, we wrote a virtual scanner to convert `.obj/.mtl` files into `.pcd` files. The scanner used barycentric coordinates to sample each face. The view direction, sample rate, and and other parameters can be set via command line arguments.

The scanner supports several noise models: no noise, Gaussian noise, and Kinect camera noise as described in [Liu, Yiping. 2012]. The Kinect noise model is one dimensional Gaussian noise (in the direction of the view vector) that scales proportional to the distance from the camera. Where $\vec{v}$ is the view direction, $\vec{e}$ is the location of the eye, and $\vec{p}$ is the location of the rendered point in space, the noise follows the equation below:

$$Noise \sim \mathcal{N}(0, 3.5 \times 10^{-3} \cdot ||proj_{\vec{v}}(\vec{e} - \vec{p})||^2)\vec{v}$$

The scanner also supports three levels of visibility culling. At the most basic level, backface culling allows the removal of any faces with normals that point away from the camera. The next level up is occlusion culling, allowing faces to be removed if none of their vertices are visible to the camera. The highest level is trace culling, which checks whether each point is visible to the camera. These occlusion models can be used together to choose an appropriate level of speed and realism.

We have downloaded almost 200 3D models in five categories of office objects. These can be converted to point-clouds as needed, using different perspectives and noise profiles. An example point-cloud generated by our virtual scanner can be seen in Figure 1.

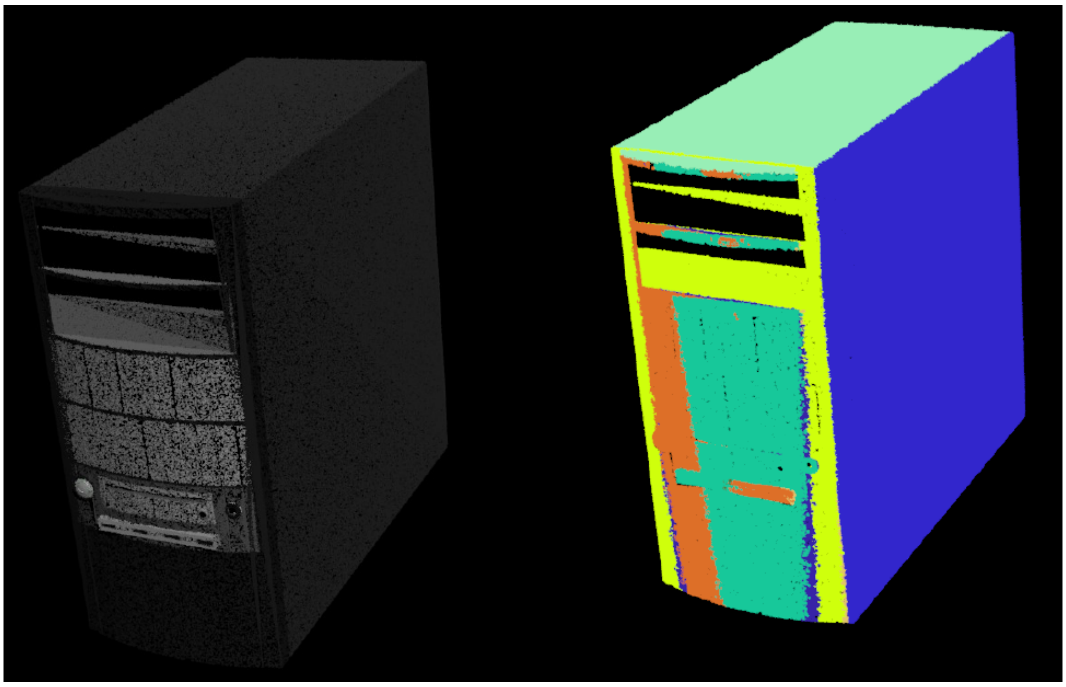

Figure 2: Two point-clouds of a computer tower. The image on the left shows a point-cloud generated from our virtual scanner. The image on the right shows the same point-cloud after being segmented by our generalized RANSAC algorithm.

3.2 Point-cloud Segmentation

We wanted to run a generalized RANSAC segmentation on these point-clouds, as outlined in [Schnabel, Ruwen, Roland Wahl, and Reinhard Klein. 2007]. This general RANSAC needed to be able to fit multiple shapes (planes, spheres, cylinders, etc.). We hoped to find a ROS package that would implement such a generalized RANSAC, but again we were unable to find a suitable package. All of the implementations we found fit only planes.

We created a ROS package to run a generalized iterative RANSAC segmentation of a point-cloud file. The algorithm generates a tree of candidate shape combinations, then performs a breadth-first search of this tree to identify the smallest set of shapes to fit some critical majority of the points in the model.

