Modeling and Packing Objects and Containers through Voxel Carving

Alex Adamson aadamson@stanford.edu
Nikhil Lele nlele@stanford.edu
Maxwell Siegelman maxsieg@stanford.edu

Abstract

This paper describes a system which attempts to determine whether it is possible to pack a set of objects into a container. This is done by using multiple photos of each object on a calibration grid to generate 3D models of the objects and containers plus a bin packing algorithm to find a working arrangement of the objects. We found that by using the CIELab colorspace it was possible to use K-Means clustering to reliably silhouette many pictures of the object with minimal user interaction, which allowed for the creation of reasonably accurate voxel models through carving. We also found that the number of frames used for each object impacted the size of the voxel models and the efficiency of the bin packing algorithm. More frames made the number of voxels in the objects decrease and thus allowed more of them to be packed. It also decreased the number of voxels in the container, which decreased the number of objects that the algorithm said could be packed.

1. Introduction

We create a semi-supervised system to create 3D models from photographed objects and containers, and then determine how to fit the objects in the containers. For example, one could take pictures of toys and pictures of a box and the system would determine if those toys could be fit into the box. Our method relies on determining the pose of the camera (camera parameters), silhouetting the object from the background, creating a 3D model from the photos, and determining the way to pack the objects.

1.1. Review of Previous Work

Previous work in image segmentation and silhouettes use a variety of different methods. For example, there are simple threshold methods, clustering methods using K-Means, EM, and mean shift [4] [7], the watershed method [2], and graph based methods such as normalized cut [9].

There are a number of methods to do 3D model generation from views. These include space carving for voxel models [6] and mesh acquisition from stereo imagery [8].

Bin packing is a problem with implications for a variety of industries, so a number of heuristics exist. The heuristic we chose to base our algorithm on moved from the bottom of the container to the top, creating levels and trying to fill each level with objects of the appropriate size [3].

We were also inspired by the project from March 2014: "Counting jelly beans: voxel carving and segmentation of a container of heterogeneous objects." [5]

1.2. Key Contributions

Our first key contribution is showing that a system combining silhouetting, voxel carving and a packing heuristic with minimal supervision can output reasonable estimates for how many objects can be packed into a container as well as arrangements of a variety of objects in a container. An important sub-point of this contribution is the usage of k-means along with the CIELab colorspace and minimal human interaction in order to quickly form silhouettes from input photos. Another key contribution from this paper is the investigation of how the number of frames used as input effects the results of the process, specifically the bin packing efficiency.

2. Technical Solution

2.1. Determining Camera Pose

In order to extract the camera extrinsics for each viewpoint around an object and to get the camera intrinsics, we use Matlab's camera calibration toolbox and then transform the parameters so that we can left-multiply world coordinates by the projection matrix in order to recover image points. This necessitates having the camera calibration grid in each picture. The camera calibration toolbox assumes that we are right-multiplying:

$$\begin{aligned} x^T &= X^T P^T \\ &= X^T \begin{bmatrix} R^T \\ T^T \end{bmatrix} K^T \end{aligned}$$

so in order to be able to recover image points via left-multiplication, we simply take the transpose of all the com-

ponent matrices:

$$x^T = X^T \begin{bmatrix} R^T \\ T^T \end{bmatrix} K^T$$
$$\Rightarrow x = (X^T \begin{bmatrix} R^T \\ T^T \end{bmatrix} K^T)^T$$
$$= K \begin{bmatrix} R & T \end{bmatrix} X$$

2.2. Silhouette Generation

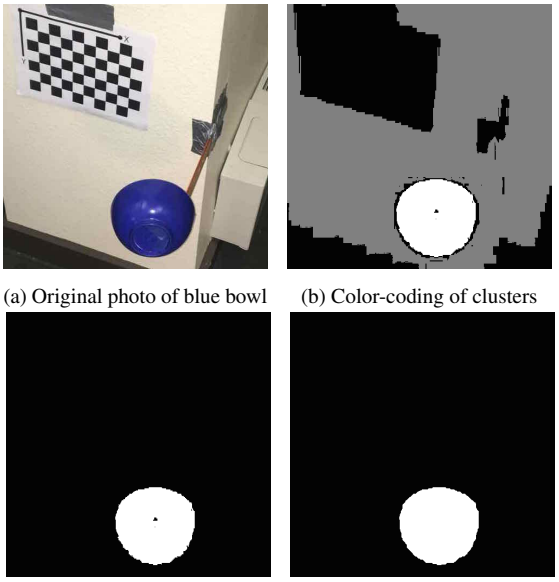
Our method to mask out the silhouette of the object relies on having a contrasting color with the background. We use the K-Means algorithm, which iteratively partitions data by assigning every data point to one of K clusters, where K is chosen before hand. In our case, the data points are every pixel in the image, and the clusters correspond to the most prominent colors in the image.

To mask out the silhouette of the object, we select all pixels that correspond to a particular cluster chosen by the user. For example, for a blue bowl we select the pixels from the cluster with the most blue centroid. See figure 1b where each pixel is colored black, gray, or white corresponding to which cluster it has been assigned to. Then in figure 1c we mask out only the pixels from the most blue centroid. Note: We choose a low value of K such as 3 or 4 to ensure that we end up with a conservative estimate of the silhouette that will later be carved down by other views.

Lastly we refine the mask by morphologically closing the holes in the image with circles of a small radius. We use the Matlab `imclose` function, which uses methods based on morphological reconstruction [10]. See figure 1d.

2.2.1 CIELab Color-Space

Before finding the silhouette, we first convert the image from RGB color-space to the CIELab color-space. The CIELab color-space consists of a lightness axis 'L' and two color channels 'a' and 'b'. The 'a' channel runs from green to red and the 'b' channel runs from blue to yellow [1]. See figure 2 for a visualization of the axes of the color-space. This color-space is more useful for separating images on the basis of color because it more faithfully represents the way humans see color at the expense of not corresponding to pixels on our screens. Grays, browns, and other dull colors are harder to reason about in CIELab because they all lie near the origin, but our example objects have stronger colors that are especially suited to this color-space. In addition, because the lightness value is orthogonal to the color information, we can choose to only use the color axes for K-Means. This makes the clustering robust to changes in lighting and shadows.



(a) Original photo of blue bowl (b) Color-coding of clusters

(c) Mask of just the blue bowl (d) Refining mask with infilling

Figure 1: Pipeline for silhouette acquisition using K-Means and infilling.

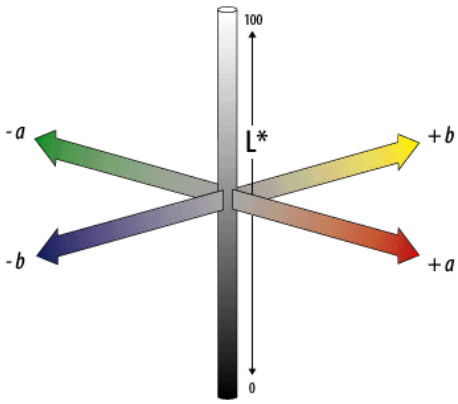


Figure 2: Axes of the CIELab color-space [1]

In figure 3 we can clearly see that in RGB space and with the full CIELab space that the blue bowl is in the same cluster as parts of the checkerboard and the wall-mounting apparatus. By removing the lightness value and only considering the color channels 'a' and 'b', we are able to easily separate the blue bowl from the background and distinguish it from the other items in the picture.

2.3. 3D Model Generation

For 3D model generation, we use a standard voxel carving approach. Given a set of tuples F where $F_i = (P_i, S_i, I_i)$ containing the camera projection matrix P_i, the silhouette of the object S_i in the image, and the image from

(a) K-Means in RGB space (b) K-Means using full CIELab space (c) K-Means without lightness from CIELab space

Figure 3: Comparing silhouette quality using different color-spaces. K-Means using only the color channels from CIELab space performs much better than either using RGB space or using the full CIELab space.

the ith viewpoint I_i, we form a voxel model for the object as follows: Using the camera parameters and silhouette of the first tuple (here we assume that all images contain the entire object within their frame), we construct an initial bounding voxel prism. On successive iterations, for each voxel that has not been cut out yet, we remove it from the carving either if it projects outside of the frame of the image (since we are assuming that for each image, the entire object is contained in the frame) or if it projects into a pixel in the silhouette that was not considered part of the object by the silhouette-forming algorithm.