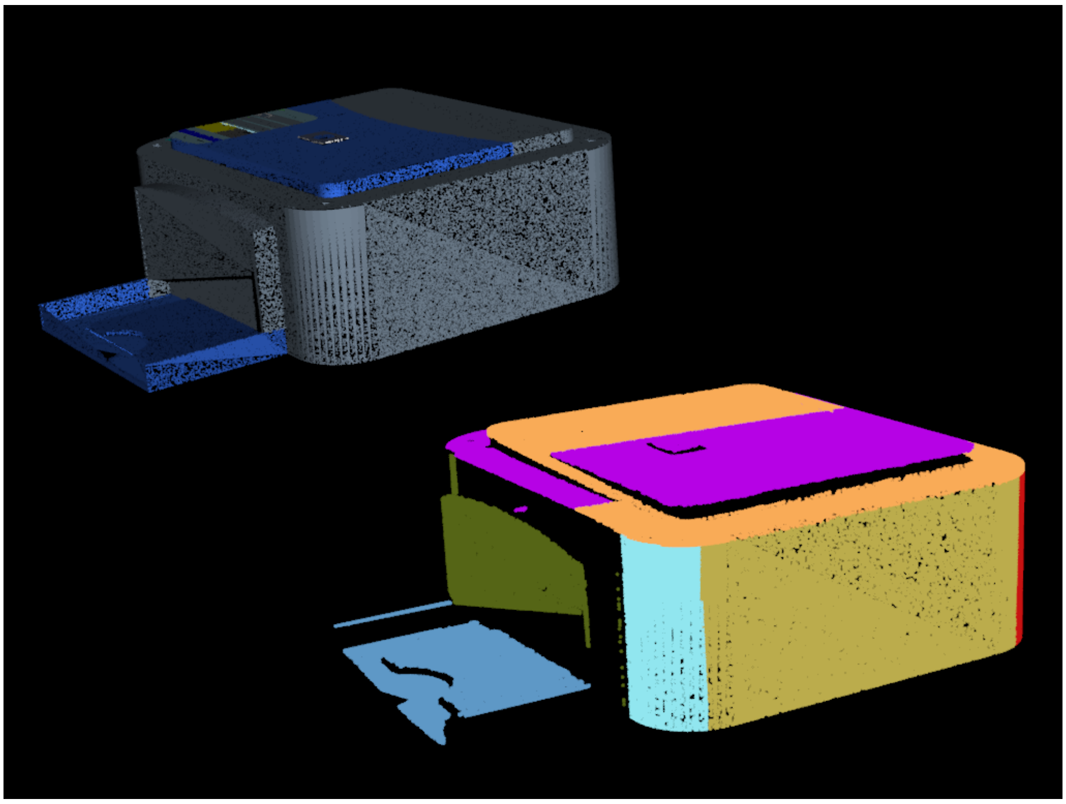

Figure 3: Two point-clouds of a computer printer. Again the left-hand side is the generated point-cloud, and the right-hand side is the segmented cloud. Note the rounded corners have been fit with cylinders.

For our RANSAC algorithm, we used three basic shapes: planes, spheres, and cylinders. Planes and spheres were already available to us as functions in PCL for random consensus, so implementing them was relatively simple. We decided not to implement shapes such as rectangular prisms since they would already be representable by multiple planes and, given the number of degrees of freedom in a rectangular prism, would take too long to converge on a final answer. We selected cylinders as the third shape because we found that after planes, the next most common shape was cylinders. While a cylindrical shape in the model could be represented by multiple planes, we elected to take the route of fitting a cylinder instead. While fitting a cylinder takes longer (due to the number of degrees of freedom in a cylinder) we were trying to fit fewer shapes overall, so fitting a cylinder instead of multiple planes was the better option. Since we were working with a specific subset of possible objects — office objects — we were able to notice that shapes other than planes, sphere, or cylinders were rare enough that we could easily represent them as multiple shapes rather than as their true shape, without significant loss of time or final efficacy in Abhishek's code.

There are some useful shapes that are currently not supported by PCL (the cone and the torus), but which are scheduled to be implemented at some future date. Our code contains a framework that allows these shapes to be easilly included once they are implemented.

3.3 Labeling and Parse-tree Construction

For parse tree construction and labeling, we interface with Abhishek's system. The RANSAC module outputs a segmented point-cloud, a neighbor map between segments, and an initial parse tree. These files can be passed directly into Abhishek's `cfg3d ParseTreeLabelerF` for parse tree construction and labeling.

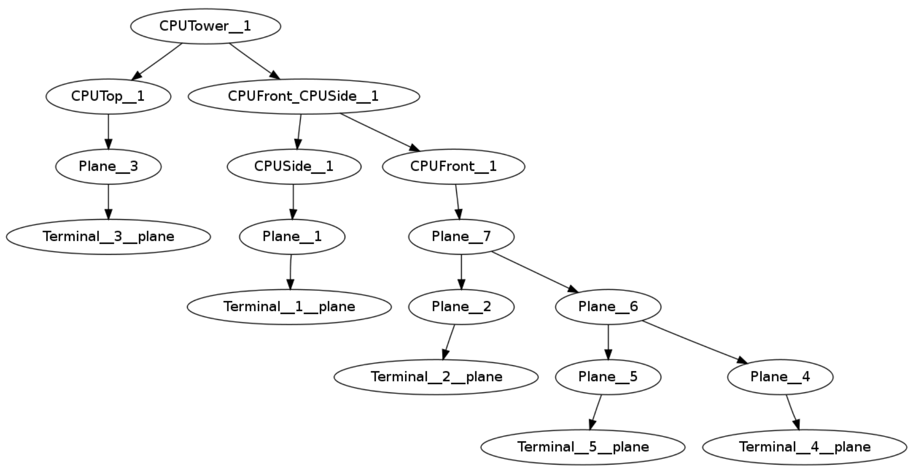

Figure 4: An example parse tree created from the tower shown in Figure 2.

The constructed parse trees are output in `.dot` format. This allows parse trees to be build or edited by hand if required. For an example parse tree constructed with the labeler, see Figure 4. The grammar extracted from the parse tree might look as follows:

ROOT → CPUTower
CPUTower → CPUTop CPUFront_CPUSide
CPUFront_CPUSide → CPUFront   CPUSide
CPUTop → Plane
CPUFront → Plane
CPUSide → Plane
Plane → Plane `plane_terminal` | `plane_terminal`

(For convention, nonterminal symbols begin with uppercase letters while terminals are teletyped and begin with lowercase letters.)

3.4 ROS Package and Dataset

As part of the project, we created a ROS package that we are in the process of getting on the official ROS listing. We also created a synthetic data set with over 1000 point-clouds rendered from almost 200 Sketchup objects.

4 Results

We performed trials to see how many shapes the RANSAC module used to fit different classes of objects. Results can be found in the table below.

Type	**Average Shapes**
Keyboards	3.61
Computer Mice	2.89
CPU Towers	6.30
Computer Monitors	5.63
Printers	7.47

In the RANSAC module, we found the percentage of points that were not fit by the RANSAC model (the number deemed outliers) was on average 3.70%. Results were computed with a small threshold and outlier cutoff value, for detail on a single object. A larger threshold and and outlier cutoff value would give detail closer to that of a full scene.

5 Conclusion

While we were not able to accomplish everything we had planned due to time restrictions, overall we found relatively good results. As noted above, we were able to reduce the number of points not associated with a shape to about 3.7% per object. We feel this is a reasonable allowance of error for the needs of the project going forward. Further, we were able to produce point-cloud sets with a minimal number of constituent shapes. For example, a keyboard is comprised of (on average) 3.61 shapes. This would be a plane for the front edge, a plane for the top of the body, a plane for all the keys, and sometimes an extra plane depending on the angle. This is basically the minimum number of shapes that could be created for the object. We see this to be common among all objects. Having a minimum number of shapes is important because it allows us to get a more accurate representation of what the object is comprised of, more needless shapes would be akin to having more noise.

We were also able to successfully coordinate our code with the current labeling code. The RANSAC program allows us to associate more shapes to the objects we want to label, giving us a deeper understanding of the objects themselves, as well as the scene they are in.

Finally, we were also pleased with our model scanner, which allowed us to easily create PCD files from obj files. It produced accurate point-cloud images to which we were able to add noise to simulate Kinect noise. This program should allow for production of PCD files from multiple angles, without spending all the time capturing the images with a real Kinect.

We have combined these in a ROS package, that we hope to have on the official listing soon.

6 Future Work

Going forward, we need to create a dataset of parse trees using the labeler. Using this dataset, we could extract a grammar of shapes and relations between shapes which we can train using an SVM (similar to SVM-cfg by Thorsten Joachims). From this, we hope to be able to RANSAC an entire office and label all objects in it with a reasonable degree of accuracy.

Even more exciting, we would be able to parse relations between objects themselves. This would allow an even deeper understanding of the composition scene.

7 Acknowledgements

We would like to thank the following individuals:

Ashutosh Saxena, course professor CS 4758 — Cornell University

Abhishek Anand, Ph.D. mentor — Cornell University

Ken Conley, ROS Liaison — WillowGarage

References

Anand, Abhishek, Sherwin Li, and Paul Heran Yang. *Understanding 3D Scenes Using Visual Grammars.* Cornell University, Ithaca: 15 December 2011.

Girshick, R., P. Felzenswalb, and D. McAllester. *Object Detection with grammar models.* NIPS: 2011.

Liu, Yiping. Precision of the Kinect Sensor. *ROS Wiki.* 2011. Web. 2 March 2012.

Schnabel, Ruwen, Roland Wahl, and Reinhard Klein. *Efficient RANSAC for Point-Cloud Shape Detection.* Computer Graphics Forum: 2007.

Y. Zhao, S.Z. *Image parsing with stochastic scene grammar.* NIPS: 2011