function CARVE(c, P, S, I)

$X \leftarrow P * c$
$packings \leftarrow 0$
for $X_i \in X$ **do**
  if $S(X_i) = 0$ **then**
    $c -= c_i$
  end if
  if $X_i \notin I$ **then**
    $c -= c_i$
  end if
end for
return c
end function

Our use of a straightforward voxel carving algorithm is justified by the fact that we are assuming that our containers are hollow and hence we in general do not care about concavities since they will not affect how many objects we can place in them later in the pipeline. Additionally, our bin packing algorithm is not advanced enough to allow objects within the container to interlock within each other's concavities, so there is no potential gain from the perspective of getting more efficient packings (less approximation error) by improving the carving routine.

2.4. Bounding Boxes

It greatly simplified the bin packing algorithm to put the object models in bounding cuboids rather than trying to pack them immediately. While algorithms exist which find the smallest bounding box of an object using the convex hull of the object's point cloud, we decided it would be prudent to save time on this aspect and simply bound the voxel model by its smallest and largest x, y and z coordinates. This approach results in some inefficiency, both because a bounding box is strictly larger than the object it bounds and because the boxes we chose to use were not minimal, but it provides sufficiently good performance to allow us to examine the viability of our overall setup as well as how changing our silhouetting algorithms and their inputs affects the efficiency of our bin-packing algorithm.

2.5. Bin Packing

The bin packing algorithm we used greedily attempts to fill the container with the objects, at this point surrounded with bounding rectangles, by going from the bottom of the bin to the top, filling one level at a time. All of the bounding rectangles are first sorted by their z coordinates, then placed into the container. The first location where a fit is attempted is determined by first finding the smallest z coordinate in the container, then the smallest y coordinate with that z coordinate, then the smallest x coordinate with that y coordinate and z coordinate. Then the objects are then placed from largest z coordinate to smallest, trying to use the initial z coordinate for as long as possible by fitting objects in the x direction and then in the y direction. Once no more objects can be fit on that level a new level is started based on the z coordinate of the first object placed (since the first one had the largest z coordinate) and the process starts again. This heuristic is fairly simplistic and could certainly be improved upon, for example by trying to fill lower levels with objects rather than giving up on them. However, just like in the case of the bounding boxes we determined that writing a highly complex packing heuristic was not necessary to examine the

viability of our overall setup or for evaluating the impact of our silhouetting algorithms on efficiency. This bin packing heuristic will work for any convex container, but it works best for containers that are close to being boxes. The heuristic will report whether it found a workable arrangement for a set of objects and a container, or with minor tweaking can also report how many of a single kind of object it can fit in a container.

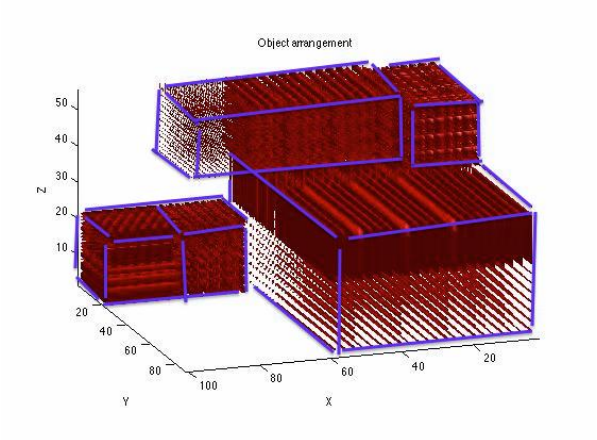


Figure 4: An example packing result, showing bounding boxes

3. Experiments

3.1. Experimental Setup

To analyze the effectiveness of our pipeline, we run it from start to finish on images taken of a small plastic bowl and a macadamia nut cluster and compare how many macadamia nut clusters our algorithm can pack into the bowl compared to how many we can pack into the bowl manually. We seek to analyze how the number of clusters that can be packed into the bowl by our bin packing algorithm scales with the number of frames used to carve the bowl and the number of frames used to carve the macadamia nut cluster.

Intuitively, as we add more frames to carve the bowl, we should be getting a progressively tighter upper bound on the actual expanse of the bowl, so increasing the number of frames used to carve the bowl should decrease strictly (assuming we are only adding frames, not removing and adding) decrease the number of clusters that we can pack. On the other hand, increasing the number of frames used to carve the nut cluster should also progressively slim the output voxel carving to the expanse of the real cluster, so using more frames to carve the nut cluster should strictly increase the number of clusters that our algorithm can fit into the bowl.

We began with 20 images of the macadamia nut cluster and 16 images of the small bowl, all with the calibration grid in the background. The camera angles are distributed roughly evenly on a spherical cap around the object (our rig involved mounting the object on a small pole that extended about a foot from a wall, so we were obstructed by the wall from taking pictures from the reverse). We construct silhouettes of the respective object from all images and store them. Then, to find the number of objects that we can pack using carvings formed from n frames of the bowl and m frames of the nut cluster, we sample as follows:

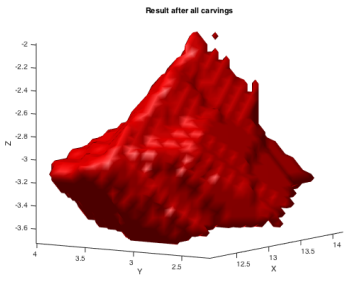
function MEANNUMBERPACKINGS($c_0, iters, f_n, f_b$)
▷ c_0 is the initial voxel box
▷ f_n is an array containing the frames (silhouettes and camera parameters) of the cluster
▷ f_b is an array containing the frames (silhouettes and camera parameters) of the bowl
  $packings \leftarrow 0$
  for $iter \in [1, iters]$ **do**
    $c_n \leftarrow c_0$
    $c_b \leftarrow c_0$
    for $i \in$ SAMPLEW/OREPLACEMENT$(20, m)$ **do**
      $c_n \leftarrow$ CARVE$(c_n, f_n(i))$
    end for
    for $i \in$ SAMPLEW/OREPLACEMENT$(16, n)$ **do**
      $c_b \leftarrow$ CARVE$(c_b, f_b(i))$
    end for
    $packings$ += NUMPACKINGS(c_b, c_n)
  end for
  return $packings/iters$
end function

3.2. Results

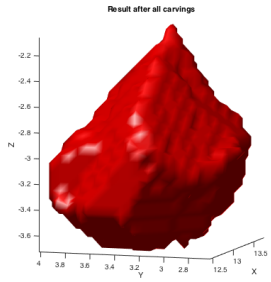
Figure 5 shows a sample progression of the carving as we use successively more frames to carve the cluster. Of particular note is that for a low number of frames, the oblong protrusion along the top of the cluster is especially large and the base of the cluster is broader and flatter than it is in reality. Figure 7 shows the effect on the volume that using additional frames has on the number of voxels: moving from four to eight frames eliminates over a quarter of the voxels used for the cluster. The effect on the deformations of adding the frames is demonstrated visually in figure 5: adding the additional frames substantially smooths out the base and shaves away the protrusion along the top.

Number of sampled frames	Mean number of voxels
4	3982.8
8	2909
14	2285

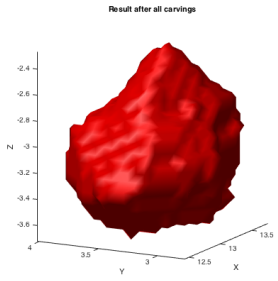
Figure 7: Voxels as we vary the number of frames used to form the carving for the macadamia nut cluster



(a) Example carving of the macadamia nut cluster using four frames

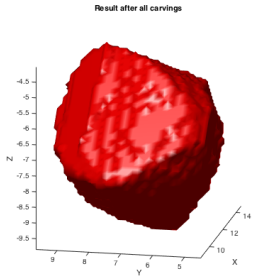


(b) Example carving of the macadamia nut cluster using eight frames

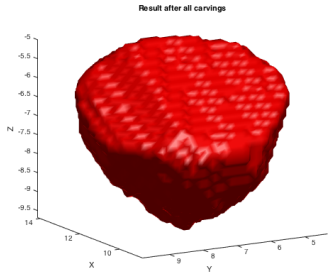


(c) Example carving of the macadamia nut cluster using fourteen frames

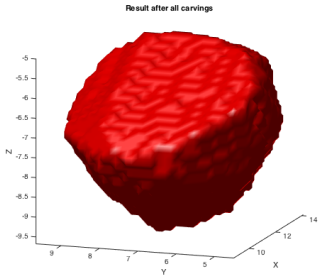
Figure 5: Carvings using sampled subset of images of macadamia nut cluster



(a) Example carving of the small bowl using four frames



(b) Example carving of the small bowl using eight frames



(c) Example carving of the small bowl using twelve frames

Figure 6: Carvings using sampled subset of images of small bowl

We should expect that refining these protrusions would have an out-sized impact on the number of objects that we are able to pack using our bin packing algorithm since we effectively consider the bounds of the object to be its extent in any direction. This effect can be seen in figure 8: as we increase the number of frames used to carve the macadamia nut cluster, the number of clusters that we can fit in the bowl increases without clear diminishing gains regardless of the number of frames used to carve the bowl.

	4	8	12
4	2.38	1.58	1.58
8	2.88	2.33	2.08
14	3.50	3.17	2.67

Figure 8: Number of clusters that our algorithm is able to pack into the bowl. Number of frames used to carve bowl varies over columns, number of frames used to carve cluster varies over rows

We can contrast with the effect that using more frames to carve the bowl has on the number of objects that we are able to pack into the bowl. Looking at figure 8, when using four or eight frames to carve the cluster, there is a clear limit to the gains (i.e. decrease in clusters packed) made from using additional frames to carve the bowl. This can largely be accounted for by the simplicity and lack of deformations present in the bowl. Looking at figure 6, we see that the carving produced by using twelve frames is nearly identical to that produced by using eight frames. There simply are not any interesting features on the bowl that necessitate the use of additional frames to get a proper carving of. This is reflected in figure 9: Adding additional frames only slightly reduces the number of voxels that we use to represent the bowl.

It is also a noteworthy result that our silhouetting algorithm never caused a failure of the voxel carving method by not marking enough of a picture as part of the object, regardless of the number of frames used.

Number of sampled frames	Mean number of voxels
4	6250.3
8	5677
12	5320

Figure 9: Voxels as we vary the number of frames used to form the carving for the bowl

4. Conclusions

We create a semi-supervised system that uses photographs of objects and containers and determines how many objects can be packed.

First we use a calibration target in each photograph to determine camera parameters. Then by using K-Means in the CIELab color-space without lightness values, we create very robust silhouettes that are improved by infilling. From these silhouettes we use voxel carving to create a 3D model that is iteratively improved. Lastly we use a simple bounding-box method to greedily pack the 3D models of the objects inside the container.

We found that more complex objects such as the macadamia nut cluster require more photos to create a tight model while simple objects like the bowl show that there are diminishing returns from more photos of the object.

Future work could greatly improve usability of the system by providing a GUI to select the color of object to be silhouetted, or even have the program determine which cluster to use automatically. Ideally, the method would be made to not require the calibration checkerboard or necessitate the user taking lots of photos. Also, the bin packing system can be greatly improved to allow for rotations of objects in the containers as well. If it is improved to allow for objects' concavities to intersect, then perhaps voxel carving will be too simple of a method to generate the 3D models.

References

[1] Adobe Systems Incorporated. Adobe technical guides, 2000.

[2] S. Beucher and F. Meyer. The morphological approach to segmentation: the watershed transformation. *OPTICAL ENGINEERING-NEW YORK-MARCEL DEKKER INCORPORATED-*, 34:433–433, 1992.

[3] J. Białek. Packaging optimization in supply chain. *Logistyka*, (6), 2014.

[4] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, 2002.

[5] A. Cope and M. O'Meara. Counting jelly beans: voxel carving and segmentation of a container of heterogenous objects. CS231A Final Project Report.

[6] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, 2000.

[7] D. Liu, B. Soran, G. Petrie, and L. Shapiro. A review of computer vision segmentation algorithms.

[8] L.-P. Morency, A. Rahimi, and T. Darrell. Fast 3d model acquisition from stereo images. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, pages 172–176. IEEE, 2002.

[9] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[10] P. Soille. *Morphological image analysis: principles and applications*. Springer Science & Business Media, 2013.